PROVENANCE AND UNCERTAINTY

Sudeepa Roy

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2012

Supervisor of Dissertation	Supervisor of Dissertation
Signature ______________	Signature ______________
Susan B. Davidson, Professor	Sanjeev Khanna, Professor
Computer and Information Science	Computer and Information Science

Graduate Group Chairperson

Signature ______________

Jianbo Shi, Professor, Computer and Information Science

Dissertation Committee

Val Tannen, Professor, Computer and Information Science (Chair)

Andreas Haeberlen, Assistant Professor, Computer and Information Science

Zachary G. Ives, Associate Professor, Computer and Information Science

Tova Milo, Professor, School of Computer Science, Tel Aviv University (External)

Christopher Re, Assistant Professor, Department of Computer Sciences,

University of Wisconsin-Madison (External)

To

my parents and my husband

Acknowledgements

This thesis would not have been possible without the generous help and support from my professors, colleagues, friends, and family. I humbly extend my heartfelt gratitude to all of them, and sincerely apologize to any of them who I forget to name here.

First and foremost, I thank my advisors Susan Davidson and Sanjeev Khanna, who taught me how to do research: right from selecting problems to improving the written and oral presentation of results. Their doors were always open in spite of their incredibly busy schedule, and I was always fortunate to get their support in academic matters and beyond.

I thank my dissertation committee members Andreas Haeberlen, Zack Ives, Tova Milo, Chris Re, and Val Tannen, who graciously agreed to serve on the committee. They helped me improve the presentation immensely with thoughtful and objective suggestions right from when I submitted my thesis proposal.

The work presented in this dissertation is a result of an extensive and fruitful collaboration with my advisors, and also with Laura Chiticariu, Vitaly Feldman, Tova Milo, Debmalya Panigrahi, Vittorio Perduca, Frederick Reiss, Val Tannen, and Huaiyu Zhu. Their expertise in a diverse set of areas like algorithms and theory, databases and systems, and machine learning enriched my knowledge and broadened my interests as a computer scientist. I gratefully acknowledge their contributions in realizing this dissertation.

I sincerely thank Val Tannen, who has been a close collaborator, a mentor, and an informal advisor to me. Val gave me far more attention and advice than I could have asked for, and patiently suggested ways to overcome my shortcomings and consolidate my strengths. My thanks to Tova Milo who was a wonderful collaborator and great source of support. My research experience would not have been the same if I did not have the opportunity to work with Tova and Val, who mentored me, made each research meeting tremendously enjoyable, and helped me get a toe-hold in the academic community.

I was fortunate to be mentored and advised by a number of eminent researchers from different fields. My heartfelt thanks to Laura Chiticariu, Julia Chuzhoy, Sudipto Guha, Zack Ives, Sampath Kannan, Benny Kimelfeld, Wang-Chiew Tan, and Cong Yu, whose every bit of advice enhanced my experience as a graduate student. My sincere gratitude to my advisors, Manindra Agrawal and Somenath Biswas, at IIT Kanpur who encouraged me to join a doctoral program in the first place.

I thank all the members of the vibrant database group at Penn, especially Allen, Julia, Ling, Marie, Medha, Mengmeng, Nan, Svilen, and Zhuowei, for giving me comments on my research directions, being critical yet immensely helpful during practice talks, and making my overall experience at Penn so enjoyable. Thanks to the administrative and business staff who were always there to take care of administrative issues. My heartfelt thanks are to Mike Felker, who is the one-stop solution for all matters in the department.

I do not have enough words to thank Annie and Marie, who were my closest friends, confidants, and my immediate source of support, encouragement, and criticism, in the last five years. We started our journey as a PhD student together five years back, and since then I have shared every single achievement and failure with them, and have fallen back on them for any need, whether academic or personal. Our gang of girls was strengthened when Medha and Nan joined us last year, who have since wholesomely added to my life. I have been blessed with a number of very close friends. I thank Aditi, Anirban (Basu and Dey), Arindam, Arnab, Barna, Chandan, Debarghya, Dipanjan, Joydeep, Lakhsminath, Maunendra, Payel, Sagarmoy, Sayak, Shaon, Shubhankar, Srobona, Subhendu, Sumit, Supratim, Sunny, Varun, Vinaya, and all my other friends, for always being with me.

I could not have reached here without the unconditional love and support from my family. My parents Sumita Roy and Sushanta Kumar Roy, my dear sister Urmimala, uncles, aunts, and grandparents had unwavering belief in me and were overjoyed at my tiniest achievements. I thank my parents-in-law, who are always more concerned than I am about my health, happiness and professional career. My grandmother would have been the happiest person today, and her memories will drive me in life forever. Last but not the least, I do not know how to thank my loving husband Debmalya in words. He is my best friend, colleague, and collaborator, my strongest source of support and encouragement, and a pillar of strength at times of sorrow and desperation who can change it to a smile in no time. I can only thank him by making our days to come rewarding and full of happiness for both of us.

ABSTRACT

PROVENANCE AND UNCERTAINTY

Sudeepa Roy

Susan B. Davidson

Sanjeev Khanna

Data provenance, a record of the origin and transformation of data, explains how output data is derived from input data. This dissertation focuses on exploring the connection between provenance and uncertainty in two main directions: (1) how a succinct representation of provenance can help infer uncertainty in the input or the output, and (2) how introducing uncertainty can facilitate publishing provenance information while hiding associated private information.

A significant fraction of the data found in practice is imprecise, unreliable, and incomplete, and therefore uncertain. The level of uncertainty in the data must be measured and recorded in order to estimate the confidence in the results and find potential sources of error. In *probabilistic databases*, uncertainty in the input is recorded as a probability distribution, and the goal is to efficiently compute the induced probability distribution on the outputs. In general, this problem is computationally hard, and we seek to expand the class of inputs for which efficient evaluation is possible by exploiting provenance structure.

In some scenarios, the output data is directly examined for errors and is *labeled* accordingly. We need to trace back the errors in the output to the input so that the input can be refined for future processing. Because of incomplete labeling of the output and complexity of the processes generating it, the sources of error may be uncertain. We formalize the problem of source refinement, and propose models and solutions using provenance that can handle incomplete labeling. We also evaluate our solutions empirically for an application of source refinement in *information extraction*.

Data provenance is extensively used to help understand and debug scientific experiments that often involve proprietary and sensitive information. In this dissertation, we consider privacy of proprietary and commercial modules when they belong to a *work-*

flow and interact with other modules. We propose a model for *module privacy* that makes the exact functionality of the modules uncertain by selectively hiding provenance information. We also study the optimization problem of minimizing the information hidden while guaranteeing a desired level of privacy.

Contents

List of Tables

List of Illustrations

Chapter 1

Introduction

Data provenance (also known as *lineage* or *pedigree*) is a record of origin and transformation of data [33]. Data provenance, which explains how the output data is derived from the input data in an experiment, simulation, or query evaluation, has been extensively studied over the last decade. Its applications include data cleaning, sharing, and integration [46, 99, 166], data warehousing [51], transport of annotations between data sources and views [34], view update and maintenance [18, 34, 44, 81, 87, 88, 173], verifiability and debugging in scientific workflows [19, 24, 27, 128, 158] and business processes [53, 69], and so on. Various tools and frameworks have been proposed to support provenance in these applications [32, 99, 175].

The central focus of this dissertation is to explore the connection between provenance and uncertainty. A significant fraction of data found in practice is imprecise, unreliable, and incomplete, which leads to uncertainty in data. For example, RFID sensor data in wireless sensor networks is often noisy due to environmental interference, inherent measurement noise, and sensor failures. Similarly, data collected by surveys or crowd-sourced platforms like Amazon Mechanical Turk [1] is uncertain because of variations in taste, opinion, and the level of effort put in by the crowd in generating the data. In addition, some information may be lost or introduced when data undergoes multiple transformations and is stored in different formats.

In any practical application, it is important to measure and record the level of uncertainty in the data in order to estimate the confidence in the results obtained, or find potential sources of errors. Uncertainty can be recorded either in the source or in the output. In applications like sensor networks, it is well-known that RFID sensors often record inaccurate values. Therefore, multiple sensors are usually deployed for the same task, *e.g.*, recording the temperature of a room. Based on the values these sensors record, we obtain a probability distribution over a range of temperatures. This is an example of *recording uncertainty in the source*. Further processing of the recorded temperatures (*e.g.*, finding whether the room temperature was greater than a certain threshold at any time of a day) induces a probability distribution on the possible outputs.

In some scenarios, we do not have a prior measure of confidence on the source data, whereas the output data can be directly examined for errors and is *labeled* accordingly. For instance, in the above example, an independent observer may certify that the room temperature was always lower than the given threshold, whereas the sensors recorded otherwise. In this case, the sensor recordings are labeled as incorrect. It is important that this (often incompletely) labeled output data is now used to identify the faulty sensors for rectification or removal.

The above examples motivate the *first part of this dissertation*. Here we study applications of provenance, *i.e.* how the output was generated from the input, in inferring uncertainty in the input or the output data. The framework of projecting uncertainty in the input, specified as probability distributions, to the output has been widely studied in the field of *probabilistic databases* [161]. Here the main challenge is to compute the probability distribution on the possible output values as efficiently as possible for a large input data set. We use a certain form of provenance, that we call *Boolean provenance*, to efficiently compute the exact or approximate probabilities of the outputs. It has been shown in the literature that there is a class of "hard" positive relational queries (involving SELECT-PROJECT-JOIN-UNION operators) that are not amenable to efficient computation of output probabilities on some instances. Our first contribution is to efficiently identify a natural class of *query-database instance pairs* that allow efficient computation of the output probabilities even if the query is hard. Our second contribution is to initiate the study of queries that involve the *set difference* operator (EXCEPT or MINUS in SQL), which adds substantial complexity to the class of positive queries.

Conversely, for tracing uncertainty in the output back to the input, we also use Boolean provenance. The goal is to *refine the source*, *i.e.* find and remove inputs that are responsible for producing many incorrect outputs. However, the same input might produce a combination of correct and incorrect outputs. So we must ensure that refining the inputs does not delete a significant number of correct outputs. We formalize the problem of balancing incorrect outputs retained and correct outputs deleted by the process of source refinement, and propose efficient algorithmic solutions for this problem. We also address the problem of incomplete labeling of the output by proposing a statistical model that helps predict the missing labels using techniques from machine learning. Our approach of source refinement is discussed via an important application in *information extraction* where we also evaluate our solutions empirically on a real-life information extraction system [114].

In general, data provenance helps understand and debug an experiment, transaction, or query evaluation in scientific, medical, or pharmaceutical research and business processes [19, 24, 27, 53, 69, 128, 158]. *Workflows*, which capture processes and data flows in these domains, constitute a critical component of experiments like genome sequencing, protein folding, phylogenetic tree construction, clinical trials, drug design and synthesis, etc. Due to substantial investments in developing a workflow and its components, which also often includes proprietary or sensitive elements (*e.g.*, medical information, proprietary algorithms, etc), organizations may wish to keep certain details private while publishing the provenance information.

The *second part of this dissertation* focuses on *module privacy* for proprietary and commercial modules when they combine with other modules in a workflow in order to complete a task. Unlike the first part of the dissertation where we use provenance to infer uncertainty in the input or the output, we intentionally *introduce uncertainty* in provenance to protect functionality of the private modules. Arbitrary network structure and interaction with other modules in a workflow make it difficult to guarantee desired privacy levels for proprietary modules. Here we show that we can find private solutions locally for individual modules and combine them to ensure privacy of modules in a workflow irrespective of its structure. Naturally, there is a tension between publishing accurate provenance information and privacy of modules. So our algorithms aim to minimize the provenance information hidden while ensuring a required privacy level. We also argue that the guarantees provided by our algorithms are essentially the best possible.

In view of the above discussion, we now describe two main directions in this dissertation connecting provenance and uncertainty: (i) how a succinct representation of provenance can help infer uncertainty in the input or the output (Section 1.1), and (ii) how uncertainty can facilitate privacy-aware provenance (Section 1.2). In these sections, we briefly review the relevant previous approaches, discuss how we use provenance to address the underlying challenges, and summarize our technical contributions.

1.1 Provenance for Inferring Uncertainty in Input or Output

We consider the relational setting For inferring uncertainty in the input or the output, where a relational algebra query runs on an input (relational) database to compute the output view. Both projecting uncertainty in the input to the output and tracing uncertainty in the output back to the input require the knowledge of *how-provenance* of the outputs, which encodes how each output tuple is generated from the input tuples[1]. Specifically, we can encode the how-provenance of an output tuple as a polynomial in variables annotating the source tuples (see Figure 1.1). Then joint use of source tuples by the JOIN operator is captured by Boolean AND ($\cdot$), alternative ways of generating an output tuple by the PROJECT and UNION operators are captured by Boolean OR ($+$), and SELECT operator maps potential output tuples to True or False. We call these polynomials *Boolean provenance* (also known as *event expressions* or *lineage*) ImielinskiL84, GreenKT07. Figure 1.1 shows an example of Boolean provenance annotating output tuples. The provenance of tuple b_2 is $u_2 \cdot v_3 \cdot w_3$, which says that the tuple b_2 is generated in the output by joining the source tuples $R(b_1, c_3), S(c_2, a_3)$ and $T(a_3)$. On the other hand, it follows from the provenance of b_1 that it can be generated by joining three different combinations of source tuples. Next, we discuss the specific use of Boolean provenance in inferring uncertainty in the input and the output.

[1]We discuss the other forms of provenance in Chapter 2

R		
b_1	c_1	u_1
b_2	c_2	u_2
b_1	c_3	u_3

S		
c_1	a_1	v_1
c_1	a_2	v_2
c_2	a_3	v_3
c_3	a_2	v_4

T	
a_1	w_1
a_2	w_2
a_3	w_3

$q(I)$	
b_1	$u_1 \cdot v_1 \cdot w_1 + u_1 \cdot v_2 \cdot w_2 + u_3 \cdot v_4 \cdot w_2$
b_2	$u_2 \cdot v_3 \cdot w_3$

$$q(x) : -R(x,y), S(y,z), T(z)$$

Figure 1.1: A database instance I with three relations R, S, and T, query q, and Boolean provenance of the output tuples.

1.1.1 Projecting Input Uncertainty to the Output

Unreliable data sources, noisy data collectors (like sensor data), and multiple transformations and revisions inject uncertainty in data. *Probabilistic databases* enable us to capture, model, and process uncertain data assuming a probability distribution on the possible instances of the database. Query evaluation on probabilistic databases results in a probability distribution on possible outputs instead of a deterministic answer.

Boolean provenance in probabilistic databases. A standard and fundamental model is to assume *tuple-independent* probabilistic databases. Here each source tuple realizes in a random instance of the database with some probability independent of the other source tuples. The Boolean variable annotating a source tuple correspond to the *event* that the source tuple realizes, *e.g.* $R(b_1, c_1)$ exists in a random instance of the database if and only if $u_1 = \textsc{True}$, and this event has a certain probability $\Pr[u_1]$. It can be easily seen that an output tuple appears as a result of query evaluation on a random database instance if and only if its Boolean provenance evaluates to True. Therefore, in tuple-independent probabilistic databases, the probability of an output tuple equals the probability of its Boolean provenance, *e.g.* $\Pr[b_1 \text{ appears in } q(I)] = \Pr[u_1 \cdot v_1 \cdot w_1 + u_1 \cdot v_2 \cdot w_2 + u_3 \cdot v_4 \cdot w_2]$.

There are several reasons to use Boolean provenance for query evaluation in probabilistic databases. First, the number of random instances of the input database can be exponential in the number of source tuples with non-zero probability. Using Boolean provenance saves the effort of dealing with all possible database instances, *i.e.* evaluating the query on each such instance and summing up the probabilities of the instances that gives the intended output tuple[2]. Second, evaluating the probability of a Boolean expression from the probabilities of its constituent variables has been extensively studied in the literature, and is closely related to computing the number of satisfying assignments of a Boolean expression. This enables us to have a better understanding of the difficulties in probability computation. Moreover, the Boolean provenance of the output tuples can be computed without much overhead when a query is evaluated on an annotated source database. The challenge in computing the probability distributions using Boolean provenance appears in the next step. The problem of computing the probability of an arbitrary Boolean expression from that of its constituent variables is directly related to counting the number of satisfying assignments of the expression which is known to be computationally hard (*#P-hard*). Further, these *hard* expressions may be generated by very simple queries. Therefore, the key challenge in probabilistic databases is to be able to efficiently compute the probability distribution on the outputs for a large class of inputs.

Previous approaches. Dalvi et. al. have studied efficient computability of probability distribution on the outputs, and have given important *dichotomy* results for *positive relational algebra queries* [54–57]. These results show that the set of such queries can be divided into (i) the queries for which poly-time computation of output probability distribution is possible on *all instances* (called *safe queries*); and (ii) the queries for which such computation is hard (called *unsafe queries*). There has been interesting progress in other directions in probabilistic databases as well. Several systems for query processing on probabilistic databases have been developed, including MistiQ [25], Trio [17, 174], and MayBMS [10]. Significant work has been done on top-k queries [117, 144, 159] and aggregate queries [146] on probabilistic databases. The relation between Boolean provenance generated by safe queries and several knowledge compilation techniques has also been explained [101, 137]. Frameworks have been proposed for exact and approximate evaluation of the answers that work well in practice (but may not run in time polynomial in the size of input database) [77, 100, 140]. A completely different approach models correlations between tuples, compiles queries and databases into probabilistic graphical models and then performs inference on these [149, 154, 155]. Another line of work has focused

[2]For some queries, explicit computation of the Boolean provenance is not needed and the probabilities of the outputs can be directly computed while evaluating the query [54, 55].

on exploring more expressible query languages for probabilistic databases and their computability. For instance, [68] considers datalog, fixpoint and "while" queries, where either the state of the database changes after each evaluation of the query, or new results are added to the answer set; [109] studies compositional use of confidence parameters in the query such as selection based on marginal and conditional probabilities.

Our contributions. In view of the above state of the art, this dissertation considers two simple yet fundamental questions. In Chapter 3, we extend the class of inputs (query-database pairs) for which poly-time exact computation is possible, while in Chapter 4, we enrich the class of queries studied in the literature by including the *difference operations*. Here we give a technical summary of our contributions in these two chapters.

- **Query Evaluation in Probabilistic Databases using Read-Once Functions**
  Chapter 3 is based on the results published in [152] (joint work with Vittorio Perduca and Val Tannen). As mentioned earlier, safe queries are amenable to poly-time computation for all input databases. However, the class of unsafe queries includes very simple queries like $q() : -R(x), S(x,y), T(y)$ and are unavoidable in practice. Our observation is that, even for unsafe queries, there may be classes of "easy" data inputs where the answer probabilities can be computed in polynomial time in the size of the database: this we call the *instance-by-instance* approach. In particular, our goal is efficiently identify the input query-database pairs where Boolean provenance of an output is *read-once*.

  A Boolean expression is in *read-once form* if each of its variables appears exactly once, *e.g.* $x(y+z)$[3]. The significance of a read-once expression is that its probability can be computed in linear-time in the number of variables when the variables are mutually independent (repeatedly use $\Pr[xy] = \Pr[x]\Pr[y]$ and $\Pr[x+y] = 1-(1-\Pr[x])(1-\Pr[y])$). However, there are expressions that are not in read-once form, but are *read-once*, *i.e.* such an expression has an equivalent read-once form (*e.g.* $xy + xz$). Indeed, there are non-read-once expressions as well (*e.g.* $xy + yz + zx$ or $xy + yz + zw$). In Figure 1.1 the provenance of b_2 (*i.e.* $u_2v_3w_3$) is read-once and also in read-once form, while the provenance of b_1 (*i.e.* $u_1v_1w_1 + u_1v_2w_2 + u_3v_4w_2$) is not read-once. Our

[3]From now on, we will omit "·" to denote AND operation in Boolean expression for brevity.

goal in this chapter is to decide whether a Boolean provenance is read-once (and if yes, to compute the read-once form) as efficiently as possible.

When the Boolean formula is given in *"irredundant" disjunctive normal form* (DNF) (*i.e.* none of the disjuncts can be omitted using absorption like $x + xy = x$), there is a fast algorithm by Golumbic et. al. [85] to decide whether a Boolean expression is read-once. This algorithm is based upon a characterization given by Gurvich [91]. For positive relational queries, the size of the irredundant DNF for a Boolean provenance is polynomial in the size of the table, but can be exponential in the size of the query. On the other hand, often optimized query plans produce more compact expressions, motivating the study of approaches that can avoid the explicit computation of these DNFs. However, DNF computation seems unavoidable for arbitrary Boolean expressions. Hellerstein and Karpinski[97] have shown that given a monotone Boolean formula defining a read-once function, the equivalent read-once form cannot be computed in poly-time in the size of the formula unless $NP = RP$[4].

In [152], we exploit special structures of Boolean provenance to avoid expanding the expression into DNF. We show that for a large and important class of queries, *conjunctive queries without self-join*, the existing characterization and algorithms can be used to decide read-once-ness of arbitrary Boolean provenance (not necessarily in irredundant DNF) by recording some additional information when the query is evaluated. Moreover, using the properties of conjunctive queries without self-join, we propose a simpler and asymptotically faster algorithm that also introduces a novel characterization of the read-once Boolean provenance for this class of queries.

- **Queries with Difference on Probabilistic Databases**
  Chapter 4 presents our results published in [106] (joint work with Sanjeev Khanna and Val Tannen). Difference operations (e.g. EXCEPT, MINUS in SQL) are frequently found in real-life queries, but the complexity of query evaluation with these operations had not been studied previously for probabilistic databases. The concept

[4]The complexity class *RP* (*Randomized Polynomial time*) consists of the languages L having a polynomial-time randomized algorithm that rejects all inputs not in L with probability 1, and accepts all inputs in L with probability $\geq 1/2$.

of Boolean provenance naturally extends to difference (see Chapter 4). However, there are some new and considerable difficulties with such queries. For any positive query without difference operations, even if the exact computation is hard, the well-known DNF-counting algorithm given by Karp and Luby [104] can be adapted to *approximate* the probabilities upto any desired level of accuracy. But, for queries with difference, the DNF of Boolean provenance is no longer monotone, and may be exponential in the size of the database. This precludes using both the read-onceness testing algorithm of [85], and also the approximation algorithm given by Karp and Luby. In recent and independent work [77], Fink et. al. proposed a framework to compute exact and approximate probabilities for answers of arbitrary relational algebra queries that allow difference operations. However, there is no guarantee of polynomial running time.

In this work, we study the complexity of computing the probability for queries with difference on tuple-independent databases. (1) We exhibit two Boolean queries q_1 and q_2, both of which independently have nice structures, namely they are *safe* conjunctive queries without self-joins (*i.e.* allow poly-time computation on all instances), but such that computing the probability of $q_1 - q_2$ is #P-hard. This hardness of exact computation result suggests that any class of interesting queries with difference which are *safe* in the spirit of [54–56] would have to be severely restricted[5] (2) In view of this lower bound for exact computation, we give a *Fully Polynomial Randomized Approximation Scheme* (*FPRAS*) for approximating probabilities for a class of queries with difference. In particular, a corollary of our result applies to queries of the form $q_1 - q_2$, where q_1 is an arbitrary positive query while q_2 is a safe positive query without self-join. Our algorithm uses a new application of the *Karp-Luby framework* [104] that goes well beyond DNF counting, and also works for the instance-by-instance approach in the spirit of Chapter 3. (3) We show that the latter restriction stated above is important: computing the probability of "*True* $- q$" is *inapproximable* (no FPRAS exists) where *True* is the Boolean query that is always true while q is the Boolean conjunctive query $q() :- S(x), R(x,y), S(y)$. Table 1.1

[5]There is always the easy and uninteresting case when the events associated with q_1 and with q_2 are independent of each other because, for example, they operate on separate parts of the input database.

		$\Pr[q_1 - q_2]$	
$\mathbf{q}_1$	$\mathbf{q}_2$	**Exact**	**Approximate**
UCQ	safe CQ without self-join	$\#P$-hard	FPRAS
UCQ	UCQ		Inapproximable

Table 1.1: Complexity of exact and approximate computation of probability for difference of two positive queries (CQ = conjunctive query, UCQ = Union of conjunctive query).

summarizes the three results for differences of positive queries. This work is a first step to explore the complexity of queries with difference operation; an important future direction in this area is to obtain a complete classification of these queries (and Boolean provenance) in terms of their complexity.

1.1.2 Tracing Output Uncertainty back to the Input

In applications like information extraction (*e.g.*, finding all instances of person names from a document), database administration (*e.g.*, curating source data in order to satisfy certain requirements on one or more views), and business reorganization (*e.g.*, reorganizing connections between employees and customers) the outputs are often identified as correct (as expected), or incorrect (spurious). In other words, an output of these systems can be *labeled* as a *true positive* or *false positive* by an expert or user of the system. A natural goal related to these applications is to use this (potentially incomplete) labeling to improve the quality of the output of the system by refining potential erroneous inputs.

Previous approaches. In this regard, there have been two main directions of work in the literature: *deletion propagation* [35, 107, 108] and *causality* [124, 125]. The goal of deletion propagation, as studied in [35, 107, 108], is to delete a single incorrect answer tuple of a positive relational query while minimizing the *source-side-effect* (the number of source tuples needed to be deleted) or *view-side-effect* (the number of other output tuples deleted). Clearly, a more general version of the deletion propagation problem can be defined which can handle deletion of multiple tuples in the output. On the other hand, the goal of the causality framework is to compute the *responsibility* of every source tuple. The responsibility is computed in terms of the minimum number of modifications needed to the other source tuples (in addition to the current source tuple) that gives the desired set of outputs (if possible). The notion of causality can be considered as propagation of undesired tuple deletion and missing tuple insertion with a minimal source-side-effect.

In the presence of only incorrect tuples (*i.e.* no missing tuples), both deletion propagation and causality approaches attempt to delete *all* false positives in the output: while the former primarily aims to minimize the number of true positives deleted, the latter aims to minimize the modifications in source data that also preserves all true positives. However, these approach have shortcomings for many practical purposes. For instance, deleting a false positive may require deleting many other true positives. Figure 1.2 shows an example mentioned in [35, 50, 107, 108]. Here there are two source tables, *UserGroup* (allocation of users to groups) and *GroupAccess* (files that a group has access to). The output view *UserAccess* computed as

$$UserAccess(u, f) = UserGroup(u, g) \bowtie GroupAccess(g, f)$$

As depicted in the figure, the output tuple $(John, f1)$ is incorrect (*e.g.*, the user $John$ does not want to work on the file $f1$) and is desired to be removed. To delete $(John, f1)$, either $(John, sale)$ or $(sale, f1)$ has to be deleted in the source. Whichever source tuple is chosen to be deleted, two other correct output tuples are deleted as well (either $(John, f2)$ and $(John, f3)$, or $(Lisa, f1)$ and $(Joe, f1)$). Though either of these two solutions are optimum for the deletion propagation approach in [35, 50, 107, 108][6], depending on the application and importance of the correct output tuples being deleted as a side effect, simply retaining the single incorrect tuple $(John, f1)$ may be a better option. To formalize the above intuition, we need to quantify and optimize the "balance" between true positives removed and false positives retained in the answer. In Chapter 5, we will see an important application of this approach in improving the output quality of an information extraction system.

To balance the *precision* (minimize false positives) and *recall* (avoid discarding true positives) in the output, our goal in Chapter 5 is to maximize the *F-score* (the harmonic mean of precision and recall) [171]. We study this optimization problem under two natural con-

[6]The causality approach computes responsibility of every source tuple as zero since no transformation on the source tuples can keep all correct tuples while deleting the single incorrect tuple.

UserGroup	
John	*sale*
Lisa	*sale*
Joe	*sale*
mary	*eng*
Ted	*mkt*
⋮	

GroupAccess	
sale	*f*1
sale	*f*2
sale	*f*3
eng	*f*4
mkt	*f*5
⋮	

UserAccess		
John	*f*1	`incorrect`
Lisa	*f*1	`correct`
Joe	*f*1	`correct`
John	*f*2	`correct`
Lisa	*f*2	`correct`
Joe	*f*2	`correct`
John	*f*3	`correct`
Lisa	*f*3	`correct`
Joe	*f*4	`correct`
Ted	*f*4	`correct`
⋮		

Figure 1.2: Input relations *UserGroup*, *GroupAccess*, and output relation *UserAccess* with correct/incorrect labeling on the output tuples.

straints that a human supervisor is likely to use: a limit on the number of source tuples to remove (*size constraint*), or the maximum allowable decrease in recall (*recall constraint*).

Boolean provenance in source refinement. *Boolean provenance* of the output tuples in terms of the input tuples help the objective of source refinement. First, an output tuple gets deleted if and only if its Boolean provenance becomes False. This is an efficient way to check if an output tuple survives, and thus to measure the F-score of the output after refinement. Second, Boolean provenance also helps when *the labeling on the output is incomplete*. In practice, only a small amount of labeled data is available since manually labeling a large output data is an expensive and time consuming task. We assume that an output tuple is correct if and only if its Boolean provenance evaluates to True given True and False values of the source tuples in the provenance (denoting whether or not the source tuples are correct for this output tuple). Given the labeled output tuples, the probability of the correctness of the source tuples can be estimated. These probabilities are then used to compute the probability of an unlabeled output tuple being correct. This approach helps to estimate and maximize the F-score when the labeling is incomplete and avoid over-fitting the small labeled data.

Although the focus of Chapter 5 is on rule-based information extraction system, this work generalizes to any application where (i) the inputs can be modeled as source tuples in relational databases, (i) the query can be abstracted as a positive relational query, and (iii) the output tuples can be marked as correct (true positive) or incorrect (false positive). In particular, the application in information extraction (extraction of structured data from unstructured text) involves operations dealing with text data (*e.g.*, finding occurrences of person names in the text using lists or dictionaries of such names, merging two occurrences of first and last names which are close to each other to get a candidate person name, etc.). These operations can be abstracted using relational algebra operators like selection and join. We defer all the details specific to information extraction systems to Chapters 2 and 5.

Our contributions. Chapter 5 is based on [151] (joint work with Laura Chiticariu, Vitaly Feldman, Frederick R. Reiss and Huaiyu Zhu), where we systematically study the source refinement problem when the labeling is incomplete. We summarize our results abstracted in terms of positive relational algebra queries on relational databases; the actual results in Chapter 5 are presented in the context of improving the quality of an information extraction system.

- **Provenance-based Approach for Dictionary Refinement in Information Extraction** In Chapter 5, we divide the refinement problem into two sub-problems: (a) *Label estimation* copes with incomplete labeled data and estimates the "fractional" labels for the unlabeled outputs assuming a statistical model. (b) *Refinement optimization* takes the (exact or estimated) labels of the output tuples as input. It selects a set of source tuples to remove that maximizes the resulting F-score under size constraint (at most k entries are removed) or recall constraint (the recall after refinement is above a threshold).

  (1) For label estimation, we give a method based on the well-known *Expectation-Maximization* (EM) algorithm [67]. Our algorithm takes Boolean provenance of the labeled outputs as input and estimates labels of the unlabeled outputs. Under certain independence assumptions, we show that our application has a closed-form

expression for the update rules in EM which can be efficiently evaluated.

(2) For refinement optimization, we show that the problem is *NP-hard* under both size and recall constraint, even for a simple query like $q_1(x,y) :- R(x), T(x,y), S(y)$ where the relations R and S are potential sources of errors and T is trusted. Under recall constraint, this problem remains NP-hard for a simpler query like $q_2(x) : -R(x), T(x)$ where only the relation R is a potential source of errors[7]. However, for queries like q_2, which is of independent interest in information extraction, we give an optimal poly-time algorithm under size-constraint.

(3) We conduct a comprehensive set of experiments on a variety of real-world information extraction rule-sets and competition datasets that demonstrate the effectiveness of our techniques in the context of information extraction systems.

1.2 Introducing Uncertainty for Privacy-aware Provenance

This section gives an overview of our contributions in the second part of this dissertation: how introducing uncertainty in provenance can protect privacy of proprietary modules when they belong to a *workflow*. In the previous section, we discussed applications of Boolean provenance in inferring uncertainty in the input or the output. Boolean provenance is a form of *fine-grained data provenance* that captures low-level operations when a relational algebra query is evaluated on an input (relational) database. However, in the experiments in scientific research, the researchers need to handle more complex inputs than tuples (files with data in different formats) and more complicated processes than the SELECT-PROJECT-JOIN-UNION operations (*e.g.* gene sequencing algorithms). To understand and debug these experiments, the researchers are often interested in *coarse-grained data provenance* (also known as *workflow provenance*), which is a record of the involved processes and intermediate data values responsible for producing a result. Coarse-grained provenance can handle arbitrary inputs and processes, but it treats individual processes (called *modules*) as black box hiding their finer details. It also does not have a well-

[7]The provenance of the answers under q_1 is of the form $x \cdot y$, whereas for q_2, they are simply of the form x. Here x and y denote variables annotating source tuples

defined semantics like fine-grained Boolean provenance like equivalence under equivalent queries.

Workflows, which graphically capture systematic execution of a set of modules connected via data paths, constitute a key component in many modern scientific experiments and business processes. Provenance support in workflows ensures repeatability and verifiability of experimental results, facilitates debugging, and helps validate the quality of the output, which is frequently considered to be as important as the result itself in scientific experiments. There are some obvious challenges related to using workflow provenance, like efficient processing of provenance queries, managing the potentially overwhelming amount of provenance data, and extracting useful knowledge from the provenance data [8]. However, we study an almost unexplored but highly important challenge, *privacy concerns in provenance*, that restrains people from providing and using provenance information. We identified and illustrated important privacy concerns in workflow provenance in several workshop and vision papers [61, 65, 162] that include privacy of sensitive data, behavior of proprietary modules, or important decision processes involved in an execution of a workflow. Previous work [42, 83, 84] proposes solutions to enforce access control in a workflow given a set of access permissions. But, the privacy notions are somewhat informal and no guarantees on the quality of the solution are provided both in terms of privacy and utility.

Module privacy by uncertainty. In the second part of this dissertation (Chapters 6 and 7), we initiate the study of *module privacy in workflow provenance*. A module can be abstracted as a function f that maps the inputs x to the outputs $f(x)$. Our goal is to *hide* this mapping from x to $f(x)$ when f is a commercial or proprietary module (called a *private module*), which interacts with other modules in a workflow. Publishing accurate provenance information, *i.e.* the data values in an execution, reveals the exact value of $f(x)$ for one or multiple inputs x to f. Our approach is to *selectively hide provenance information*, so that the exact value of $f(x)$ is not revealed with probability more than γ, where γ is the desired level of privacy. We show that selectively hiding provenance information makes a

[8]In fact, our work in this area has considered two approaches to address this issue: compact *user-views* that reduce provenance overload but preserve relevant information [20]; and compact and efficient *reachability labeling scheme* to answer dependency queries among data items [14]. Other work includes [15, 16, 96] etc.

private module indistinguishable from many other modules (called *possible worlds*), even when the private module interacts with other modules in a workflow, and when there are *public modules* in the workflow with known functionality (like reformatting modules). Moreover, each input x is mapped to a large number of outputs by these possible worlds (there are more than $1/\gamma$ different possible values of $f(x)$).

Our contributions. Our first contribution is to formalize the notion of Γ-privacy of a private module given a "privacy requirement" Γ ($= 1/\gamma$), when it is a standalone entity (Γ-*standalone privacy*) as well as when it is a component of a workflow (Γ-*workflow privacy*). We extend the notion of ℓ-diversity [121][9] to our workflow setting to define Γ-privacy. Our other contributions in these two chapters are summarized below.

- **Provenance Views for Module Privacy**

  Chapter 6 presents the results in [63] (joint work with Susan B. Davidson, Sanjeev Khanna, Tova Milo and Debmalya Panigrahi). (1) We start with *standalone modules*, *i.e.* a simple workflow with a single module. We analyze the computational and communication complexity of obtaining a minimal cost set of input/output data items to hide such that the remaining, visible attributes guarantee Γ-standalone-privacy. In addition to being the simplest special case, the solutions for Γ-standalone-privacy serve as building blocks for Γ-workflow-privacy.

  (2) Then we consider workflows in which all modules are *private*, i.e. modules for which the user has no a priori knowledge and whose behavior must be hidden. The privacy of a module within a workflow is inherently linked to the workflow topology and the functionality of the other modules. Nevertheless, we show that guaranteeing workflow-privacy in this setting essentially reduces to implementing the standalone-privacy requirements for each module. For such *all-private workflows*, we also analyze the complexity of minimizing the cost of hidden attributes by assembling the standalone-private solutions optimally. We show that the minimization problem is NP-hard, even in a very restricted case. Therefore, we give poly-time approximation algorithms and matching hardness results for different

[9] In ℓ-diversity, the values of *non-sensitive attributes* in a relation are generalized so that, for every such generalization, there are at least ℓ different values of *sensitive attributes*.

variants of inputs (based on how standalone solutions are provided). We also show that a much better approximation can be achieved when *data sharing*, *i.e.* using the same data as input by multiple modules, is limited.

(3) However, as expected, ensuring privacy of private modules is much more difficult when public modules with known functionality are present in the same workflow. In particular, the standalone-private solutions no more compose to give a workflow-private solution. To solve this problem, we make some of the public modules private (*privatization*) and prove that this ensures workflow-privacy of the private modules. We study the revised optimization problem assuming a cost associated with privatization; the optimization problem now has much worse approximation in all the scenarios.

- **Propagation Model for Module Privacy in Public/Private Workflows**
  Chapter 7 is based on [66] (joint work with Susan B. Davidson and Tova Milo), where we take a closer look at module privacy in workflows having both public and private modules. Although the approach in Chapter 6 (privatization) is reasonable in some cases, there are many practical scenarios where it cannot be employed. For example, privatization does not work when the workflow specification (the module names and connections) is known to the users, or when the identity of the privatized public module can be discovered through the structure of the workflow and the names or types of its inputs/outputs.

  To overcome this problem we propose an alternative novel solution, based on *propagation* of data hiding through public modules. Suppose a private module m_1 is embedded in a chain workflow $m_1 \longrightarrow m_2$, where m_2 is a public equality modules (*e.g.*, a formatting module). In the propagation model, if the output from m_1 is hidden, then the output from m_2 would also be hidden, although the user would still know that m_1 is the equality function (in privatization, the "name" of m_2 is hidden so that no one knows it is an equality module).

  We show that for a special class of workflows, that includes common tree and chain workflows, propagating data hiding downstream along the paths comprising entirely of public modules (which we call *public closure*) is sufficient. This is facilitated by a different composability property between standalone-private solutions of private modules and "safe" solutions for public modules. We also argue why the assumptions of downward propagation (instead of propagating both upward and downward) and the properties of this special class of workflows are necessary. However, we also provide solutions to handle general workflows, where the data hiding may have to propagated beyond the public-closure of a private module.

Then we consider the optimization problem of minimizing the cost of the hidden data. We study the complexity of finding "safe" subsets of public modules, and show that for chain and tree workflows an optimal assembly is possible in poly-time, whereas for general DAG workflows the problem becomes NP-hard.

1.3 Roadmap

The rest of the dissertation is organized as follows. In Chapter 2, we discuss preliminaries and review related work that will be frequently used in this dissertation. The first part of the dissertation, the results on inferring input and output uncertainty using Boolean provenance, is discussed in Chapters 3 to 5. Chapter 3 discusses query evaluation in probabilistic databases using read-once functions, Chapter 4 discusses evaluation of queries with difference operations in probabilistic databases, and Chapter 5 discusses tracing errors in the output back to the input in the context of information extraction. Then we present the second part of this dissertation, introducing uncertainty to ensure module privacy, in Chapters 6 and 7. Chapter 6 introduces the notion of module privacy and studies the problem with a focus on workflows with all private modules. Chapter 7 describes the propagation model to handle workflows with both private and public modules. We conclude in Section 8 by discussing future directions.

Chapter 2

Background

This chapter presents some preliminary notions that are relevant to the rest of the dissertation. As mentioned in Chapter 1, this dissertation considers both Boolean provenance (used in Chapters 3 to 5), and workflow provenance (used in Chapters 6 and 7); here we discuss these two notions in more detail. First in Section 2.1, we review the background and related work for fine-grained *data provenance*, which generalizes Boolean provenance, in the context of both projecting input uncertainty to the output in probabilistic databases and tracing errors in the output back to the input in information extraction. Then in Section 2.2, we present our model for *coarse-grained workflow provenance*. Finally, Section 2.3 describes several complexity classes and notions of approximation and hardness used in this dissertation.

2.1 Data Provenance

The fine-grained *data provenance* describes the derivation of a particular data item in a dataset [33, 36, 43]. In recent years, several forms of provenance for database queries have been proposed to explain the origin of an output and its relation with source data (see the surveys [43, 165]). Cui et. al. formalized the notion *lineage* of output tuples [51], where each output tuple is associated with the source tuples that contributed to that output tuple. However, this form of provenance does not explain whether two source tuples have to co-exist, or, whether there are more than one possible way to produce an output tuple. Subsequently, Buneman et. al. proposed the notion of *why-provenance* that captures different *witnesses* for an output tuple [33]. They showed that a certain notion of “minimal” witnesses (a set of source tuples that is sufficient to ensure existence of an output tuple) exhibits equivalent why-provenance under equivalent queries. Buneman et. al. [33] also introduced another notion of provenance called *where-provenance*, which describes the relation between the source and output locations. Here the location refers to a *cell* in the table, *i.e.* the column name in addition to the identity of a tuple. However, why-provenance and where-provenance fail to capture some information about “how” an output tuple is derived. For instance, in the case of queries with self-joins, why-provenance does not reflect whether or not a source tuple joins with itself to produce an output tuple.

This is addressed in the notion of *how-provenance*, which has been formalized using *provenance semirings* by Green et. al. [87]. Intuitively, the provenance of an output tuple is represented as a polynomial (using semiring operations) with integer coefficients in terms of variables annotating the source tuples. These polynomials can reflect not only the result of the query, but also how the query has been evaluated on the source tuples to produce the output tuples using the specific *query plan*. Nevertheless, equivalent queries always produce equivalent provenance polynomials. Moreover, provenance semirings generalize any other semiring that can be used to annotate the output tuples in terms of the source tuples. Examples include set and bag semantics, event tables by Fuhr-Rölleke and Zimanyi [79, 178], Imielinski-Lipski algebra on c-tables [98], and also why-provenance. The semiring approach captures where-provenance as well when each location in the source relations (*i.e.* each attribute of each source tuple) is annotated with a variable [165]. In addition, provenance semirings exhibit a commutativity property for this mapping (homomorphism) to another semiring: first applying the mapping and then evaluating the query yields the same result as first evaluating the query and then applying the mapping.

In Chapters 3, 4 and 5 we will use a certain form of how-provenance (same as *event expressions* in [79, 178]), where the output tuples are annotated with Boolean polynomials in variables annotating the source tuples. This particular semiring has been called

PosBool$[X]$ in [87], that eliminates the integer coefficients of monomials and exponents of variables by the idempotence property of Boolean algebra (*e.g.*. $2x^2 \equiv x$); we call this *Boolean provenance*. Next we discuss its connection with projecting input uncertainty to the output in probabilistic databases, and tracing errors in the output back to the input in the context of information extraction in Sections 2.1.1 and 2.1.2 respectively.

2.1.1 Boolean provenance in Probabilistic Databases

Continuing the discussion of Section 1.1, here we detail relevant previous work as well as the connection between query evaluation and Boolean provenance in probabilistic databases. In both Chapters 3 and 4, we work with the widely studied and fundamental model of *tuple-independent probabilistic databases* [41, 54–57, 79, 86, 178]. Tuple-independent databases are defined by tables such as those in Figure 2.1, where each tuple t_i is associated with a probability $p_i \in [0,1]$. We annotate each tuple t_i by a distinct Boolean variable u_i which can be thought of as a notation for the event that t_i appears in a random instance of the database independent of the other tuples. Then $p_i = \Pr[u_i]$. We call the u_i's *tuple variables*. For example in Figure 2.1, the tuple variable u_1 denotes the event that the tuple $R(b_1, c_1)$ appears in a random instance defined by I and the fraction 0.7 inside parentheses is the probability of that event (*i.e.* $\Pr[u_1] = 0.7$).

R		
b_1	c_1	$u_1(0.7)$
b_2	c_2	$u_2(0.8)$
b_1	c_3	$u_3(1.0)$

S		
c_1	a_1	$v_1(0.1)$
c_1	a_2	$v_2(0.5)$
c_2	a_3	$v_3(0.2)$
c_3	a_2	$v_4(0.9)$

T	
a_1	$w_1(0.3)$
a_2	$w_2(0.4)$
a_3	$w_3(0.6)$

Figure 2.1: A tuple-independent probabilistic database I with three input relations R, S, T, where the tuples are annotated with tuple variables and probabilities.

Boolean provenance in probabilistic databases. Explicitly manipulating all possible instances is expensive (the number of such instances can be exponential in the size of the database), so techniques have been developed for obtaining the query answers from the much smaller representation tables [10, 17, 79, 116, 178]. Then query evaluation on tuple-independent probabilistic databases reduces to the computation of probability of the *Boolean provenance* of the output tuples given the probabilities of the source tuples (see Section 1.1).

For all queries q, given a probabilistic database I, the computation of the probabilities for the tuples in the answer $q(I)$ can be presented in two stages [79, 178]. In the *first stage* we compute the Boolean provenance for each tuple in the answer $q(I)$. This first stage is not a source of computational hardness, provided we are concerned, as is the case in this dissertation, only with *data complexity* [10] (i.e., we assume that the size of the query is a constant). Indeed, every Boolean provenance in $q(I)$ is of polynomial size and can be computed in polynomial time in the size of I.

$q(x) : -R(x,y), S(y,z)$

$q(I)$	
b_1	$u_1(v_1 + v_2) + u_3v_4$
b_2	u_2v_3

Figure 2.2: A *safe* query q and Boolean provenance of the output tuples in $q(I)$.

In the *second stage*, the probability of each Boolean provenance is computed from the probabilities that the model associates with the tuples in I, i.e., with the tuple variables using the standard laws. For example, in Figure 2.2 the event that the output tuple b_2 appears in $q(I)$ for a random instance described by I is described by its Boolean provenance u_2v_3. By the independence assumptions it has probability $\Pr[u_2v_3] = \Pr[u_2]\Pr[v_3] = 0.4 \times 0.2 = 0.08$.

This break into two stages using the Boolean provenance is called *intensional* semantics by Fuhr and Rölleke [79], and they observe that with this method computing the query answer probabilities requires exponentially many steps in general. Indeed, the data complexity of query evaluation on probabilistic databases is related to counting the number of its satisfying assignments which leads to #SAT, Valiant's first #P-complete problem [169], This problem has shown to be #P-hard, even for conjunctive queries [86], in fact even for quite simple Boolean queries like $q() : -R(x), S(x,y), T(z)$ [55].

[10]In this dissertation, the data input consists of the representation tables [55, 86] (possible source tuples and probabilities) rather than the collection of possible worlds.

Safe and Unsafe queries. Fuhr and Rölleke also observe that certain event independences can be taken advantage of, when present, to compute answer probabilities in PTIME, with a procedure called *extensional* semantics. The idea behind the extensional approach is the starting point for the important dichotomy results of Dalvi et. al. [54–57]. In a series of seminal papers they showed that the positive queries can be decidably and elegantly separated into those whose data complexity is #P-hard (called *unsafe queries*) and those for whom a *safe plan* taking the *extensional* approach can be found to compute the answer probabilities in poly-time (called *safe queries*). Figures 2.3 and 2.2 respectively show examples of safe and unsafe queries.

$$q'(x) :- R(x,y), S(y,z), T(z)$$

$q'(I)$	
b_1	$u_1v_1w_1 + u_1v_2w_2 + u_3v_4w_2$
b_2	$u_2v_3w_3$

Figure 2.3: An *unsafe* query q' and Boolean provenance of the output tuples in $q'(I)$.

However, even for safe queries the separate study of the Boolean provenance is beneficial. Olteanu and Huang have shown that if a conjunctive query without self-join is safe then for any database instance I the Boolean provenance of any output tuple in $q(I)$ is read-once [137]. A Boolean expression is in *read-once form* if each of its variables appears exactly once, and is *read-once* if it is equivalent to a Boolean expression in read-once form. It is easy to see that the probability of an expression whose variables denote independent events can be computed in linear time if the expression is in read-once form (basic trick: $\Pr[\varphi \vee \psi] = 1 - (1 - \Pr[\varphi])(1 - \Pr[\psi])$, because φ and ψ have disjoint sets of variables and hence are independent). With hindsight, the quest in [55] for plans for safe queries that on *all* inputs compute probability in poly-time can be seen as a quest for plans that compute Boolean provenance directly in read-once form. In Figure 2.2, the Boolean provenance of both the output tuples produced by the safe query q are already in read-once form, and are computed in this form by the safe plan $\Pi_x(R \bowtie \Pi_y S)$ found through the algorithm given in [55]. On the other hand, for the unsafe query q' in Figure 2.3, the Boolean provenance of the tuple $q'(b_1)$ is not read-once (there is no equivalent read-once form). Using *knowledge compilation* techniques such as BDDs and d-DNNFs [60], the study of

Boolean provenance also enables poly-time computation of probabilities for classes of queries larger than safe conjunctive queries without self-join [101, 137].

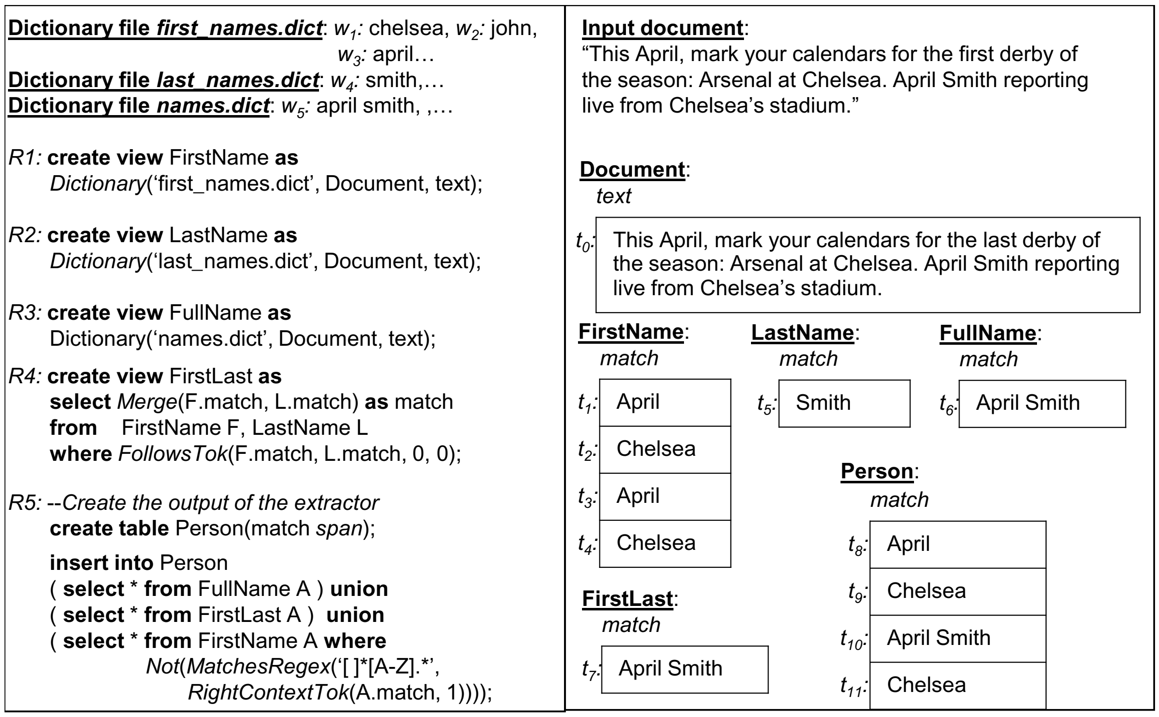
What of *unsafe positive queries*? As we saw from the example above, they can be quite common and cannot be ignored. One observation relevant for both Chapters 3 and 4 is that an unsafe query q can produce Boolean provenance that are amenable to poly-time computation on *some* instances of database. For instance, modify our example in Figure 2.1 by deleting the tuple $S(c_3, a_2)$. To obtain the modified Boolean provenance just set the corresponding tuple variable v_4 to 0 in Figure 2.3. The query q' is unsafe, but on this modified instance the Boolean provenance of the output tuple b_1 is read-once because $u_1v_1w_1 + u_1v_2w_2 = u_1(v_1w_1 + v_2w_2)$; therefore the efficient way of computing probabilities applies. This leads to an "instance-by-instance" approach (discussed in Chapter 3, and also in Chapter 4) that considers both query and the database instead of only classifying the input query as safe or unsafe.

Another observation relevant for this dissertation is that giving up on the exact computation of the probabilities solves the problem for unsafe positive queries. This is because the equivalent disjunctive normal form (DNF) of a Boolean provenance from a positive query is of polynomial size in the size of the database. Therefore, as shown in [55] (see also [144]), an FPRAS (Fully Polynomial Randomized Approximation Scheme) can be obtained by slightly adapting the well-known algorithm for DNF counting given by Karp and Luby [104]. In this dissertation, however, for positive conjunctive queries without self-join, we consider the problem of exactly evaluating the probabilities (Chapter 3). On the other hand, for queries with difference operations, we focus on approximating the probabilities (Chapter 4). Here we show that the exact computation is hard for very simple queries, and there are queries for which even any non-trivial approximation may not be possible. .

2.1.2 Boolean Provenance in Information Extraction

We discussed applications of Boolean provenance for source refinement in Section 1.1, where it is used to trace uncertainty in the output back to the input. In Section 1.1, we gave an example of view maintenance where a relational query is evaluated on an input (relational) database to produce the output view. In this section we show another interesting application of source refinement in information extraction, which can also be abstracted using relational setting.

Information extraction systems aim to extract structured data like *entities* (*e.g.* Person, Organization, Location) and relations between entities (*e.g.* Person's birth date or phone number) from unstructured text. The major components for most (rule-based) information extraction systems are a set of dictionaries of entries (like names of people, organization, location etc.) and a set of extraction rules. In Chapter 5, our goal is to improve the quality of an information extraction system by refining the dictionaries used in the system. Here we discuss the rule language, its abstraction using relational algebra operators, and Boolean provenance of the output tuples in terms of the input dictionary entries. The motivation behind dictionary refinement will be discussed in detail in Chapter 5.



Dictionary file *first_names.dict*: w_1: chelsea, w_2: john, w_3: april…
Dictionary file *last_names.dict*: w_4: smith,…
Dictionary file *names.dict*: w_5: april smith, ,…

```
R1: create view FirstName as
      Dictionary('first_names.dict', Document, text);

R2: create view LastName as
      Dictionary('last_names.dict', Document, text);

R3: create view FullName as
      Dictionary('names.dict', Document, text);

R4: create view FirstLast as
      select Merge(F.match, L.match) as match
      from    FirstName F, LastName L
      where FollowsTok(F.match, L.match, 0, 0);

R5: --Create the output of the extractor
      create table Person(match span);
      insert into Person
      ( select * from FullName A ) union
      ( select * from FirstLast A )  union
      ( select * from FirstName A where
                 Not(MatchesRegex('[ ]*[A-Z].*',
                       RightContextTok(A.match, 1))));
```

Input document:
"This April, mark your calendars for the first derby of the season: Arsenal at Chelsea. April Smith reporting live from Chelsea's stadium."

Document:

	text
t_0:	This April, mark your calendars for the last derby of the season: Arsenal at Chelsea. April Smith reporting live from Chelsea's stadium.

FirstName:

	match
t_1:	April
t_2:	Chelsea
t_3:	April
t_4:	Chelsea

LastName:

	match
t_5:	Smith

FullName:

	match
t_6:	April Smith

FirstLast:

	match
t_7:	April Smith

Person:

	match
t_8:	April
t_9:	Chelsea
t_{10}:	April Smith
t_{11}:	Chelsea

Figure 2.4: Example Person Extractor with Input Document, rules, dictionaries and intermediate and output views.

Figure 2.4 illustrates a toy example of Person extractor. The inputs to an information extraction system are a set of documents ("Input Document" in the figure), a set of dictionaries (*e.g. first_names.dict*, *last_names.dict*, etc.), and a set of rules (R_1 to R_5). The output is a set of extracted entities or relations (*Person* in the figure).

Rule Language: We use the SELECT - PROJECT - JOIN - UNION (SPJU) operations from positive relational algebra queries. enriched with a number of basic extraction primitives handling the input text. This subset expresses a core set of functionality underlying most rule languages in common use today [12, 52, 114, 148, 157], and therefore our work can be easily applied to the other rule languages. Specifically, we augment SQL with some basic information extraction primitives like *span, FollowsTok, Right(Left)ContextTok, Merge* and *MatchesRegex*. We explain these primitives using the example Person extractor given in Figure 2.4.

To model data values extracted from the input document we add a new atomic data type called *span*. A *span* is an ordered pair $\langle begin, end \rangle$ that identifies the region of an input document between the *begin* and *end* offsets. We model the input document as a table called Document with a single attribute of type *span* named *text*. For clarity, we may sometimes identify a *span* using its string value and drop the offsets when they are clear from the context. For example, to identify the region starting at offset 5 and ending at offset 10 in the document in Figure 2.4 we may use $\langle 5,10 \rangle$, or $\langle 5,10 \rangle$*: "April"*, or simply, *"April"*.

The extractor in Figure 2.4 consists of individual rules, labeled R_1 through R_5. Rules R_1 through R_4 define logical views, while rule R_5 materializes a table of extraction results. In our subsequent discussion, we refer to tuples in the output of an extractor such as those generated by R_5 as *output tuples*, or *extraction results*, or simply *occurrences*, to distinguish them from *intermediate tuples*, or *intermediate results*, generated by remaining extraction rules (e.g., R_1 to R_4).

Rules R_1, R_2 and R_3 illustrate one of our text extraction additions to SQL: the *Dictionary* table function, which identifies all occurrences of a given set of terms specified as entries in a dictionary file. The dictionary files used in R_1 to R_3 contain a list of common first names, a list of common last names, and respectively a list of common full names. The three rules define three single-column views *FirstName*, *LastName* and *FullName* containing one intermediate tuple for each dictionary match in the document. Since each dictionary entry may have multiple matches in a document, we use w_i to denote individual dictionary entries in order to distinguish them from their matches in a document. For example, the dictionary entry w_1 generates the intermediate tuples t_2 and t_4.

Rule R_4 identifies pairs of first and last names that are 0 tokens apart in the input document. The view definition uses two of the scalar functions that we add to SQL: *FollowsTok* and *Merge*. The *FollowsTok* function is used as a JOIN predicate: it takes two spans as arguments, along with a minimum and maximum character distance, and returns *true* if the spans are within the specified distance of each other in the text. The *Merge* function takes as input a pair of spans and returns the shortest span that completely covers both input spans. (*Merge* is sensitive to the order of its input spans: if the second span appears before the first, the result of is undefined.) The *select* clause of R_4 uses *Merge* to define a span that extends from the beginning of each first name to the end of the last name.

Finally, rule R_5 materializes the table Person, which constitutes the output of our extractor. It uses a *union* clause to union together candidate name spans identified by rules R_3 and R_4, as well as candidates identified by R_1 that are not immediately followed by a capitalized word. Note the *where* clause of the last union operand of R_5 which uses two other additions to SQL: the *RightContextTok* function returns the span of a given length measured in tokens to the right of the input span, while the *MatchesRegex* predicate returns true if the text of the input span matches the given regular expression. In this case, the regular expression `‘[ ]*[A-Z].*’` identifies a sequence of zero or more whitespaces, followed by an upper-case letter, followed by zero or more occurrences of any character.

Provenance of the output tuples: In Chapter 5, our goal is to improve the quality of an information extraction system by refining the noisy dictionaries used in the system. Therefore, we model the dictionaries as input relations, where each entry in each dictionary is annotated with a unique variable. The input text document is considered as a *trusted* relation comprising the words, therefore, this relation is not annotated. Now to relate to the notion of Boolean provenance, we assume a canonical algebraic representation of extraction rules as trees of operators. For the SPJU subset of the rule language, the canonical representation is essentially the same as the representation of corresponding SQL statements in terms of the relational operators. In addition, the other operations handling text are modeled as one of these relational operations. Extracting spans from the text matching a dictionary entry (the *Dictionary* operation) is treated as SELECT, whereas "merging" two spans combined by operators like *FollowsTok* is considered as JOIN. Here we give an example. $\texttt{Prov}(t_{10}) = t_6 + t_7 = t_6 + t_3 \cdot t_5$, since $\texttt{Prov}(t_7) = t_3 \cdot t_5$. We regard individual dictionary entries as the source of intermediate tuples generated by the *Dictionary* operator and express the provenance of an output tuple directly in terms of the dictionary entries that are responsible for generating that output tuple. For example, we have that $\texttt{Prov}(t_3) = w_3$, $\texttt{Prov}(t_5) = w_4$ and $\texttt{Prov}(t_6) = w_5$, and hence $\texttt{Prov}(t_{10}) = w_5 + w_3 \cdot w_4$.

2.2 Workflow Provenance

In this section, we discuss some preliminary notions of workflows and provenance in workflows that will be used in Chapters 6 and 7. The coarse-grained *workflow provenance* simply records the sequence of steps taken corresponding to the derivation of a dataset as a whole [24, 62, 158]. This is unlike the fine-grained data provenance discussed in the previous section, which considers derivation of each data item in a dataset from the source data and captures all low level operations used in the derivation. Data provenance has primarily been studied in the relational setting where the input and output data values are stored as tuples in relational databases, or can be abstracted as relational databases (see Section 2.1.1). However, in applications like scientific experiments, the scientists need to handle more complex data than tuples (*e.g.*files) and more complicated processes than relational algebra operations (*e.g.*, algorithms in biological experiments). Workflow provenance, that simply records the processes and data values involved in the generation of a result, plays an important role in understanding and debugging scientific experiments. Workflow provenance does not exhibit a well-defined semantics like data provenance (*e.g.* equivalence under equivalent queries). Nevertheless, it facilitates understanding an experiment that uses complex data and processes by treating each data value as a unit, and each process as a black-box.

In recent years, the importance of workflow provenance has been recognized by the development of several influential workflow management systems, *e.g.*, myGrid/Taverna

[136], Kepler [26], and VisTrails [78], that capture and store provenance in scientific workflows. A standard for provenance representation called the Open Provenance Model (OPM) [129] has been designed, and is used in several of the aforementioned systems. Similarly, for business processes, tools have been developed for tracking, capturing, and processing large volumes of provenance data in order to monitor and analyze executions of business operations [53, 153].

Abstracting the notion of workflows in the existing literature, in Section 2.2.1 we describe our workflow model as a graphical structure connecting nodes or modules by data flow. Then in Section 2.2 we describe how provenance information of a set of executions of the workflow can be stored using a *provenance relation*.

2.2.1 Workflows

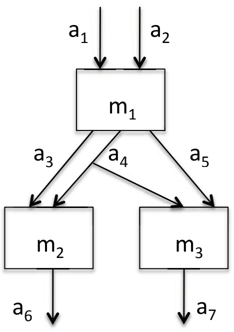
A workflow comprises a set of processes, called *modules*, connected by edges indicating potential *data flow* between modules. There are also initial input data to the workflow and final output data from the workflow. Figure 2.5c shows an example workflow with three modules m_1, m_2, m_3. Here a_1, a_2 are initial inputs, a_6, a_7 are final output, and a_3, a_4, a_5 are intermediate data that connect the modules. Note that we allow *data sharing*, *i.e.*, output of a module can be fed as input to more than one modules. In Figure 2.5c, a_4 is input to both m_2 and m_3.

Note that the modules are connected in a *DAG* (*Directed Acyclic Graph*) structure in Figure 2.5c[11]. This is the case under the assumption that the workflows are *static* and all modules in the workflow are executed exactly once in any *execution* or *run* of the workflow. In other words, the workflows do not involve any dynamic operations like loops (repeated serial executions of a module), forks (repeated parallel executions of a module), recursion, or decision processes (if-then-else). Although workflows may contain these operations in practice, standard workflow repositories [2] show that most of the real-life workflows have static structures. Moreover, static workflows form a simple and fundamental, yet non-trivial class of workflows for publishing provenance information while guaranteeing privacy.

[11]We can assume that all initial inputs originate from a dummy "source node" and all final outputs are collected by a dummy "sink node" to complete the DAG structure.

2.2.2 Provenance Relations

In every execution of the workflow, given a set of initial data values, a set of data values for intermediate and final data items are computed. We record the executions in a relation with functional dependencies, that we call the *provenance relation*.



I		O		
a_1	a_2	a_3	a_4	a_5
0	0	0	1	1
0	1	1	1	0
1	0	1	1	0
1	1	1	0	1

(b) R_1: Functionality of m_1 with functional dependency $a_1a_2 \rightarrow a_3a_4a_5$

a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	0	0	1	1	1	0
0	1	1	1	0	0	1
1	0	1	1	0	0	1
1	1	1	0	1	1	1

(c) R: Provenance relation for workflow executions with functional dependencies $a_1a_2 \rightarrow a_3a_4a_5$, $a_3a_4 \rightarrow a_6$, and $a_4a_5 \rightarrow a_7$

Figure 2.5: An example workflow, and module and workflow executions as provenance relations

Provenance Relations for Module Executions. We start with relation for a single module, that constitutes a workflow with simplest possible structure and also serves as a building block for a more complicated workflow. Let m be a module with a set I of input data items and a set O of (computed) output data items. We model executions of m as a relation R over a set of attributes $A = I \cup O$ that satisfies the functional dependency $I \rightarrow O$. In other words, I serves as a (not necessarily minimal) key for R. We assume that $I \cap O = \emptyset$, since each data item has a unique name. We will refer to I as the *input attributes* of R and to O as its *output attributes*.

We assume that the values of each attribute $a \in A$ come from a finite but arbitrarily large domain Δ_a, and let $\texttt{Dom} = \prod_{a \in I} \Delta_a$ and $\texttt{Range} = \prod_{a \in O} \Delta_a$ denote the *domain* and *range* of the module m respectively. The relation R thus represents the (possibly partial) function $m : \texttt{Dom} \rightarrow \texttt{Range}$ and tuples in R describe executions of m, namely for every $t \in R$, $\pi_O(\mathbf{t}) = m(\pi_I(\mathbf{t}))$. We overload the standard notation for projection, $\pi_A(R)$, and use it for a tuple $\mathbf{t} \in R$. Thus $\pi_A(\mathbf{t})$, for a set A of attributes, denotes the projection of $\mathbf{t}$ to the attributes in A.

Example 2.1. Figure 2.5 shows a simple workflow involving three modules m_1, m_2, m_3 with Boolean input and output attributes; we will return to it shortly and focus for now on the top module m_1. Module m_1 takes as input two data items, a_1 and a_2, and computes $a_3 = a_1 \vee a_2$, $a_4 = \neg(a_1 \wedge a_2)$ and $a_5 = \neg(a_1 \oplus a_2)$. (The symbol $\oplus$ denotes XOR). The relational representation (functionality) of module m_1 is shown in Figure 2.5b as relation R_1, with the functional dependency $a_1 a_2 \longrightarrow a_3 a_4 a_5$. For clarity, we have added I (input) and O (output) above the attribute names to indicate their role. $\square$

Provenance Relations for Workflow Executions. A workflow W consists of a set of modules $m_1, \cdots, m_n$, connected as a DAG (see, for instance, the workflow in Figure 2.5). Each module m_i has a set I_i of input attributes and a set O_i of output attributes. We assume that (1) for each module, its input and output attributes are disjoint, i.e. $I_i \cap O_i = \emptyset$, (2) the output attributes of distinct modules are disjoint, namely $O_i \cap O_j = \emptyset$, for $i \neq j$ (since each data item is produced by a unique module), and (3) whenever an output of a module m_i is fed as input to a module m_j the corresponding output and input attributes of m_i and m_j are the same. The DAG shape of the workflow guarantees that these requirements are not contradictory. Note that, it is possible that $I_i \cap I_j \neq \emptyset$ for $i \neq j$ since we allow data sharing.

We model executions of W as a *provenance relation* R over the set of attributes $A = \cup_{i=1}^{n}(I_i \cup O_i)$, satisfying the set of functional dependencies $F = \{I_i \rightarrow O_i : i \in [1,n]\}$. Equivalently, the provenance relation R for all possible executions of W can be formed by *joining* the individual relations for the modules $m_1, \cdots, m_n$. Each tuple in R describes an execution of the workflow W. In particular, for every $t \in R$, and every $i \in [1,n]$, $\pi_{O_i}(\mathbf{t}) = m_i(\pi_{I_i}(\mathbf{t}))$.

Example 2.2. Returning to the sample workflow in Figure 2.5, let module m_2 compute $a_6 = a_3 \oplus a_4$ and the module m_3 compute $a_7 = \neg(a_4 + a_5)$. The relation R in Figure 2.5c shows all possible executions of the workflow. The input and output attributes of modules m_1, m_2, m_3 respectively are (i) $I_1 = \{a_1, a_2\}$, $O_1 = \{a_3, a_4, a_5\}$, (ii) $I_2 = \{a_3, a_4\}$, $O_2 = \{a_6\}$ and (iii) $I_3 = \{a_4, a_5\}$, $O_3 = \{a_7\}$. The underlying functional dependencies in R reflect the keys of the constituent modules, e.g. from m_1 we have $a_1a_2 \longrightarrow a_3a_4a_5$, from m_2 we have $a_3a_4 \longrightarrow a_6$, and from m_3 we have $a_4a_5 \longrightarrow a_7$. $\square$

Note that since the modules in the workflow are static, the provenance relation R contains the entire provenance information for any run of the workflow (all modules and values of the other data items that have been used in the generation of an output data item can be known). The preliminary notions related to module privacy will be discussed in Chapters 6 and 7.

2.3 Complexity Classes and Approximation

In addition to the more standard notion of optimality, NP-hardness and polynomial running time, we will use some other complexity classes, approximation algorithms, and hardness of approximation in this dissertation.

In Chapter 3 and 4, we will frequently use the terms *#P-hardness*, *FPRAS*, and *inapproximability*. The complexity class ***#P*** captures the class of function problems of the form "compute $f(x)$", where f is the number of accepting paths of a non-deterministic turing machine. A problem is *#P-hard* if every problem in #P can be reduced to it by a polynomial-time counting reduction. The canonical first #P-hard (also #P-complete) problem defined by Valiant is #SAT [169], which counts the number of satisfying assignments of a Boolean formula.

An ***FPRAS*** (*fully polynomial randomized approximation scheme*) is a randomized algorithm that runs in time polynomial in the input size and $1/\epsilon$ (ϵ is the *accuracy parameter*) and produces a result that is correct to within relative error $(1 \pm \epsilon)$ with probability $\geq 3/4$. The specific value of $3/4$ is not important; the success probability can be boosted to $1 - \delta$ for any given *confidence parameter* δ by repeating the experiment $\log(1/\delta)$ times and then taking the median of the values. On the other hand, a counting problem is ***inapproximable***, if it has no FPRAS under standard complexity assumptions.

We will use the following standard notions of approximation: an algorithm is said to be a $\mu(n)$***-approximation algorithm*** for a given optimization problem, for some non-decreasing function $\mu(n) : \mathbb{N}^+ \to \mathbb{R}^+$, if for every input of size n it outputs a solution of value at most a multiplicative factor of $\mu(n)$ away from the optimum.

An optimization problem is said to be $\mu(n)$***-hard to approximate*** if a poly-time $\mu(n)$-approximation algorithm for the problem does not exist under standard complexity assumptions. In Chapter 6, we will use standard complexity assumptions of the form $NP \nsubseteq DTIME(n^{f(n)})$, where $f(n)$ is a poly-logarithmic or sub-logarithmic function of n and $DTIME$ represents deterministic time. If a problem is $\Omega(\mu(n))$-hard to approximate unless $NP \nsubseteq DTIME(n^{f(n)})$, then there cannot be a $o(\mu(n))$-approximation algorithm unless all problems in NP have $O(n^{f(n)})$-time deterministic exact algorithms.

In Chapter 6, we also use APX-hardness. An optimization problem is said to be ***APX-hard*** if there exists a constant $\epsilon > 0$ such that a $(1+\epsilon)$-approximation in polynomial time would imply $P = NP$. If a problem is APX-hard, then the problem cannot have a $PTAS$, i.e, a $(1+\epsilon)$-approximation algorithm which runs in poly-time for all constant $\epsilon > 0$, unless $P = NP$. This chapter also discusses ***communication complexity*** of some problems, which tries to quantify the amount of communication required to compute a given function, ignoring the complexity of computation discussed so far.

Chapter 3

Query Evaluation in Probabilistic Databases using Read-Once Functions

In this chapter, we present our first work to efficiently compute the uncertainty in the output from uncertain inputs. Here we evaluate probabilities of the output tuples in probabilistic databases using *read-once* Boolean provenance (see Sections 1.1.1 and 2.1.1 for a discussion on the query evaluation in probabilistic databases).

3.1 Overview

For tuple-independent probabilistic databases, Dalvi and Suciu showed an elegant dichotomy result to divide the class of conjunctive queries into those whose data complexity is #P-hard and those for whom a *safe plan* can be found to compute the answer probabilities in poly-time [54–56]. Our starting point in this work is the observation that even when the data complexity of a query is #P-hard (i.e. the query is *unsafe* [55]), there may be classes of data inputs for which the computation can be done in poly-time. We illustrate with a simple example.

Example 3.2. Consider the tuple-independent probabilistic database in Figure 3.1 and the conjunctive query $q() : -R(x), S(x,y), T(y)$. The Boolean provenance annotating the an-

$R=$

a_1	$w_1(0.3)$
b_1	$w_2(0.4)$
a_2	$w_3(0.6)$

$S=$

a_1	c_1	$v_1(0.1)$
b_1	c_1	$v_2(0.5)$
a_2	c_2	$v_3(0.2)$
a_2	d_2	$v_4(0.1)$

$T=$

c_1	$u_1(0.7)$
c_2	$u_2(0.8)$
d_2	$u_3(0.4)$

Figure 3.1: A tuple-independent probabilistic database where the tuples are annotated with tuple variables and probabilities.

swer True is[12]

$$f = w_1v_1u_1 + w_2v_2u_1 + w_3v_3u_2 + w_3v_4u_3 \tag{3.1}$$

This can be obtained with the standard plan $\pi_{()}((R \bowtie S) \bowtie T)$. However, it is equivalent to

$$(w_1v_1 + w_2v_2)u_1 + w_3(v_3u_2 + v_4u_3) \tag{3.2}$$

which has the property that each variable occurs exactly once.

The independence of source tuples allows us to use $\Pr(xy) = \Pr(x)\Pr(y)$ and $\Pr(x + y) = 1 - (1 - \Pr(x))(1 - \Pr(y))$. Therefore, the probability of the answer (3.2) can be computed as: $P(f) = P((w_1v_1 + w_2v_2)u_1 + w_3(v_3u_2 + v_4u_3))$= $1 - [1 - P(w_1v_1 + w_2v_2)P(u_1)]$ $[1 - P(w_3)P(v_3u_2 + v_4u_3)]$, where $P(w_1v_1 + w_2v_2) = 1 - [1 - P(w_1)P(v_1)][1 - P(w_2)P(v_2)]$ and $P(v_3u_2 + v_4u_3) = 1 - [1 - P(v_3)P(u_2)][1 - P(v_4)P(u_3)]$. $\square$

It can be verified that there is no relational algebra plan that directly yields (3.2) above. We can extend this example to an entire class of representation tables of unbounded size. Boolean expressions where each variable appears exactly once have *read-once forms*, and the expressions that are equivalent to read-once forms are called *read-once*[13]. Of course, not all Boolean expressions are read-once, eg., $xy + yz + zx$ or $xy + yz + zu$ are not.

In this work, we consider only *Boolean conjunctive queries*. We can do this without loss of generality because we can associate to a non-Boolean conjunctive query q and an

[12]To reduce the size of expressions and following established tradition we use $+$ for $\vee$ and $\cdot$ for $\wedge$, and we even omit the latter in most terms.

[13]Read-once expressions have been called by various names, eg., separable, fanout-free [95], repetition-free [90], μ-expressions [170], non-repeating [141], but since the late 80's [97] the terminology seems to have converged on *read-once*

instance I a set of Boolean queries in the usual manner: for each tuple t in the answer relation $q(I)$, consider the Boolean conjunctive query Q_t which is obtained from q by replacing the head variables with the corresponding values in t. We will also use $q(I)$ to denote the Boolean expression generated by evaluating the query q on instance I, which may have different (but equivalent) forms based on the query plan.

Moreover, we consider only queries *without self-join*. Therefore our queries have the form $q() : -R_1(\mathbf{x}_1), \ldots, R_k(\mathbf{x}_k)$ where $R_1, \ldots, R_k$ are all *distinct* table names while the $\mathbf{x}_i$'s are tuples of first order (FO) variables [14] or constants, possibly with repetitions, matching the arities of the tables. If the database has tables that do not appear in the query, they are of no interest, so we will always assume that our queries feature all the table names in the database schema $\mathbf{R}$.

Therefore the goal of the work is as follows: *Given a tuple-independent database I and a Boolean conjunctive query without self join q, when is $q(I)$ read-once and if so, can its read-once form be computed efficiently*?

For this class of queries, we only need to work with *monotone* Boolean formulas (all literals are positive, only disjunction and conjunction operations). Every such formula is equivalent to (many) *disjunctive normal forms* (DNFs) which are disjunctions of conjunctions of variables. These conjunctions are called *implicants* for the DNF. By idempotence we can take the variables in an implicant to be distinct and the implicants of a DNF to be distinct from each other. A *prime implicant* of a formula f is one with a minimal set of variables among all that can appear in DNFs equivalent to f. By absorption, we can retain only the prime implicants in a DNF. The result is called *the irredundant* DNF (IDNF) of f, as it is uniquely determined by f (modulo associativity and commutativity). We usually denote it by f_{IDNF}. Note that in particular the set of prime implicants is uniquely determined by f.

Golumbic et. al.[85] gave a fast algorithm to check read-onceness of a Boolean expression and compute the equivalent read-once form (if possible). However, the Boolean expression has to be specified in IDNF. The algorithm is based upon a characterization in

[14]"First-order" is to emphasize the distinction between the variables in the query subgoals and the variables in the Boolean expressions.

terms of the formula's *co-occurrence graph* by Gurvich [91] which we explain in Section 3.2. However, for positive relational queries (even for conjunctive queries without self-join), the size of the IDNF of Boolean provenance is polynomial in the size of the table, but often (and necessarily) exponential in the size of the query. This is a good reason for avoiding the explicit computation of the IDNFs, and in particular for not relying on the algorithm in [85]. In a recent and independent work, Sen et al. [156] proved that for the Boolean expressions that arise out of the evaluation of conjunctive queries without self-joins the characterization in [91] can be simplified. However, it is also stated [156] that even for conjunctive queries without self-joins computing co-occurrence graphs likely requires obtaining the IDNF of the Boolean expressions.

In this context, we summarize our contributions in this chapter below.

1. We show that an excursion through the IDNF is in fact not necessary because the co-occurrence graphs can be computed directly from the *provenance graph* [88, 105] that captures the computation of the query on a table. Provenance graphs are DAG representations of the Boolean provenance in such a way that most common subexpressions for the entire table (rather than just each tuple) are not replicated. The smaller size of the provenance graphs likely provides practical speedups in the computations (compared for example with the provenance trees of [156]). Moreover, our approach may be applicable to other kinds of queries, as long as their provenance graphs satisfy a simple criterion that we identify. To give more context to our results, we also note that Hellerstein and Karpinski[97] have shown that if $RP \neq NP$ then deciding whether an arbitrary monotone Boolean formula is read-once cannot be done in PTIME in the size of the formula.

2. The restriction to conjunctive queries without self-joins further allows us to contribute improvements even over an approach that composes our efficient computation of co-occurrence graphs with one of the linear-time algorithms to check the desired property of these graphs [29, 49, 92]. Indeed, we show that only a certain subgraph of the co-occurrence graph (we call it the *co-table* graph) is relevant for our stated problem. The co-table graph can be asymptotically smaller than the co-occurrence graph for some classes of queries and instances. To enable the use of

only part of the co-occurrence graph we contribute a novel algorithm that computes (when they exist) the read-once forms, using two new ideas: *row decomposition* and *table decomposition*. Using just connectivity tests (eg., DFS), our algorithm is simpler to implement than the cograph recognition algorithms in [29, 49, 92] and it has the potential of affecting the implementation of probabilistic databases.

The proof of completeness for our algorithm does not use the cograph characterization on which [156] relies. As such, the algorithm itself provides an alternative new characterization of read-once expressions generated by conjunctive queries without self-joins. This may provide useful insights into extending the approach to handle larger classes of queries. We compare the time complexity of our algorithm with the previous approaches in detail.

It is also important to note that neither the results of this work, nor those of [156] provide complexity dichotomies as does, eg. [55]. It is easy to give a family of probabilistic databases for which the query in Figure 3.1 (c) generates event expressions of the form: $x_1x_2 + x_2x_3 + \cdots + x_{n-1}x_n + x_nx_{n+1}$. These formulas are not read-once, but with a simple memoization (dynamic programming) technique we can compute their probability in time linear in n (see Section A.1.1 in the appendix). In general, Jha and Suciu have studied classes of queries that allow poly-time computation using other knowledge compilation techniques beyond read-onceness of Boolean provenance[101].

Organization. In Section 3.2 we explain how to compute provenance DAGs for SELECT-PROJECT-JOIN (SPJ) queries and compare the sizes of the co-occurrence and co-table graphs. Section 3.3 presents a characterization of the co-occurrence graphs for conjunctive queries without self-joins. The characterization uses the provenance DAG. With this characterization we give an efficient algorithm for computing the co-table (and co-occurrence) graph. In Section 3.4 we give an efficient algorithm that, using the co-table graph, checks if the result of the query is read-once, and if so computes its read-once form. Putting together these two algorithms we obtain an efficient query-answering algorithm that is *complete* for Boolean conjunctive queries without self-joins and tuple-independent databases that yield read-once event expressions. In Section 3.5 we compare the time complexity of this algorithm with that of Sen et al. [156], and other approaches that take advantage of past work in the read-once and cograph literature. Related work, conclusions and ideas for further work ensue.

3.2 Preliminaries

In this section we explain some notions that will be used later in this chapter.

3.2.1 Provenance Graphs

We start with *provenance graphs* (PG). The concept that we define here is a small variation on the provenance graphs defined in [88, 105] where conjunctive queries (part of mapping specifications) are treated as a black box. It is important for the provenance graphs used in this work to reflect the structure of different SPJ query plans that compute the same conjunctive query.

A provenance graph is a directed acyclic graph (DAG) H such that the nodes $V(H)$ of H are labeled by variables or by the operation symbols $\cdot$ and $+$. As we show below, each node corresponds to a tuple in an event table that represents the set of possible worlds of either the input database or some intermediate database computed by the query plan. An edge $u \to v$ is in $E(H)$ if the tuple corresponding to u is computed using the tuple corresponding to v in either a join (in which case u is labeled with $\cdot$) or a projection (in which case u is labeled with $+$). The nodes with no outgoing edges are those labeled with variables and are called leaves while the nodes with no incoming edges are called roots (and can be labeled with either operation symbol). Provenance graphs (PGs) are closely related to the *lineage trees* of [156]. In fact, the lineage trees are tree representations of the Boolean event expressions, while PGs are more economical: they represent the same expressions but without the multiplicity of common subexpressions. Thus, they are associated with an entire table rather than with each tuple separately, each root of the graph corresponding to a tuple in the table. [15]

We explain how the SPJ algebra works on tables with PGs. If tables R_1 and R_2 have PGs H_1 and H_2 then the PG for $R_1 \bowtie R_2$ is constructed as follows. Take the disjoint union

[15]Note that to facilitate the comparison with the lineage trees the edge direction here is the opposite of the direction in [88, 105].

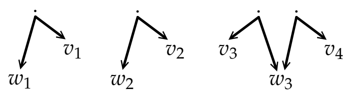


Figure 3.2: Provenance graph for $R \bowtie S$.

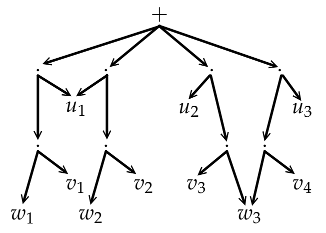


Figure 3.3: Provenance graph for $\pi_{()}((R \bowtie S) \bowtie T)$.

H of H_1 and H_2. For every $t_1 \in R_1$ and $t_2 \in R_2$ that do join, add a new root labeled with $\cdot$ and make the root of H_1 corresponding to t_1 and that of H_2 corresponding to t_2 children of this new root. Afterwards, delete (recursively) any remaining roots from H_1 and H_2. For example, referring again to Figure 3.1, the PG associated with the table computed by $R \bowtie S$ is shown in Figure 3.2.

For selection, delete (recursively) the roots that correspond to the tuples that do not satisfy the selection predicate. For projection, consider a table T with PG H and X a subset of its attributes. The PG for $\pi_X R$ is constructed as follows. For each $t \in \pi_X R$, let $t_1, \ldots, t_m$ be all the tuples in R that X-project to t. Add to H a new root labeled with $+$ and make the roots in H corresponding to $t_1, \ldots, t_m$ the children of this new root. Referring again to Figure 3.1, the PG associated with the result of the query plan $\pi_{()}((R \bowtie S) \bowtie T)$ is shown in Figure 3.3. Since the query is Boolean, this PG has just one root.

The Boolean provenance that annotate tuples in the event tables built in the intensional approach can be read off the provenance graphs. Indeed, if t occurs in an (initial, intermediate, or final) table T whose PG is H, then, starting at the root u of H corresponding to t, traverse the subgraph induced by all the nodes reachable from u and build the Boolean expression recursively using parent labels as operation symbols and the subexpressions corresponding to the children as operands. For example, we read $(((w_1 \cdot v_1) \cdot u_1) + (u_1 \cdot (w_2 \cdot v_2)) + (u_2 \cdot (v_3 \cdot w_3)) + ((w_3 \cdot v_4) \cdot u_3))$ off the PG in Figure 3.3.

3.2.2 Co-occurrence Graphs vs. Co-table Graphs

Gurvich [91] gave a characterization of the read-once Boolean expressions in terms of their co-occurrence graphs which is defined as follows:

Definition 3.3. *The* co-occurrence graph, *notation G_{co}, of a Boolean formula f is an undirected graph whose set of vertices $V(G_{co})$ is the set $\mathtt{Var}(f)$ of variables of f and whose set $E(G_{co})$ of edges is defined as follows: there is an edge between x and y iff they both occur in the same prime implicant of f.*

Therefore, G_{co} is uniquely determined by f and it can be constructed from f_{IDNF}. This construction is quadratic in the size of f_{IDNF} but of course f_{IDNF} can be exponentially larger than f. Figure 3.4 shows the co-occurrence graph for the Boolean expression f in equation (3.1) of Example 3.2. As this figure shows, the co-occurrence graphs for expressions generated by conjunctive queries without self join are always *k-partite*[16] graphs on tuple variables from k different tables.

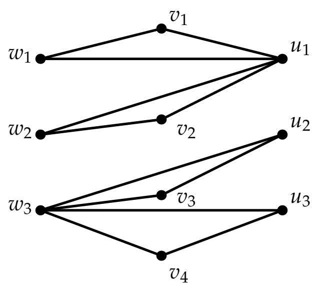


Figure 3.4: Co-occurrence graph G_{co} for the Boolean provenance f in Example 3.2.

We are interested in the co-occurrence graph G_{co} of a Boolean formula f because it plays a crucial role in f being read-once. [91] has shown that a monotone f is read-once iff (1) it is "normal" and (2) its G_{co} is a "cograph". We don't need to discuss normality because [156] has shown that for Boolean provenance that arise from conjunctive queries without self-joins it follows from the cograph property. We will also avoid defining what a cograph is (see [47, 49]) except to note that cograph recognition can be done in linear time [29, 49, 92] and that when applied to the co-occurrence graph of f the recognition

[16]A graph $(V_1 \cup \cdots \cup V_k, E)$ is *k-partite*, if for any edge $(u,v) \in E$ where $u \in V_i$ and $v \in V_j$, $i \neq j$.

algorithms also produce, in effect, the read-once form of f, when it exists.

Although the co-occurrence graph of f is defined in terms of f_{IDNF}, we show in Section 3.3 that when f is the Boolean provenance produced by a Boolean conjunctive query without self-joins then we can efficiently compute the G_{co} of f from the provenance graph H of any plan for the query. Combining this with any of the cograph recognition algorithms we just cited, this yields one algorithm for the goal of this work, which we will call a **cograph-help algorithm**.

Because it uses the more general-purpose step of cograph recognition a cograph-help algorithm will not fully take advantage of the restriction to conjunctive queries without self-joins. Intuitively, with this restriction there may be lots of edges in G_{co} that are irrelevant because they link tuples that are not joined by the query. This leads us to the notion of co-table graph defined below.

Toward the definition of the co-table graph we also need that of table-adjacency graph, denoted by G_T, as follows:

Definition 3.4. *Given a Boolean query without self-joins $q() : -R_1(\mathbf{x}_1), \ldots, R_k(\mathbf{x}_k)$, the vertex set $V(G_T)$ of its* table-adjacency graph *G_T is the set of k table names $R_1, \cdots, R_k$. We will say that R_i and R_j are* adjacent *iff $\mathbf{x}_i$ and $\mathbf{x}_j$ have at least one FO variable in common i.e., R_i and R_j are joined by the query. The set of edges $E(G_T)$ consists of the pairs of adjacent table names.*

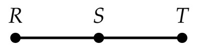


Figure 3.5: Table-adjacency graph G_T for the relations in Example 3.2.

The table-adjacency graph G_T for the query in Example 3.2 is depicted in Figure 3.5. This graph helps us remove edges irrelevant to a query from the graph G_{co}. For example, if there is an edge between $x \in R_i$ and $x' \in R_j$ in G_{co}, but there is no edge between R_i and R_j in G_T, then (i) either there is no path connecting R_i to R_j in G_T (so all tuples in R_i pair with all tuples in R_j), or, (ii) x and x' are connected in G_{co} via a set of tuples $x_1, \cdots, x_\ell$, such that the tables containing these tuples are connected by a path in G_T. Our algorithm in Section 3.4 shows that all such edges (x, x') can be safely deleted from G_{co} for the evaluation of the query that yielded G_T. Next we define a co-table graph.

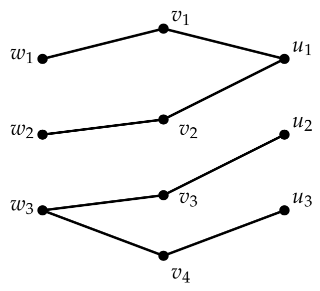


Figure 3.6: Co-table graph G_C for the Boolean provenance f in Example 3.2.

Definition 3.5. *The* ***co-table graph*** *G_C is the subgraph of G_{co} with $V(G_C) = V(G_{co})$ and such that given two tuples $x \in R_i$ and $x' \in R_j$ there is an edge $(x, x') \in E(G_C)$ iff $(x, x') \in E(G_{co})$ and R_i and R_j are adjacent in G_T.*

Figure 3.6 shows the co-table graph G_C generated by the example in Figure 3.1.

Co-occurrence graph vs. co-table graph. The advantage of using the co-table graph instead of the co-occurrence graph is most dramatic in the following example:

Example 3.6. Consider $q() : -R_1(\mathbf{x}_1), R_2(\mathbf{x}_2)$ where $\mathbf{x}_1$ and $\mathbf{x}_2$ have no common FO variable. Assuming that each of the tables R_1 and R_2 has n tuples, G_{co} has n^2 edges while G_C has none. A cograph-help algorithm must spend $\Omega(n^2)$ time even if it only reads G_{co}. $\square$

On the other hand, G_C can be as big as G_{co}. In fact, when G_T is a complete graph (see next example), $G_C = G_{co}$.

Example 3.7. Consider $q() : -R_1(\mathbf{x}_1, y), \ldots, R_k(\mathbf{x}_k, y)$ where $\mathbf{x}_i$ and $\mathbf{x}_j$ have no common FO variable if $i \neq j$. Here G_T is the complete graph on $R_1, \ldots, R_k$ and $G_C = G_{co}$. $\square$

However, it can be verified that both our algorithm and the cograph-help algorithm have the same time complexity on the above example.

3.3 Computing the Co-Table Graph

In this section we show that given as input the provenance DAG H of a Boolean conjunctive query *plan* without self-joins Q on a table-independent database representation I, the co-table graph G_C and the co-occurrence graph G_{co} of the Boolean formula $Q(I)$ (see definitions in section 3.2) can be computed in poly-time in the sizes of H, I and Q.

It turns out that G_C and G_{co} are computed by similar algorithms, one being a minor modification of the other. As discussed in section 3.1, the co-occurrence graph G_{co} can then be used in conjunction with cograph recognition algorithms (eg., [29, 49, 92]), to find the read-once form of $Q(I)$ if it exists. On the other hand, the smaller co-table graph G_C is used by our algorithm described in section 3.4 for the same purpose.

We use $\texttt{Var}(f)$ to denote the sets of variables in a monotone Boolean expression f. Recall that the provenance DAG H is a layered graph where every layer corresponds to a select, project or join operation in the query plan. We define the *width* of H as the maximum number of nodes at any layer of the DAG H and denote it by β_H. The main result in this section is summarized by the following theorem.

Theorem 3.8. *Let $f = Q(I)$ be the Boolean expression computed by the query plan Q on the table representation I (f can also be read off the provenance graph of Q on I, H), $n = |\texttt{Var}(f)|$ be the number of variables in f, $m_H = |E(H)|$ be the number of edges of H, β_H be the width of H, and $m_{co} = |E(G_{co})|$ be the number of edges of G_{co}, the co-occurrence graph of f.*

1. *G_{co} can be computed in time $O(nm_H + \beta_H m_{co})$.*
2. *Further, the co-table graph G_C of f can be computed in time $O(nm_H + \beta_H m_{co} + k^2\alpha \log \alpha)$ where k is the number of tables in Q, and α is the maximum arity (width) of the tables in Q.*

3.3.1 LCA-Based Characterization of the Co- Occurrence Graph

Here we give a characterization of the presence of an edge (x, y) in G_{co} based on the *least common ancestors* of x and y in the graph H.

Again, let $f = Q(I)$ be the Boolean expression computed by the query plan Q on the table representation I. As explained in section 3.2 f can also be read off the provenance graph H of Q and I since H is the representation of f without duplication of common subexpressions.

The absence of self-joins in Q implies the following.

Lemma 3.9. *The DNF generated by expanding f (or H) using only the distributivity rule is in fact the IDNF of f up to idempotency (i.e. repetition of the same prime implicant is allowed).*

Proof. Let g be the DNF generated from f by applying distributivity repeatedly. Due to the absence of self-joins g every implicant in g will have exactly one tuple from every table. Therefore, for any two implicants in g the set of variables in one is not a strict subset of the set of variables in the other and further absorption (eg., $xy + xyz = xy$) does not apply. (At worst, two implicants can be the same and the idempotence rule reduces one.) Therefore, g is also irredundant and hence *the* IDNF of f (up to commutativity and associativity). $\square$

Denote by f_{IDNF} the IDNF of f, which, as we have seen, can be computed from f just by applying distributivity.

As with any DAG, we can talk about the nodes of H in terms of successors, predecessors, ancestors, and descendants, and finally about the *least common ancestors* of two nodes, denoted $\texttt{lca}(x,y)$. Because H has a root $\texttt{lca}(x,y)$ is never empty. When H is a tree, $\texttt{lca}(x,y)$ consists of a single node. For a node $u \in V(H)$, we denote the set of leaf variables which are descendants of u by $\texttt{Var}(u)$ (overloaded notation warning!); in other words, a variable x belongs to $\texttt{Var}(u)$, $u \in V(H)$, if and only if x is reachable from u in H. Now we prove the key lemma of this section:

Lemma 3.10. *Two variables $x,y \in \texttt{Var}(f)$ belong together to a (prime) implicant of f_{IDNF} if and only if the set $\texttt{lca}(x,y)$ contains a $\cdot$-node.*

Proof. (if) Suppose $\texttt{lca}(x,y)$ contains a $\cdot$-node u, i.e., x,y are both descendants of two distinct successors v_1,v_2 of u. Since the $\cdot$ operation multiplies all variables in $\texttt{Var}(v_1)$ with all variables in $\texttt{Var}(v_2)$, x and y will appear together in some implicant in f_{IDNF} which will not be absorbed by other implicants by Lemma 3.9.

(only if) Suppose that x,y appear together in an implicant of f_{IDNF} and $\texttt{lca}(x,y)$ contains no $\cdot$-node. Then no $\cdot$-node in $V(H)$ has x,y in $\texttt{Var}(v_1),\texttt{Var}(v_2)$, where v_1,v_2 are its two distinct successors (note that any $\cdot$-node in a provenance DAG H can have exactly two successors). This implies that x and y can never be multiplied, contradiction. $\square$

Since there are exactly k tables in the query plan, every implicant in f_{IDNF} will be of size k. Therefore:

Lemma 3.11. *For every variable $x \in \texttt{Var}(f)$ and $\cdot$-node $u \in V(H)$, if $x \in \texttt{Var}(u)$, then $x \in \texttt{Var}(v)$ for exactly one successor v of u.*

Proof. If $x \in \texttt{Var}(f)$ belongs to $\texttt{Var}(v_1)$,$\texttt{Var}(v_2)$ for two distinct successors v_1,v_2 of u, then some implicant in f_{IDNF} will have $< k$ variables since $x \cdot x = x$ by idempotence. □

The statement of Lemma 3.10 provides a criterion for computing G_{co} using the computation of least common ancestors in the provenance graph, which is in often more efficient than computing the entire IDNF. We have shown that this criterion is satisfied in the case of conjunctive queries without self-joins. But it may also be satisfied by other kinds of queries, which opens a path to identifying other cases in which such an approach would work.

3.3.2 Computing the Table-Adjacency Graph

It is easier to describe the computation of G_T if we use the query in rule form $Q() :- R_1(\mathbf{x}_1), \ldots, R_k(\mathbf{x}_k)$.

The rule form can be computed in linear time from the SPJ query plan. Now the vertex set $V(G_T)$ is the set of table names $R_1, \cdots, R_k$. and an edge exists between R_i,R_j iff $\mathbf{x}_i$ and $\mathbf{x}_j$ have at least one FO variable in common i.e., R_i and R_j are joined. Whether or not such an edge should be added can be decided in time $O(\alpha \log \alpha)$ by sorting and intersecting $\mathbf{x}_i$ and $\mathbf{x}_j$. Here α is the maximum arity (width) of the tables $R_1, \cdots, R_k$. Hence G_T can be computed in time $O(k^2 \alpha \log \alpha)$.

3.3.3 Computing the Co-Table Graph

Recall that co-table graph G_C is a subgraph of the co-occurrence graph G_{co} where we add an edge between two variables x,y, only if the tables containing these two tuples are adjacent in the table-adjacency graph G_T. Algorithm 1 CompCoTable constructs the co-table graph G_C by a single *bottom-up* pass over the graph H.

It is easy to see that a minor modification of the same algorithm can be used to compute the co-occurrence graph G_{co}: in Step 11 we simply skip the check whether the tables containing the two tuples are adjacent in G_T. Since this is the only place where

Algorithm 1 *Algorithm* CompCoTable

Input: Query plan DAG H and table-adjacency graph G_T

Output: Co-table graph G_C.

```
1:  – Initialize $V(G_C) = \texttt{Var}(f)$, $E(G_C) = \phi$.
2:  – For all variables $x \in \texttt{Var}(f)$, set $\texttt{Var}(x) = \{x\}$.
3:  – Do a topological sort on $H$ and reverse the sorted order.
4:  for every node $u \in V(H)$ in this order do
5:     /* Update $\texttt{Var}(u)$ set for both $+$-node and $\cdot$-node $u$*/
6:     – Set $\texttt{Var}(u) = \bigcup_v \texttt{Var}(v)$, where the union is over all successors $v$ of $u$.
7:     if $u \in V(H)$ is a $\cdot$-node then
8:        /* Add edges to $G_C$ only for a $\cdot$-node*/
9:        – Let $v_1,v_2$ be its two successors.
10:       for every two variables $x \in \texttt{Var}(v_1)$ and $y \in \texttt{Var}(v_2)$ do
11:          if (i) the tables containing $x,y$ are adjacent in $G_T$ and (ii) the edge $(x,y)$ does
             not exist in $E(G_C)$ yet then
12:             – Add an edge between $x$ and $y$ in $E(G_C)$.
13:          end if
14:       end for
15:    end if
16: end for
```

G_T is used, the time for the computation of G_C does not include the time related to computing/checking G_T.

Correctness. By a simple induction, it can be shown that the set $\texttt{Var}(u)$ is correctly computed at every step, i.e., it contains the set of all nodes which are reachable from u in H (since the nodes are processed in reverse topological order and $\texttt{Var}(u)$ is union of $\texttt{Var}(v)$ for over all successors v of u). Next lemma shows that algorithm CompCoTable correctly builds the co-table graph G_C (proof is in Appendix A.1.2).

Lemma 3.12. *Algorithm* CompCoTable *adds an edge (x,y) to G_C if and only if x,y together appear in some implicant in f_{IDNF} and the tables containing x,y are adjacent in G_T.*

Time Complexity. Here we give a sketch of the time complexity analysis, details can be found in the appendix (Section A.1.3). Computation of the table adjacency graph takes $O(k^2 \alpha \log \alpha)$ time as shown in Section 3.3.2. The total time complexity of algorithm CompCoTable as given in Theorem 3.8 is mainly due to two operations: (i) computation of the $\texttt{Var}(u)$ set at every internal node $u \in V(H)$, and (ii) to perform the test for pairs x, y at two distinct children of a $\cdot$-node, whether the edge (x, y) already exists in G_C, and if not, to add the edge.

We show that the total time needed for the first operation is $O(nm_H)$ in total: for every internal node $u \in V(H)$ we can scan the variables sets of all its immediate successor in $O(nd_u)$ time to compute $\texttt{Var}(u)$, where d_u is the outdegree of node u in H. This gives total $O(nm_H)$ time. On the other hand, for adding edges (x, y) in G_C, it takes total $O(m_{co} \beta_H)$ time during the execution of the algorithm, where m_{co} is the number of edges in the co-occurrence graph (and not in the co-table graph, even if we compute the co-table graph G_C) and β_H is the width of the graph H. To show this, we show that two variables x, y are considered by the algorithm at Step 10 if and only if the edge (x, y) already exists in the co-occurrence graph G_{co}, however, the edge may not be added to the co-table graph G_C if the corresponding tables are not adjacent in the table adjacency graph G_T. We also show that any such edge (x, y) will be considered at a unique level of the DAG H. In addition to these operations, the algorithm does initialization and a topological sort on the vertices which take $O(m_H + n_H)$ time ($n_H = |V(H)|$) and are dominated by the these two operations.

3.4 Computing the Read-Once Form

Our algorithm CompRO (for *Compute Read-Once*) takes an instance I of the schema $\mathbf{R} = R_1, \cdots, R_k$, a query $Q() : -R_1(\mathbf{x_1}), R_2(\mathbf{x_2}), \cdots, R_k(\mathbf{x_k})$ along with the table adjacency graph G_T and co-table graph G_C computed in the previous section as input, and outputs whether $Q(I)$ is read-once. (if so it computes its unique read-once form).

Theorem 3.13. *Suppose we are given a query Q, a table-independent database representation I, the co-table graph G_C and the table-adjacency graph G_T for Q on I as inputs. Then*

1. *Algorithm* CompRO *decides correctly whether the expression generated by evaluating Q on*

I is read-once, and if yes, it returns the unique read-once form of the expression, and,

2. *Algorithm* CompRO *runs in time $O(m_T \alpha \log \alpha + (m_C + n) \min(k, \sqrt{n}))$,*

where $m_T = |E(G_T)|$ is the number of edges in G_T, $m_C = |E(G_C)|$ is the number of edges in G_C, n is the total number of tuples in I, k is the number of tables, and α is the maximum size of any subgoal.

3.4.1 Algorithm CompRO

In addition to the probabilistic database with tables $R_1, \cdots, R_k$ and input query Q, our algorithm also takes the *table-adjacency graph* G_T and the *co-table graph* G_C computed in the first phase as discussed in Section 3.3. The co-table graph G_C also helps us to remove *unused tuples* from all the tables which do not appear in the final expression – every unused tuple won't have a corresponding node in G_C. So from now on we can assume wlog. that every tuple in every table appears in the final expression f.

The algorithm CompRO uses two decomposition operations: *Row decomposition* is a *horizontal* decomposition operation which partitions the rows or tuples in every table into the same number of groups and forms a set of sub-tables from every table. On the other hand, *Table decomposition* is a *vertical* decomposition operation. It partitions the set of tables into groups and a *modified sub-query* is evaluated in every group. For convenience, we will represent the instance I as $R_1[T_1], \cdots, R_k[T_k]$, where T_i is the set of tuples in table R_i. Similarly, for a subset of tuples $T'_i \subseteq T_i$, $R_i[T'_i]$ will denote the instance of relation R_i containing exactly the tuples in T'_i. The algorithm CompRO is given in Algorithm 2.

Row Decomposition. The row decomposition operation partitions the tuples variables in *every* table into ℓ disjoint groups. In addition, it decomposes the co-table graph G_C into $\ell \geq 2$ disjoint *induced subgraphs*[17] corresponding to the above groups. For every pair of distinct groups j, j', and for every pair of distinct tables $R_i, R_{i'}$, no tuple in group j of R_i ever joins with a tuple in group j' of $R_{i'}$ (recall that the query does not have any self-join operation). The procedure for row decomposition is given in Algorithm 3. It should be

[17]A subgraph H of G is an induced subgraph, if for any two vertices $u,v \in V(H)$, if $(u,v) \in E(G)$, then $(u,v) \in E(H)$.

Algorithm 2 CompRO(Q, $I = \langle R_1[T_1], \cdots, R_k[T_k] \rangle$, G_C, G_T, Flag)

Input: Query Q, tables $R_1[T_1], \cdots, R_k[T_k]$, co-table graph G_C, table-adjacency graph G_T, and a Boolean parameter Flag which is true if and only if row decomposition is performed at the current step.

Output: If successful, the unique read-once form f^* of the expression for $Q(I)$

```
 1: if $k = 1$ then
 2:   return $\sum_{x \in T_1} x$ with success. (/* all unused tuples are already removed */)
 3: end if
 4: if Flag = True then {/* Row decomposition */}
 5:   – Perform RD($\langle R_1[T_1], \cdots, R_k[T_k] \rangle, G_C$).
 6:   if row decomposition returns with success then {/* RD partitions every table and $G_C$
      into $\ell \geq 2$ disjoint groups*/}
 7:     – Let the groups returned be $\langle \langle T_1^j, \cdots, T_k^j \rangle, G_{C,j} \rangle$, $j \in [1, \ell]$.
 8:     – $\forall j \in [1, \ell]$, let $f_j$ = CompRO($Q$, $\langle R_1[T_1^j], \cdots, R_k[T_k^j] \rangle$, $G_{C,j}, G_T$, False).
 9:     return $f^* = f_1 + \cdots + f_\ell$ with success.
10:   end if
11: else {/* Table decomposition */}
12:   – Perform TD($\langle R_1[T_1], \cdots, R_k[T_k] \rangle$, $Q$, $G_T, G_C$).
13:   if table decomposition returns with success then {/* TD partitions $I, G_C$ and $G_T$ into
      $\ell \geq 2$ disjoint groups, $\sum_{j=1}^{\ell} k_j = k$ */}
14:     Let the groups returned be $\langle \langle R_{j,1}, \cdots, R_{j,k_j} \rangle, \widehat{Q_j}, G_{C,j}, G_{T,j} \rangle$, $j \in [1, \ell]$.
15:     – $\forall j \in [1, \ell]$, $f_j$ = CompRO($\widehat{Q_j}$, $\langle R_1[T_1], \cdots, R_k[T_k] \rangle$, $G_{C,j}, G_{T,j}$, True).
16:     return $f^* = f_1 \cdot \ldots \cdot f_\ell$ with success.
17:   end if
18: end if
19: if the current operation is not successful then {/* Current row or table decomposition is
    not successful and $k > 1$*/}
20:   return with failure: "$Q(I)$ is not read-once".
21: end if
```

noted that the row decomposition procedure may be called on $R_{i_1}[T'_{i_1}], \cdots, R_{i_p}[T'_{i_p}]$ and G'_C, where $R_{i_1}, \cdots, R_{i_p}$ is a subset of the relations from $R_1, \cdots, R_k$, $T'_{i_1}, \cdots, T'_{i_p}$ are subsets of the respective set of tuples $T_{i_1}, \cdots, T_{i_p}$, and G'_C is the induced subgraph of G_C on $T'_{i_1}, \cdots, T'_{i_p}$. For simplicity in notations, we use $R_1[T_1], \cdots, R_k[T_k]$. This holds for table decomposition as well.

Algorithm 3 RD($\langle R_1[T_1], \cdots, R_k[T_k] \rangle, G'_C$)

Input: Tables $R_1[T_1], \cdots, R_k[T_k]$**, and induced subgraph** G'_C **of** G_C **on** $\bigcup_{i=1}^{k} T_i$

Output: If successful, the partition of G'_C **and tuple variables of every input tables into** $\ell \geq 2$ **connected components:** $\langle \langle T_{1,j}, \cdots, T_{k,j} \rangle, G'_{C,j} \rangle$, $j \in [1, \ell]$

```
1: – Run BFS or DFS to find the connected components in $G'_C$.
2: – Let $\ell$ be the number of connected components.
3: **if** $\ell = 1$ **then** {/* *there is only one connected component* */}
4:     **return** with failure: "Row decomposition is not possible".
5: **else**
6:     – Let the tuples (vertices) of table $R_i$ in the $j$-th connected component $j$ of $G'_C$ be $T_{i,j}$
7:     – Let the induced subgraph for connected component $j$ be $G_{C,j}$.
8:     **return** $\langle \langle T_{1,1}, \cdots, T_{k,1} \rangle, G'_{C,1} \rangle, \cdots, \langle \langle T_{1,\ell}, \cdots, T_{k,\ell} \rangle, G'_{C,\ell} \rangle$ with success.
9: **end if**
```

Table Decomposition. On the other hand, the table decomposition operation partitions the set of tables $\mathbf{R} = R_1, \cdots, R_k$ into $\ell \geq 2$ disjoint groups $\mathbf{R_1}, \cdots, \mathbf{R_\ell}$. It also decomposes the table-adjacency graph G_T and co-table graph G_C into ℓ disjoint induced subgraphs $G_{T,1}, \cdots, G_{T,\ell}$, and, $G_{C,1}, \cdots, G_{C,\ell}$ respectively corresponding to the above groups. The groups are selected in such a way that all tuples in the tables in one group join with all tuples in the tables in another group. This procedure also modifies the sub-query to be evaluated on every group by making the subqueries of different groups mutually independent by introducing free variables, i.e., they do not share any common variables after a successful table decomposition. Algorithm 4 describes the table decomposition operation. Since the table decomposition procedure changes the input query Q to $\widehat{Q} = \widehat{Q_1}, \cdots, \widehat{Q_\ell}$, it is crucial to ensure that changing the query to be evaluated does not change the answer to the final expression (see Lemma A.5 in Appendix A.1.4).

The following lemma shows that if row-decomposition is successful, then table decomposition cannot be successful and vice versa. However, both of them may be unsuccessful when the Boolean provenance is not read-once (proof is in Appendix A.1.4).

Lemma 3.14. *At any step of the recursion, if row decomposition is successful then table decomposition is unsuccessful and vice versa.*

Therefore, in the top-most level of the recursive procedure, we can verify which operation can be performed – if both of them fail, then the final expression is not read-once which follows from the correctness of our algorithm. If the top-most recursive call performs a successful row decomposition initially the algorithm CompRO is called as CompRO(Q, $\langle R_1[T_1], \cdots, R_k[T_k]\rangle$, G_C, G_T, True). The last Boolean argument is True if and only if row decomposition is performed at the current level of the recursion tree. If in the first step table decomposition is successful, then the value of the last Boolean variable in the initial call will be False.

Correctness. The following two lemmas respectively show the soundness and completeness of the algorithm CompRO (proofs are in Appendix A.1.4).

Lemma 3.15. ***(Soundness)*** *If the algorithm returns with success, then the expression f^* returned by the algorithm* CompRO *is equivalent to the expression $Q(I)$ generated by evaluation of query Q on instance I. Further, the output expression f^* is in read-once form.*

Lemma 3.16. ***(Completeness)*** *If the expression $Q(I)$ is read-once, then the algorithm* CompRO *returns the unique read-once form f^* of the expression.*

For completeness, it suffices to show that if $Q(I)$ is read-once, then the algorithm does not exit with error. Indeed, if the algorithm returns with success, as showed in the soundness lemma, the algorithm returns an expression f^* in read-once form which is the unique read-once form of $Q(I)$ [85],[47].

Time Complexity. Consider the recursion tree of the algorithm CompRO. Lemma 3.14 shows that at any level of the recursion tree, either all recursive calls use the row decomposition procedure, or all recursive calls use the column decomposition procedure. The time complexity of CompRO given in Theorem 3.13 is analyzed in the following steps. If

Algorithm 4 TD($\langle R_1[T_1], \cdots, R_k[T_k]\rangle$, $\langle Q() :- R_1(\mathbf{x_1}), \cdots, R_k(\mathbf{x_k})\rangle$, G'_C, G'_T)

Input: Tables $R_1[T_1], \cdots, R_k[T_k]$ query $Q() :- R_1(\mathbf{x_1}), \cdots, R_k(\mathbf{x_k})$ induced subgraph G'_T of G_T on $\bigcup_{i=1}^{k} R_i$, induced subgraph G'_C of G_C on $\bigcup_{i=1}^{k} T_i$

Output: If successful, a partition of input tables, G'_T, G'_C into ℓ groups, and an updated sub-query for every group

```
1:  for all edges $e = (R_i, R_j)$ in $G'_T$ do
2:      – Annotate the edge $e$ with common variables $C_e$ in the vectors $\mathbf{x_i}$, $\mathbf{x_j}$.
3:      – Mark the edge $e$ with a "$+$" if for every pair of tuple variables $x \in T_i$ and $y \in T_j$,
        the edge $(x, y)$ exists in $G'_C$. Otherwise mark the edge with a "$-$".
4:  end for
5:  – Run BFS or DFS to find the $\ell$ connected components in $G_T$ w.r.t "$-$" edges
6:  if $\ell = 1$ then {/* there is only one connected component */}
7:      return  with "Failure: Table decomposition is not possible".
8:  else
9:      – Let $G'_{T,1}, \cdots, G'_{T,\ell}$ be the induced subgraphs of $\ell$ connected components of $G'_T$ and
        $G'_{C,1}, \cdots, G'_{C,\ell}$ be the corresponding induced subgraph for $G'_C$.
10:     – Let $\mathbf{R_p} = \langle R_{p,1}, \cdots, R_{p,k_p}\rangle$ be the tables in the $p$-th component of $G'_T$, $p \in [1, \ell]$.
11:     for every component $p$ do {/* Compute a new query for every component */}
12:         for every table $R_i$ in this component $p$ do
13:             – Let $C_i = \bigcup_e C_e$ be the union of common variables $C_e$ over all edges $e$ from $R_i$
                to tables in different components of $G_{T'}$ (all such edges are marked with '+')
14:             – For every common variable $z \in C_i$, generate a new (free) variable $z^i$, and re-
                place all occurrences of $z$ in vector $\mathbf{x_i}$ by $z'$. Let $\widehat{\mathbf{x_i}}$ be the new vector.
15:             – Change the query subgoal for $R_i$ from $R_i(\mathbf{x_i})$ to $R_i(\widehat{\mathbf{x_i}})$.
16:         end for
17:         Let $\widehat{Q_p}() :- R_{p,1}(\widehat{\mathbf{x_{p,1}}}), \cdots, R_{p,k_p}(\widehat{\mathbf{x_{p,k_p}}})$ be the new query for component $p$.
18:     end for
19:     return  $\langle\langle R_{p,1}, \cdots, R_{p,k_1}\rangle, \widehat{Q_p}, G_{C,p}, G_{T,p}\rangle$, $p \in [1, \ell]$ with success.
20: end if
```

n' = the total number of input tuples at the current recursive call and m'_C = the number of edges in the induced subgraph of G'_C on these n' vertices, we show that row decomposition takes $O(m'_C + n')$ time and, *not considering the time needed to compute the modified queries* $\widehat{Q_j}$ (Step 11 in Algorithm 4), the table decompositions procedure takes $O(m'_C + n')$ time. Then we consider the time needed to compute the modified queries and show that these steps over *all* recursive calls of the algorithm take $O(m_T \alpha \log \alpha)$ time in total, where α is the maximum size of a subgoal in the query Q. Finally, we give a bound of $O(\min(k, \sqrt{n}))$ on the height of the recursive tree for the algorithm CompRO. However, note that at every step, for row or table decomposition, every tuple in G'_C goes to exactly one of the recursive calls, and every edge in G'_C goes to at most one of the recursive calls. So for both row and table decomposition at every level of the recursion tree the total time is $O(m_C + n)$. Combining all these observations, the total time complexity of the algorithm is $O(m_T \alpha \log \alpha + (m_C + n) \min(k, \sqrt{n}))$ as stated in Theorem 3.13. The details can be found in Appendix A.1.5.

Example. Here we illustrate our algorithm. Consider the query Q and instance I from Example 3.2 in the introduction. The input query is $Q() : -R(x)S(x,y)T(y)$. In the first phase, the table-adjacency graph G_T and the co-table graph G_C are computed. These graphs are depicted in Fig. 3.5 and Fig. 3.6 respectively.

Now we apply CompRO. There is a successful row decomposition at the top-most recursive call that decomposes G_C into the two subgraphs $G_{C,1}, G_{C,2}$ shown in Fig. 3.7. So the final expression f^* annotating the answer $Q(I)$ will be the *sum* of the expressions f_1, f_2 annotating the answers of Q applied to the relations corresponding to $G_{C,1}$ and $G_{C,2}$ respectively.

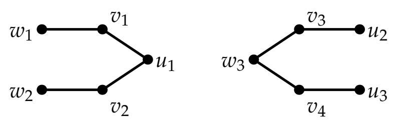


Figure 3.7: $G_{C,1}$ and $G_{C,2}$.

The relations corresponding to $G_{C,1}$ are

$$R = \begin{array}{|c|l} \hline a_1 & w_1 \\ \hline b_1 & w_2 \\ \hline \end{array} \qquad S = \begin{array}{|cc|l} \hline a_1 & c_1 & v_1 \\ \hline b_1 & c_1 & v_2 \\ \hline \end{array} \qquad T = \begin{array}{|c|l} \hline c_1 & u_1 \\ \hline \end{array}$$

and, the relations corresponding to $G_{C,2}$ are

$$R = \begin{array}{|c|l} \hline a_2 & w_3 \\ \hline \end{array} \qquad S = \begin{array}{|cc|l} \hline a_2 & c_2 & v_3 \\ \hline a_2 & d_2 & v_4 \\ \hline \end{array} \qquad T = \begin{array}{|c|l} \hline c_2 & u_2 \\ \hline d_2 & u_3 \\ \hline \end{array}$$

Now we focus on the first recursive call at the second level of recursion tree with input co-table subgraph $G_{C,1}$. Note that the table-adjacency graph for this call is the same as G_T. At this level the table decomposition procedure is invoked and the edges of the table-adjacency graph are marked with $+$ and $-$ signs, see Fig. 3.8. In this figure the common variable set for R, S on the edge (R, S) is $\{x\}$, and for S, T on the edge (S, T) is $\{y\}$. Further, the edge (S, T) is marked with a "$+$" because there are all possible edges between the tuples in S (in this case tuples v_1, v_2) and the tuples in T (in this case u_1). However, tuples in R (here w_1, w_2) and tuples in S (here v_1, v_2) do not have all possible edges between them so the edge (R, S) is marked with a "$-$".

Table decomposition procedure performs a connected component decomposition using "$-$"-edges, that decomposes G_T in two components $\{R, S\}$ and $\{T\}$. The subset C of common variables collected from the "$+$"-edges across different components will be the variables on the single edge (S, T), $C = \{y\}$. This variable y is replaced by new *free variables* in all subgoals containing it, which are S and T in our case. So the modified queries for disjoint components returned by the table decomposition procedure are $\widehat{Q_1}() : -R(x)S(x, y_1)$ and $\widehat{Q_2}() : -T(y_2)$. The input graph $G_{C,1}$ is decomposed further into $G_{C,1,1}$ and $G_{C,1,2}$, where $G_{C,1,1}$ will have the edges (w_1, v_1) and (w_2, v_2), whereas $G_{C,1,2}$ will have no edges and a single vertex u_1. Moreover, the expression f_1 is the product of f_{11} and f_{12} generated by these two queries respectively. Since the number of tables for $\widehat{Q_2}$ is only one, and T has a single tuple, by the base step (Step 2) of CompRO, $f_{12} = u_1$. For expression f_{11} from $\widehat{Q_1}$, now the graph $G_{C,1,1}$ can be decomposed using a row decomposition to two connected components with single edges each ((w_1, v_1) and (w_2, v_2) respectively). There will be recursive subcalls on these two components and each one of them will perform a table decomposition (one tuple in every table, so the single edges in both calls will be marked with "$+$"). Hence f_{11} will be evaluated to $f_{11} = w_1v_1 + w_2v_2$. So $f_1 = f_{11} \cdot f_{12} = (w_1v_1 + w_2v_2)u_1$.

By a similar analysis as above, it can be shown that the same query Q evaluated on the

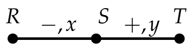


Figure 3.8: Marked table-adjacency graph for R, S, T.

tables R, S, T given in the above tables give $f_2 = w_3(v_3v_4 + u_2u_3)$. So the overall algorithm is successful and outputs the read-once form $f^* = f_1 + f_2 = (w_1v_1 + w_2v_2)u_1 + w_3(v_3u_2 + v_4u_3)$.

3.5 Discussion

To evaluate our approach, we discuss the time complexity of the entire query-answering algorithm and compare it with the time complexity of the previous algorithms. Putting together our results from Sections 3.3 and 3.4, we propose the following algorithm for answering Boolean conjunctive queries without self-joins on tuple-independent probabilistic databases.

- Phase 0 (Compute provenance DAG)
  Input: query q, event table rep I
  Output: provenance DAG H
  Complexity: $O((\frac{ne}{k})^k)$ *(see below)*

- Phase 1 (Compute co-table graph)
  Input: H, q
  Output: table-adjacency graph G_T, co-table graph G_C
  Complexity: $O(nm_H + \beta_H m_{co} + k^2\alpha\log\alpha)$ (Thm. 3.8)

- Phase 2 (Compute read-once form)
  Input: event table rep I, q, G_T, G_C
  Output: read-once form f^* or FAIL
  Complexity: $O(m_T\alpha\log\alpha + (m_C + n)\min(k, \sqrt{n}))$ (Thm. 3.13)

Size of the provenance DAG H. Let f be the Boolean provenance generated by some query plan for q on the database I. The number of edges m_H in the DAG H represents the size of the expression f. Since there are exactly k subgoals in the input query q, one for every table, every prime implicant of f_{IDNF} will have exactly k variables, so the size of f_{IDNF} is at most $\binom{n}{k} \leq (\frac{ne}{k})^k$. Further, the size of the input expression f is maximum when f is already in IDNF. So size of the DAG H is upper bounded by $m_H \leq (\frac{ne}{k})^k$. Again, the "leaves" of the DAG H are exactly the n variables in f. So $m_H \geq n - 1$, where the lower bound is achieved when the DAG H is a tree (every node in H has a unique predecessor); in that case f must be read-once and H is the unique read-once tree of f.

Therefore $n - 1 \leq m_H \leq (\frac{ne}{k})^k$. Although the upper bound is quite high, it is our contention that for practical query plans the size of the provenance graph is much smaller than the size of the corresponding IDNF.

Data complexity. The complexity dichotomy of [55] is for data complexity, i.e., the size of the query is bounded by a constant. This means that our k and α are $O(1)$. Hence the time complexities of the Phase 1 and Phase 2 are $O(nm_H + \beta_H m_{co})$ and $O(m_C + n)$ respectively. Since the co-occurrence graph is at least as large as the co-table graph $m_{co} \geq m_C$, and therefore the first phase always dominates asymptotically.

In any case, we take the same position as [156] that for unsafe queries [55] the competition comes from the approach that does *not* try to detect whether the formulas are read-once and instead uses probabilistic inference [111] which is in general EXPTIME. In contrast, our algorithm runs in PTIME, and works for a larger class of queries than the safe queries [55] (but of course, not on all instances).

Comparisons with other algorithms. Here we do not restrict ourselves to data complexity, instead taking the various parameters of the problem into consideration.

First consider the general read-once detection algorithm. This consists of choosing some plan for the query, computing the answer Boolean provenance f, computing its IDNF, and then using the (so far, best) algorithm [85] to check if f is read-once and if so to compute its read-once form. The problem with this approach is that the read-once check is indeed done in time a low polynomial, but *in the size of* f_{IDNF}. For example, consider a Boolean query like the one in Example 3.6. This is a query that admits a plan (the safe plan!) that would generate the Boolean provenance $(x_1 + y_1) \cdots (x_n + y_n)$ on an instance in which each R_i has two tuples x_i and y_i. This is a read-once expression easily detected by our algorithm, which avoids the computation of the IDNF.

Next consider the cograph-help algorithm that we have already mentioned and justified in Section 3.2. This consists of our Phase 0 and a slightly modified Phase 1 that computes the co-occurrence graph G_{co}, followed by checking if G_{co} is a cograph using one of the linear-time algorithms given in [29, 49, 92] which also outputs the read-once form if possible. Since Phase 0 and Phase 1 are common we only need to compare the last phases.

The co-graph recognition algorithms will all run in time $O(m_{co} + n)$. Our Phase 2 complexity is better than this when $m_C \min(k, \sqrt{n}) = o(m_{co})$ (for instance, when we consider data complexity assuming that the query size is much smaller than the size of the data). Although in the worst case this algorithm performs at least as well as our algorithm (when m_C is $\theta(m_{co})$ and we consider combined complexity including both the data and the query size), (i) almost always the query will be of much smaller size than the data, and therefore the time required in first phases will dominate, so the asymptotic running time of both these algorithms will be comparable, (ii) as we have shown earlier, the ratio $\frac{m_{co}}{m_C}$ can be as large as $\Omega(n^2)$, and the benefit of this could be significantly exploited by caching co-table graphs computed for other queries (see discussions in Section 3.7), and (iii) these linear time algorithms use complicated data structures, whereas we use simple graphs given as adjacency lists and connectivity-based algorithms, so our algorithms are simpler to implement and may run faster in practice.

Finally we compare our algorithm against that given in [156]. Let us call it the lineage-tree algorithm since they take the lineage tree of the result as input as opposed to the provenance DAG as we do. Although [156] does not give a complete running time analysis of the lineage tree algorithm, for the brief discussion we have, we can make, to the best of our understanding, the following observations.

Every join node in the lineage tree has two children, and every project node can have arbitrary number of children. When the recursive algorithm computes the read-once trees of every child of a project node, every pair of such read-once trees are merged which may take $O(n^2k^2)$ time for every single merge (since the variables in the read-once trees to be merged are repeated). Without counting the time to construct the lineage tree this algorithm may take $O(Nn^2k^2)$ time in the worst case, where N is the number of nodes in the lineage tree.

Since [156] does not discuss constructing the lineage tree we will also ignore our Phase 0. We are left with comparing N with m_H. It is easy to see that the number of edges in the provenance DAG H, $m_H = \theta(N)$, where N is the number of nodes in the lineage tree, when both originate from the same query plan[18] Since the lineage-tree algorithm takes $O(Nn^2k^2)$ time in the worst case, and we use $O(nm_H + \beta_H m_{co} + k^2\alpha \log \alpha) + O((m_C + n)\min(k, \sqrt{n})) = O(nN + \beta_H n^2 + k^2\alpha \log \alpha + n^{\frac{5}{2}})$. The width β_H of the DAG H in the worst case can be the number of nodes in H. So our algorithm *always* gives an $O(k^2)$ improvement in time complexity over the lineage-tree algorithm given in [156] whereas the benefit can often be more.

3.6 Related Work

The beautiful complexity dichotomy result of [55] classifying conjunctive queries without self-joins on tuple-independent databases into "safe" and "unsafe" has spurred and intensified interest in probabilistic databases. Some papers have extended the class of safe relational queries [54, 56, 137, 138]. Others have addressed the question of efficient query answering for unsafe queries on *some* probabilistic databases. This includes mixing the intensional and extensional approaches, in effect finding subplans that yield read-once subexpressions in the Boolean provenance [100]. The technique identifies "offending" tuples that violate functional dependencies on which finding safe plans relies and deals with them intensionally. It is not clear that this approach would find the read-once forms that our algorithm finds. The OBDD-based approach in [137] works also for some unsafe queries on some databases. The SPROUT secondary-storage operator [139] can handle efficiently some unsafe queries on databases satisfying certain functional dependencies.

Exactly like us, [156] looks to decide efficiently when the extensional approach is applicable given a conjunctive query without self-joins and a tuple-independent database. We have made comparisons between the two papers in various places, especially in Sec-

[18]If we "unfold" provenance DAG H to create the lineage tree, the tree will have exactly m_H edges, and the number of nodes in the tree will be $N = m_H + 1$.

tion 3.5. Here we only add that that our algorithm deals with different query plans uniformly, while the lineage tree algorithm needs to do more work for non-deep plans. The graph structures used in our approach bear some resemblance to the graph-based synopses for relational selectivity estimation in [160].

The read-once property has been studied for some time, albeit under various names [90, 91, 95, 97, 103, 141, 170]. It was shown [97] that if RP≠NP then read-once cannot be checked in PTIME for arbitrary monotone Boolean formulas, but for formulas in IDNF (as input) read-once can be checked in PTIME [85]. Our result here sheds new light on another class of formulas for which such an efficient check can be done.

3.7 Conclusions

We have investigated the problem of efficiently deciding when a conjunctive query *without self-joins* applied to a *tuple-independent* probabilistic database representation yields result representations featuring *read-once* Boolean provenance (and, of course, efficiently computing their read-once forms when they exist). We have given a complete and simple to implement algorithm of low polynomial data complexity for this problem, and we have compared our results with those of other approaches.

As explained in the introduction, the results of this work do not constitute complexity dichotomies. However, there is some hope that the novel proof of completeness that we give for our algorithm may be of help for complexity dichotomy results in the space coordinated by the type of queries and the type of databases we have studied.

Of independent interest may be that we have also implicitly performed a study of an interesting class of monotone Boolean formulas, those that can be represented by the provenance graphs of conjunctive queries without self-joins (characterizations of this class of formulas that do not mention the query or the database can be easily given). We have shown that for this class of formulas the read-once property is decidable in low PTIME (the problem for arbitrary formulas is unlikely to be in PTIME, unless RP=NP). Along the way we have also given an efficient algorithm for computing the co-occurrence graph of such formulas (in all the other papers we have examined, computing the co-occurrence graph entails an excursion through computing a DNF; this, of course, may be the best one can do for arbitrary formulas, if RP$\neq$NP). It is likely that nicely tractable class of Boolean formulas may occur in other database applications, to be discovered.

For further work one obvious direction is to extend our study to larger classes of queries and probabilistic databases [54, 56]. Recall from the discussion in the introduction however, that the class of queries considered should not be able to generate arbitrary monotone Boolean expressions. Thus, SPJU queries are too much (but it seems that our approach might be immediately useful in tackling unions of conjunctive queries without self-joins, provided the plans do the unions last).

On the more practical side, work needs to be done to apply our approach to non-Boolean queries, i.e., they return actual tables. Essentially, one would work with the provenance graph associated with each table (initial, intermediate, and final) computing simultaneously the co-table graphs of the Boolean provenance on the graph's roots. It is likely that these co-table graphs can be represented together, with ensuing economy.

However we believe that the most practical impact would have the *caching of co-table graphs* at the level of the system, over batches of queries on the same database, since the more expensive step in our algorithm is almost always the computation of the co-table graph (see discussion in Section 3.5).

This would work as follows, for a fixed database I. When a (let's say Boolean for simplicity) conjunctive query Q_1 is processed, consider also the query $\bar{Q}_1$ which is obtained from Q_1 by replacing each *occurrence* of constants with a distinct fresh FO variable. Moreover if an FO variable x occurs several times in a subgoal $R_i(\mathbf{x}_i)$ of q but does *not* occur in any of the other subgoals (i.e., x causes selections but not joins), replace also each occurrence of x with a distinct fresh FO variable. In other words, Q_1 is doing what $\bar{Q}_1$ is doing, but it first applies some selections on the various tables of I. We can say that $\bar{Q}_1$ is the "join pattern" behind Q_1. Next, compute the co-table graph for $\bar{Q}_1$ on I and *cache* it together with $\bar{Q}_1$. It is not hard to see that the co-table graph for Q_1 can be efficiently computed from that of $\bar{Q}_1$ by a "clean-up" of those parts related to tuples of I that do not satisfy the select conditions of Q_1.

When another query Q_2 is processed, check if its join-pattern $\bar{Q}_2$ *matches* any of the join-patterns previously cached (if not, we further cache *its* join-pattern and co-table graph). Let's say it matches $\bar{Q}_1$. Without defining precisely what "matches" means, its salient property is that the co-table graph of $\bar{Q}_2$ can be efficiently obtained from that of $\bar{Q}_2$ by another clean-up, just of edges, guided by the table-adjacency graph of $\bar{Q}_2$ (which is the same as that of Q_2). It can be shown that these clean-up phases add only an $O(n\alpha)$ to the running time.

There are two practical challenges in this approach. The first one is efficiently finding in the cache some join-pattern that matches that of an incoming query. Storing the join-patterns together into some clever data structure might help. The second one is storing largish numbers of cached co-table graphs. Here we observe that they can all be stored with the same set of nodes and each edge would have a list of the co-table graph it which it appears. Even these lists can be large, in fact the number of all possible joint-patterns is exponential in the size of the schema. More ideas are needed and ultimately the viability of this caching technique can only be determined experimentally.

Chapter 4

Queries with Difference on Probabilistic Databases

This chapter presents our second work in efficiently computing the uncertainty in the output from uncertain inputs in probabilistic databases. We considered positive conjunctive queries in the previous chapter. In this chapter, we initiate the study of queries with difference operations in probabilistic databases. Computing the probabilities of the outputs becomes harder in the presence of difference operations. We explore properties of queries with difference and Boolean provenance from these queries that allow efficient computation of the probabilities of the outputs.

4.1 Overview

We have reviewed previous work in probabilistic databases for positive queries in Sections 1.1.1 and 2.1.1. However, queries with relational *difference operations* are also important for both theoretical and practical purposes. Practical SQL queries make use of difference not just explicitly (the EXCEPT construct) but also implicitly as when SELECT subqueries are nested within WHERE clauses with logically complex conditions. The concept of boolean provenance and the intensional semantics extend to difference. This was essentially shown in an early seminal paper by Imielinski and Lipski, and sets of tuples annotated by boolean provenance form particular cases of their *c*-tables [98].

Here is an example. Consider a relation $R = R_1 - R_2$ where the tuples in R_1, R_2 are annotated with boolean provenance. For a tuple t to appear in R, t must appear in R_1, let's say with boolean provenance ϕ. If t does not appear in R_2, then the boolean provenance of t in R will be also ϕ; on the other hand if t appears in R_2, let's say with boolean provenance ψ, then the boolean provenance of t in R will be $\phi \wedge \overline{\psi}$. Figure 4.1 shows the boolean provenance for the difference query $q' - q$ on the tuple independent probabilistic database in Figure 2.1 where q, q' are the queries given in Figures 2.2 and 2.3.

b_1	$(u_1v_1w_1 + u_1v_2w_2 + u_3v_4w_2) \cdot \overline{(u_1(v_1 + v_2) + u_3v_4)}$
b_2	$(u_2v_3w_3) \cdot \overline{(u_2v_3)}$

Figure 4.1: Boolean provenance for the query $q - q'$.

However, there are some new and considerable difficulties with such queries. The following proposition holds for positive queries.

Proposition 4.1. *For any positive (UCQ/SPJU) query q and for any probabilistic database I where $|I| = n$, the DNF* [19] *of the boolean provenance of the tuples in $q(I)$ can be computed in time polynomial in n (and so it also has poly size).*

When difference is added, Proposition 4.1 fails. To see this consider the simple difference query $q() : -\text{True} \; - R(x,y), S(y,z)$, which is true if and only if no two tuples in R and S join. Suppose in the database instance I, the tuples in R are $R(a_i, b_i)$, annotated by the variables x_i, $i = 1$ to m, and the tuples in S are $S(b_i, c_i)$, annotated by the variables y_i, $i = 1$ to m. Then the total number of tuples in I is $n = 2m$. In this case the boolean provenance of the answer "True" in $q(I)$ is

$$\overline{(x_1 \cdot y_1) + \ldots + (x_m \cdot y_m)} \quad = \quad (\overline{x_1} + \overline{y_1}) \cdot \ldots \cdot (\overline{x_m} + \overline{y_m})$$

If we expand this boolean provenance into IDNF, then its size will be $2^m = 2^{n/2}$, exponential in $n = |I|$ [20]. This precludes using either the read-onceness testing algorithm of [85]

[19]Through this chapter we use DNF for IDNF or Irredundant Disjunctive Normal Form of an expression, i.e., a disjunction of minterms that cannot be further shrunk by idempotence or absorption.

[20]However, this is an "easy" scenario and there is a simple poly-time algorithm that can compute the answer probability using the read-onceness of the resulting boolean provenance.

and the FPRAS for DNF counting in [104]. The latter is already observed by Koch [110] who also discusses techniques for avoiding the difference operation in some practical applications.

In a recent and independent work [77], Fink et. al. proposed a framework to compute exact and approximate probabilities for answers of arbitrary relational algebra queries that allow difference operations. They extended their approach in [140] that computes a *d-tree* given a DNF expression by repeated use of independent AND, independent OR and Shannon's expansion, and showed that the Boolean provenance of the answer is not required to be converted to DNF for queries with difference. Though this approach always returns an exact or approximate probability of the answer, the size of the d-tree is not guaranteed to be of poly-size in the input expression and therefore the running time may not be polynomial in the worse case. In fact, in this work we show that it is not possible to get any non-trivial poly-time approximation of the answer probability for very simple queries with difference under standard complexity assumptions.

In [106], we study the complexity of computing the probability of relational queries *with difference* on tuple-independent databases in detail. Here is a summary of our contributions:

1. Our *first result* is negative, and a somewhat surprising one. We exhibit two Boolean queries q_1 and q_2, both of which are very nice by themselves, namely they are *safe* CQ^-s, but such that computing the probability of $q_1 - q_2$ is #P-hard. The proof is complex, involving reductions from counting independent sets in 3-regular bipartite graphs to counting edge covers in another special class of bipartite graphs to counting satisfying assignments of certain configurations of Boolean expressions[21]. This hardness of exact computation result suggests that any class of interesting queries with difference which are *safe* in the spirit of [54–56] would have to be severely restricted[22].

[21]It has recently been brought to our attention that this result also follows from a hardness result in [54]. However, our proof uses completely different ideas and also shows #P-hardness of some natural graph problems which may be of independent interest.

[22]There is always the obvious easy case when the events associated with q_1 and with q_2 are independent of each other because, for example, they operate on separate parts of the input database. We consider this

2. In view of the above lower bound for exact computation we turn to approximate computation of the probabilities. Our *second result* gives an FPRAS for computing probabilities for large classes of queries with difference. In particular, a corollary of our result applies to queries of the form $q_1 - q_2$ where q_1 is an arbitrary positive query (UCQ/SPJU) while q_2 is a safe query in CQ^-.

   In fact, our full FPRAS result is not restricted to a single differences of positive queries. In particular, it gives an FPRAS for any relational algebra query q with multiple uses of difference as long as they are restricted as follows: (a) If $q_1 \bowtie q_2$ is a subquery of q then at least one of q_1, q_2 must be positive, (b) If $q_1 - q_2$ is a subquery of q then q_1 must be positive and q_2 must be a safe query in CQ^-.

   Following the motivation in Chapter 3, we also study the instance-by-instance approach, *i.e.*, even if the query q does not follow the above restrictions, whether we can exploit the structure of the Boolean provenance of the answers in $q(I)$. In this scenario, the requirement (b) above can be relaxed to $q_2(I)$ having read-once boolean provenance on a given I. To allow for instance-by-instance approaches in our analysis, we state our full FPRAS result in terms of requirements on the boolean provenance of the queries, using combinations of DNFs and d-DNNFs. The proof uses a new application of the Karp-Luby framework [104] that goes well beyond DNF counting.

3. For difference of two positive queries, like $q = q_1 - q_2$, the restriction of q_2 being a safe query is important for our approximation result stated above, because our *third result* shows the *inapproximability* of computing the probability of "*True* $-$ q" where *True* is the Boolean query that is always true while q is the Boolean CQ $q() :- S(x), R(x,y), S(y)$ which has a self-join. The three results give a simple summary of the situation for differences of positive queries, see Table 1.1.

Organization. In Section 4.2 we introduce the notion of difference rank, review some knowledge compilation techniques like read-onceness and d-DNNFs, and describe the connection between graph configurations (such as cliques or independent sets) and truth

case uninteresting.

assignments that satisfy the boolean provenance of certain queries. In Section 4.3 we give our #P-hardness result for exact computation of probability of tuples produced by difference queries $q_1 - q_2$, where both q_1,q_2 are safe for UCQ. Section 4.4 gives our inapproximability results for general SPJUD queries with difference rank 1, as well as an FPRAS when the boolean provenance are in a special form. We also discuss classes of queries that produce this special form. In Section 4.5 we review some of the related work. Finally we conclude in Section 4.6 with directions for future work.

4.2 Preliminaries

In this section, we give some definitions and review existing knowledge compilation techniques. We will discuss some applications of queries with difference to count graph configurations.

4.2.1 Difference Rank

As we have seen, for queries with difference the DNF of the boolean provenance can have exponential size. Thus, it is natural to study queries in which difference is used in restricted ways. The first step toward this is to limit the amount of "nesting" involving difference in a relational algebra expression. We do it through the following definition.

Definition 4.2. *The* difference rank *of a relational algebra expression is a function $\delta : \mathcal{R}A \to \mathbb{N}$ defined inductively as:*

- (Base relation) $\delta(R) = 0$,
- (Project) $\delta(\Pi q) = \delta(q)$,
- (Select) $\delta(\sigma q) = \delta(q)$,
- (Join) $\delta(q_1 \bowtie q_2) = \delta(q_1) + \delta(q_2)$,
- (Union) $\delta(q_1 \cup q_2) = \max(\delta(q_1), \delta(q_2))$,
- (Difference) $\delta(q_1 - q_2) = \delta(q_1) + \delta(q_2) + 1$.

The queries of difference rank 0 are exactly the positive (SPJU) queries. The positive results in this work will be for queries of difference rank 1, subject to further restrictions. There are natural queries of difference rank 2 however (see subsection 4.2.3). Proposition 4.3 below is an extension of Proposition 4.1 and it justifies our focus on queries of difference rank 1. It shows that their boolean provenance has a certain structure that is potentially more manageable, and indeed, we will exploit this structure in designing approximation algorithms (Section 4.4).

Proposition 4.3. *For any relational algebra query q such that $\delta(q) \leq 1$, and for any probabilistic database I where $|I| = n$, the boolean provenance of any tuple $t \in q(I)$ can be computed in poly-time in n in the form*

$$\alpha_0 + \sum_{i=1}^{r} \alpha_i \overline{\beta_i}$$

where each of $\alpha_0, \cdots, \alpha_r$ and $\beta_1, \cdots, \beta_r$ is a monotone DNF in poly-size in n while r is also polynomial in n; moreover, if $\delta(q) = 0$, then $r = 0$ (we have a single DNF α_0).

We prove the above proposition in Section A.2.1 in the appendix.

4.2.2 Read-Once and d-DNNF

A Boolean expression ϕ is said to be in *read-once form* if every variable $x \in \mathtt{Var}(\phi)$ appears *exactly once*, where $\mathtt{Var}(\phi)$ denotes the set of variables in ϕ. An expression having an equivalent read-once form is called *read-once*. For example, $\overline{x_1}(x_2 + \overline{x_3}x_4) + x_5x_6$ is read-once but $\overline{x_1}(x_2 + \overline{x_3}x_4) + x_4x_6$ is not. The expression $xy + yz$ is read-once but $xy + yz + zx$ is not. A read-once expression ϕ can be naturally represented by a read-once-tree (see Figure 4.2 (a)).

A *d-DNNF* (for *deterministic decomposable negation normal form*), introduced by Darwiche [58, 60], is a rooted DAG, where the leaves are labeled with positive or negative literals x or $\overline{x}$, and internal nodes are $+$-nodes or $\cdot$-node (see Figure 4.2). The $\cdot$-nodes are *decomposable*, in the sense that, for any two children u_i and u_j of a $\cdot$-node u, $\mathtt{Var}(\phi_{u_i}) \cap \mathtt{Var}(\phi_{u_j}) = \emptyset$ (i.e., ϕ_{u_i} and ϕ_{u_j} are independent). On the other hand, $+$-nodes are *deterministic*, i.e., for any $+$-node u, and for any two children u_i, u_j of u, the set of assignments of $\mathtt{Var}(\phi)$ that satisfy ϕ_{u_i} and the set of assignments that satisfy ϕ_{u_j} are

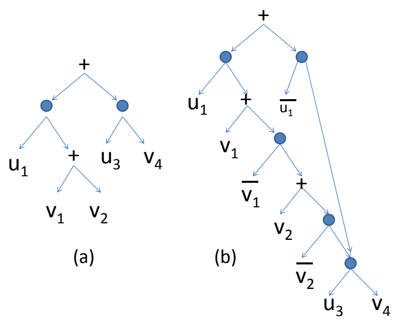


Figure 4.2: $\phi = u_1(v_1 + v_2) + u_3v_4$: (a) as a read-once tree, (b) as a d-DNNF.

disjoint. Similar to read-once expressions, for a Boolean expression ϕ represented as a d-DNNF, $\Pr[\phi]$ can be computed in linear time in the size of the d-DNNF repeatedly using $\Pr[\phi_u] = \Pi_{i=1}^{\ell} \Pr[\phi_{u_i}]$ (if u is a $\cdot$-node) and $\Pr[\phi_u] = \sum_{i=1}^{\ell} \Pr[\phi_{u_i}]$ (if u is a $+$-node), where $u_1, \cdots, u_\ell$ are children of u. The following proposition shows the containment of these knowledge compilation forms which we will exploit while designing our approximation algorithms (the containment is strict [101]):

Proposition 4.4. *If ϕ is in read-once form, then ϕ has a d-DNNF representation of size $O(n)$ ($n = |\mathtt{Var}(\phi)|$) which can be computed in $O(n)$ time from the read-once tree of ϕ.*

The proof of the above proposition follows from [58, 101, 137] using the construction of an OBDD [30] as an intermediate step. We illustrate the read-once-tree and d-DNNF representation of the read-once boolean provenance $u_1(v_1 + v_2) + u_3v_4$ of the tuple $q(b_1)$ from Figures 2.2 (see, for instance, [101] for the exact procedure).

Whether a Boolean expression is read-once was known to be decidable in poly-time (along with the computation of the equivalent read-once form), when ϕ is in DNF [85, 91]; recently [152, 156] gave poly-time algorithm to achieve the same when a Boolean expression ϕ produced by a CQ$^-$ query is given in *any* arbitrary form along with the query plan that produced ϕ. An equivalent d-DNNF for a given Boolean expression may not be compact, and also it is not known whether the Boolean expressions having poly-size d-DNNF expressions are closed under negation [59, 60]. Nevertheless, since read-once Boolean expressions are obviously closed under negation it follows from Proposition 4.4 that if ϕ is read-once, then $\overline{\phi}$ has a d-DNNF representation of size $O(|\mathtt{Var}(\phi)|)$.

4.2.3 From Graphs to Queries

There is a tight connection between graph properties that assert the existence of vertex configurations such as cliques, vertex covers, independent sets, etc., and certain (Boolean) relational algebra queries. Indeed, consider the relational schema (V,E,S) where $V(A), S(A)$ are unary relations while $E(A_1,A_2)$ is binary. The idea is that (V,E) encodes a simple undirected graph while the vertex subset $S \subseteq V$ and describes a *configuration* such as a vertex cover, a clique, etc. in the graph. Not all (V,E)-relational databases encode simple graphs so we restrict attention to those satisfying the constraints $E \subseteq V \times V$, $\forall v_1 v_2\; E(v_1,v_2) \rightarrow E(v_2,v_1)$ and $\forall v\; \neg E(v,v)$.

To express the relational algebra queries we begin with formulating the clique, cover, etc. properties as first-order sentences which we then transform into queries (making sure of domain independence).

Examples

1. S induces a *clique*: $\forall x,y\; S(x) \wedge S(y) \wedge x \neq y \rightarrow E(x,y)$
   $q_{clique} = True - [\sigma_{A_1 \neq A_2}(\rho_{A_1/A} S \times \rho_{A_2/A} S) - E]$
2. S is a *vertex cover*: $\forall x,y\; E(x,y) \rightarrow S(x) \vee S(y)$
   $q_{v-cover} = True - [E \bowtie \rho_{A_1/A}(V - S) \bowtie \rho_{A_2/A}(V - S)]$
3. S is an *independent set*: $\forall x,y\; S(x) \wedge S(y) \rightarrow \neg E(x,y)$
   $q_{ind-set} = True - [E \bowtie \rho_{A_1/A} S \bowtie \rho_{A_2/A} S]$

Note that $\delta(q_{ind-set}) = 1$ while $\delta(q_{v-cover}) = \delta(q_{clique}) = 2$.

Consider probabilistic databases I with schema (V,E,S) such that the tuples in V (the vertices) and E (the edges) have probability 1 while S has the same tuples as V but with arbitrary probabilities. The tuple variables that matter are those for S and they correspond 1-1 to the vertices of the graph. A random instance of S is subset of V and these are in 1-1 correspondence with the truth assignments to the tuple variables. Hence there is a 1-1 correspondence between configurations satisfying the query (e.g., forming a clique, or an independent set) and the truth assignments that make the boolean provenance of the query true. This gives a reduction from counting the configurations to counting the corresponding satisfying assignments (this is further reduced to computing probability by choosing all probabilities in S to be $\frac{1}{2}$).

4.3 Hardness of Exact Computation

In this section we give a hardness result for SPJUD queries. We will consider Boolean SPJUD queries of the form $q = q_1 - q_2$, where both q_1, q_2 are CQ^- queries; hence $\delta(q) = 1$. We show that the problem of exact probability evaluation for this class of simple-looking query is hard even in very restricted cases.

Given an SPJUD query $q = q_1 - q_2$ and a probabilistic database instance I, let ϕ_1 and ϕ_2 denote the two boolean provenance-s produced by q_1 and q_2 on I respectively. Hence the boolean provenance of the unique tuple in $q(I)$ will be $\phi = \phi_1 \cdot \overline{\phi_2}$. As we discussed in Section 4.2.2, if ϕ (and therefore $\overline{\phi}$) is a read-once (RO) Boolean expression on independent random variables, then $\Pr[\phi]$ and $\Pr[\overline{\phi}]$ can be computed efficiently. In this section we show that, the exact computation of $\Pr[\phi_1 \cdot \overline{\phi_2}]$ is #P-hard, even when both ϕ_1, ϕ_2 are RO. In fact, we show that the expressions ϕ_1, ϕ_2 can be generated by simple queries q_1 and q_2 of constant size that are individually *safe* for the class CQ^- [55]. The following theorem states the main result proved in this section.

Theorem 4.5. *There exists a fixed-size SPJUD query q of the form $q = q_1 - q_2$ where both q_1, q_2 are safe CQ^- queries, such that the exact computation of the probabilities of the answers is #P-hard.*

Let $\texttt{C}(\phi)$ be the number of satisfying assignments of a Boolean expression ϕ. A tuple-variable x in a Boolean expression or in a probabilistic database instance will be called *uncertain* if the variable x is present with some uncertainty, i.e., $\Pr[x] \notin \{0,1\}$. In our reduction, we will construct a probabilistic database instance where for all uncertain tuple variables x, $\Pr[x] = \frac{1}{2}$. In this scenario, computation of $\Pr[\phi_1.\overline{\phi_2}]$ is equivalent to the computation of the number of satisfying assignments of $\phi_1.\overline{\phi_2}$ (since $\Pr[\phi_1.\overline{\phi_2}] = \frac{\texttt{C}(\phi_1.\overline{\phi_2})}{2^N}$, where $N =$ the number of uncertain tuple variables); hence we focus on the counting version from now on.

To prove Theorem 4.5, intuitively, (i) first we need to construct a Boolean expression ϕ which is expressible as the product of two RO expressions ϕ_1 and $\overline{\phi_2}$, and where the computation of $\Pr[\phi]$ is $\#P$-hard; (ii) then we need to construct two safe Boolean queries q_1, q_2 and a database instance I such that q_1, q_2 produce ϕ_1, ϕ_2 as the boolean provenance of the unique answer tuples in $q_1(I)$ and $q_2(I)$ respectively. Note that if ϕ_1 and ϕ_2 do not share any variables and are RO, then $\Pr[\phi_1 \cdot \overline{\phi_2}] = \Pr[\phi_1] \cdot (1 - \Pr[\phi_2])$ can be easily computed. Hence the challenge in the first step is to find a "hard" expression which can be factored into two "easy " expressions which share variables, whereas, the challenge in the second step is to construct a query that produces these expressions without using self join operation (since ϕ_1 and ϕ_2 share variables, we need to ensure that no two variables from the same relation join in either of them).

Before we describe the steps of the proof of Theorem 4.5, we first define two counting versions of satisfiability problems, one is the product of two RO CNF expressions, and, the other is the product of two RO DNF expressions.

Definition 4.6. *We are given two CNF (resp. DNF) expressions ψ_1 and ψ_2, such that (i) $\texttt{Var}(\psi_1) = \texttt{Var}(\psi_2) = V$ (say), (ii) all literals are positive in both ψ_1 and ψ_2, (iii) both ψ_1, ψ_2 are RO, (iv) every clause (resp. term) in both ψ_1 and ψ_2 has* at most *four variables, and, (v) the variables in V can be partitioned into four groups $V_1, \cdots, V_4$ such that no two variables from any group V_i co-occur in any clause (resp. term) of ψ_1 and ψ_2. The goal is to compute $\texttt{C}(\psi_1 \cdot \psi_2)$. We call this problem* #RO×RO-4Partite-4CNF *(resp.* #RO×RO-4Partite-4DNF*).*

An example of a RO×RO-4Partite-4CNF expression is $\psi = \psi_1 \cdot \psi_2$, where $\psi_1 = (x_1 + y_1 + z_2)(x_2 + y_2 + z_1 + w_1)$ and $\psi_1 = (x_2 + y_1 + z_2 + w_1)(x_1 + y_2 + z_1)$ (both ψ_1 and ψ_2 read-once, and the variables $\{x_i\}$, $\{y_i\}$, $\{z_i\}$ and $\{w_i\}$ form a partition into four groups). Note that #RO×RO-4Partite-4CNF is a special case of the #Twice-SAT problem[23] which has been proved to be $\#P$-hard in [31] without the requirements of bipartiteness of clauses in the expression $\psi_1 \cdot \psi_2$ in terms of sharing variables and 4-partiteness of variables in terms of belonging to the same clause.

Now we can give a proof sketch of Theorem 4.5, the details are deferred to Section A.2.2 in the appendix. The theorem is proved by a sequence of $\#P$-hardness proofs:

- **Step 1:** Show that counting edge covers in bipartite graphs of degree ≤ 4 where the

[23]Twice-SAT is an instance of SAT where every variable appears in at most two clauses.

edge set can be partitioned into four matchings (called #*4Partite-4BEC*) is #P-hard by a reduction from counting independent sets in 3-regular bipartite graphs.

- **Step 2:** Show that #RO×RO-4Partite-4CNF is #P-hard by a reduction from #4Partite-4BEC.
- **Step 3:** Show that #RO×RO-4Partite-4DNF is #P-hard by a reduction from #RO×RO-4Partite-4CNF.
- **Step4:** Show that $\Pr[q(I)]$ is #P-hard by a reduction from #RO×RO-4Partite-4DNF, where q satisfies the desired properties stated in Theorem 4.5.

In Step4, the queries q_1, q_2 we construct are $q_1() := R_1(x, y_1)\ R_2(x, y_2)\ R_3(x, y_3)\ R_4(x, y_4)$ and $q_2() := R_1(x_1, y)\ R_2(x_2, y)\ R_3(x_3, y)\ R_4(x_4, y)$. Clearly, q_1, q_2 do not use self-join and hence belong to CQ$^-$. Further, both of them are *hierarchical* (i.e. for every two variables x, y, the sets of subgoals that contain x, y are either disjoint or one is contained in the other) and therefore are safe for the class CQ$^-$ [56].

4.4 Approximating Tuple Probabilities

Since the exact computation of the probabilities of the result tuples of a simple class of SPJUD queries has shown to be #P-hard in Section 4.3, now we focus on the question whether we can efficiently approximate the probabilities to a desired accuracy level. In other words, we attempt to get a *fully polynomial randomized approximation scheme* (or *FPRAS*). An FPRAS is a randomized algorithm that runs in time polynomial in the input size and $1/\epsilon$ (ϵ is the *accuracy parameter*) and produces a result that is correct to within relative error $(1 \pm \epsilon)$ with high probability. In particular, for the boolean provenance ϕ_t of a tuple t in the answer $q(I)$ for an SPJUD query q and database instance I, the approximation algorithm should run in time polynomial in $n = |I|$, and $1/\epsilon$, and output a value $\widehat{P}_t$ such that

$$\Pr[|\Pr[\phi_t] - \widehat{P}_t| \leq \epsilon \Pr[\phi_t]] \geq 3/4$$

The success probability can be boosted to $1 - \delta$ for any given *confidence parameter* δ by repeating the experiment $\log(1/\delta)$ times and then taking the median of the values.

The results in this section are summarized in Theorem 4.7 and Theorem 4.8. Theorem 4.7 gives an inapproximability result even for queries with difference rank 1.

Theorem 4.7. *There exists a fixed size SPJUD query of difference rank 1 such that the computation of probabilities of the answers does not have any FPRAS unless $P = \mathcal{N}P$.*

We prove Theorem 4.7 in Section A.2.7 in the appendix. Nevertheless, in Theorem 4.8 we show that an FPRAS can be achieved in some cases where the boolean provenance of the answers can be computed in a certain *probability friendly form* (PFF). We will define PFF and discuss its relation with the *Karp-Luby* framework to approximate the probability of a Boolean expression in Section 4.4.1 and Section 4.4.2 respectively.

Theorem 4.8. *There is an FPRAS for approximating the probabilities of the answers of any SPJUD query q on probabilistic databases I such that the boolean provenance of the tuples in $q(I)$ have PFF-s that can be computed in polynomial time in the size of I.*

Later in Section 4.4.4 we show that, indeed there is a natural class of queries of difference rank 1 that will produce boolean provenance expressions in PFF which can also be computed in poly-time.

4.4.1 Probability-Friendly Form (PFF)

Definition 4.9. *A Boolean expression ϕ on n variables is said to be in PFF if ϕ is in form*

$$\phi = \alpha_0 + \sum_{i=1}^{r} \alpha_i \gamma_i$$

where $\alpha_0, \alpha_1, \cdots, \alpha_r$ are DNFs and $\gamma_1, \cdots, \gamma_r$ are d-DNNFs[24], all of polynomial size in n, and r is polynomial in n.

Note that we are allowed to have negative literals in α_i-s and γ_j-s in the above definition.

4.4.2 PFF and General Karp-Luby Framework

Consider a PFF ϕ of the form $\phi = \alpha_0 + \sum_{i=1}^{r} \alpha_i \gamma_i$, Where r is polynomial in $n = |\texttt{Var}(\phi)|$, all α_i-s are represented in DNF, and γ_i-s are represented in d-DNNF, all having size

[24]To be specific, each γ_i is represented by d-DNNF DAGs.

polynomial in n. By expanding the minterms in the α_i-s, ϕ can be expressed in the form

$$\phi = A_1 + A_2 + \cdots + A_s + B_1 \cdot \gamma_1 + \cdots + B_q \cdot \gamma_q \tag{4.1}$$

where A_j-s and B_j-s are minterms (conjunction of positive and negative literals) from the DNFs α_i-s and some of the γ_i-s may be the same. Moreover, s, q will be polynomial in n. The equivalent DNF of ϕ can be of exponential size in n. However, we show that the *general Karp-Luby framework* to estimate the size of union of sets [104, 131] can still be used to obtain an FPRAS for $\Pr[\phi]$.

Given $\Pr[x]$ for all (independent) variables in $\texttt{Var}(\phi)$, the general Karp-Luby framework works for estimating $\Pr[\phi]$ where $\phi = \phi_1 + \phi_2 + \cdots + \phi_m$, and ϕ_i-s are Boolean expressions *(not necessarily DNF minterms)* satisfying the following three properties:

(Q1) For each ϕ_i, $\Pr[\phi_i]$ can be computed in poly-time,

(Q2) For each ϕ_i, a random satisfying assignment σ (of variables in $\texttt{Var}(\phi)$) of ϕ_i can be generated in poly-time (i.e. σ is sampled with probability $\Pr[\sigma|\phi] = \frac{\Pr[\sigma]}{\Pr[\phi]}$).

(Q3) For each assignment σ and each ϕ_i, it can be decided in poly-time whether σ satisfies ϕ_i.

The framework is presented in Algorithm 5 for the sake of completeness. It is well-known that $\mathbb{E}[\frac{C}{M} \cdot \sum_j \Pr[\phi_j]] = \Pr[\phi]$, and a set of samples of size $O(\frac{m}{\epsilon^2} \log(\frac{1}{\delta}))$ suffices to estimate $\Pr[\phi]$ within accuracy $(1 \pm \epsilon)$ with probability $\geq 1 - \delta$ [104, 131].

4.4.3 FPRAS to Approximate Tuple Probabilities

Now we prove Theorem 4.8. Theorem 4.8 states that if the boolean provenance of the answers of the SPJUD query can be expressible in PFF, then the probability computation of the answers has an FPRAS. Following the discussion in Section 4.4.2, it suffices to show that all (Q1), (Q2), (Q3) hold when $\phi = \phi_1 + \cdots + \phi_m$ is in PFF. The property (Q3) trivially holds for all assignments σ and for all ϕ_i. Again, if ϕ_i in (4.1) is one of $A_1, \cdots, A_s$ (conjunction of literals), then properties (Q1) and (Q2) easily hold. Therefore we focus on proving (Q1) and (Q2), when ϕ_i is of the form $B_i\gamma_i$, where B_i is conjunction of literals and γ_i is in d-DNNF.

Algorithm 5 *Karp-Luby algorithm*

Input: A Boolean expression $\phi = \phi_1 + \cdots + \phi_m$ where properties (Q1), (Q2), (Q3) hold for each ϕ_i (ϕ_i are not necessarily DNF minterms), accuracy parameter ϵ, confidence parameter δ

Output: An estimation of $\Pr[\phi]$.

```
Initialize $C = 0$.
for $t = 1$ to $M$ do do {/* $M$ = number of samples */}
  – Sample $\phi_i$ w.p. $\frac{\Pr[\phi_i]}{\sum_j \Pr[\phi_j]}$.
  – Sample a random satisfying assignment $\sigma$ of $\phi_i$.
  if $\sigma$ does not satisfy any of $\phi_1, \phi_2, \cdots, \phi_{i-1}$ then
    – $C = C + 1$.
  end if
end for
– Output $\frac{C}{M} \cdot \sum_j \Pr[\phi_j]$.
```

Restricted d-DNNFs for $\phi_i = B_i \cdot \gamma_i$: It is easy to see that if a Boolean expression f is represented in a poly-size d-DNNF, then any partial assignments of the variables in f also has a poly-size d-DNNF, which can be computed in poly-time by replacing some of the variables by their unique assignments and reducing the d-DNNF with repeated use of $\textsc{True} + f' = \textsc{True}$, $\textsc{False} + f' = f'$, etc. Consider the Boolean expression $B_i \cdot \gamma_i$. If $B_i \cdot \gamma_i$ is true, then B_i must be true, which forces a unique assignment of the variables in B_i. The d-DNNF for the expression γ_i with this partial assignment of the variables in $\texttt{Var}(B_i) \cap \texttt{Var}(\gamma_i)$ can be computed in poly-time. Let us call this d-DNNF as $\mathcal{D}_i$ (on the variable set $\texttt{Var}(\gamma_i) \setminus \texttt{Var}(B_i)$).

(Q1): Computation of $\Pr[B_i \cdot \gamma_i]$. The value of $\Pr[B_i \cdot \gamma_i]$ can be computed from its d-DNNF $\mathcal{D}_i$ in time linear in the size of $\mathcal{D}_i$ as discussed in Section 4.2.2, and multiplying this probability with the probability of the unique satisfying assignments of the variables in $\texttt{Var}(B_i)$.

(Q2): Uniform Sampling from d-DNNF. Uniform sampling of a satisfying assignment of $\phi_i = B_i \cdot \gamma_i$, will be done by uniformly sampling a satisfying assignment σ of $\texttt{Var}(\gamma_i) \setminus$

$\texttt{Var}(B_i)$ using d-DNNF $\mathcal{D}_i$, extending σ to include the unique satisfying assignment of variables in $\texttt{Var}(B_i)$ and then further extending that assignment to an assignment σ' of $\texttt{Var}(\phi)$ by a random assignment to the variables in $\texttt{Var}(\phi) \setminus \texttt{Var}(\phi_i)$ (for every variable $x \in \texttt{Var}(\phi) \setminus \texttt{Var}(\phi_i)$, assign $x = 1$ with probability $\Pr[x]$).

For a node u in the d-DNNF $\mathcal{D}_i$, we will use ϕ_u to denote the sub-expression induced by the node u. Uniform sampling from a d-DNNF critically uses the *determinism* of the $+$-nodes: If a $+$-node u has children $u_1, \cdots, u_k$, then for every pair of j, ℓ, where $j \neq \ell$, the set of satisfying assignments for ϕ_{u_j} and that for ϕ_{u_k} are disjoint. The procedure for uniform sampling is given in Algorithm 6. It processes the nodes in the d-DNNF DAG $\mathcal{D}_i$ in reverse topological order (i.e., all the children of a node u in $\mathcal{D}_i$ is processed before u is processed). After a node u is processed by the procedure, a satisfying assignment σ_u of $\texttt{Var}(\phi_u)$ is returned. For a $\cdot$-node u, σ_u is computed by concatenating the assignments σ_{u_j}, $j = 1$ to k, where $u_1, \cdots, u_k$ are the children of u. This works because $\texttt{Var}(\phi_{u_i}) \cap \texttt{Var}(\phi_{u_j}) = \emptyset$, for every two distinct children u_i, u_j. On the other hand, for every $+$-node u, one of the children u_j is chosen at random, and σ_u is assigned to be σ_{u_j}. It is easy to check that the sampling procedure runs in time linear in the size of $\mathcal{D}_i$.

Correctness of Algorithm 6. A d-DDNNF $\mathcal{D}$ with root $\texttt{r}$ represents the expression $\phi_{\texttt{r}}$, (ϕ_u is the sub-expression at node u of $\mathcal{D}$). Here we prove that, for every node $u \in \texttt{Var}(\phi_{\texttt{r}})$, Algorithm 6 assigns a random assignment σ_u of the variables $\texttt{Var}(\phi_u)$ with probability $\frac{\Pr[\sigma_u]}{\Pr[\phi_u]}$ (which shows that at the end, a random satisfying assignment of $\texttt{Var}(\phi_{\texttt{r}})$ will be output with probability $\frac{\Pr[\sigma_u]}{\Pr[\phi_u]}$). The proof is by induction on the reverse topological order π on $\texttt{Var}(\phi_u)$. The first node u in π must be a sink node, and if u is labeled with x (resp. $\overline{x}$), the *unique* satisfying assignment σ_u will be $x = 1$ (resp. 0) which is assigned with probability 1. Assume that the induction holds up to the i-th node in order π and consider the $i+1$-th node u with children $u_1, \cdots, u_\ell$. If u is a $\cdot$-node, then by the *disjointness* of $\cdot$-nodes, $\texttt{Var}(u_j) \cap \texttt{Var}(u_\ell) = \emptyset$. Hence ϕ_{u_j} and ϕ_{u_ℓ} are independent, and $\Pr[\phi_u] = \Pi_{j=1}^{k} \Pr[\phi_{u_j}]$. Each satisfying assignment σ_u of ϕ_u must be a concatenation of satisfying assignments σ_{u_j}, $j = 1$ to k, where σ_{u_j} is the projection of the assignment σ_u on variables $\texttt{Var}(\phi_{u_j})$, and since the variables are disjoint in all ϕ_{u_j}, $\Pr[\sigma_u] = \Pi_{j=1}^{k} \Pr[\sigma_{u_j}]$. By induction hypothesis, σ_{u_j} is assigned with probability $\frac{\Pr[\sigma_{u_j}]}{\Pr[\phi_{u_j}]}$. Therefore $\frac{\Pr[\sigma_u]}{\Pr[\phi_u]} = \Pi_{j=1}^{k} \frac{\Pr[\sigma_{u_j}]}{\Pr[\phi_{u_j}]}$.

Algorithm 6 *Uniform satisfying assignment generation from a d-DNNF.*

```
Input: A d-DNNF $\mathcal{D}$ with root $\mathtt{r}$.
Output: A satisfying assignment $\sigma$ of $\mathtt{Var}(\phi_{\mathtt{r}})$ output with probability $\Pr[\sigma]/\Pr[\phi_{\mathtt{r}}]$ .
  – Compute $\Pr[\phi_u]$ for every node $u$ in $\mathcal{D}$ (see Section 4.2.2).
  – Compute a reverse topological order $\pi$ of the nodes in the d-DNNF DAG $\mathcal{D}$. Process the nodes in the order $\pi$.
  for each node $u$ in $\mathcal{D}$ do
    if $u$ is a sink-node marked with variable $x$ then
      if $\phi_u = x$, set $\sigma_u$ to be $x = 1$; else set $\sigma_u$ to be $x = 0$.
    else {/* $u$ is an internal node */}
      – Let $u_1, \cdots, u_k$ be the children of $u$.
      if $u$ is a $\cdot$-node then
        – Set $\sigma_u$ to be concatenation of $\sigma_{u_j}$, $j = 1$ to $k$.
      else {/* $u$ is a $+$-node */}
        – Choose child $u_j$ with probability $\frac{\Pr[u_j]}{\sum_{\ell=1}^{k}\Pr[u_j]}$.
        – Extend $\sigma_{u_j}$ to $\sigma'_{u_j}$ by randomly assigning the variables in $\mathtt{Var}(\phi_u) \setminus \mathtt{Var}(\phi_{u_j})$.
        – Set $\sigma_u$ to be $\sigma'_{u_j}$.
      end if
    end if
  end for
  return – $\sigma = \sigma_{\mathtt{r}}$.
```

On the other hand, if u is a $+$-node, $\Pr[\phi_u] = \sum_{j=1}^{k}\Pr[\phi_{u_j}]$ (satisfying assignments of every $\phi_{u_j}, \phi_{u_\ell}$ are disjoint). For a satisfying assignment σ_u of ϕ_u, let σ_u satisfies ϕ_{u_j} (j is unique) which is assigned with probability $\frac{\Pr[\sigma_{u_j}]}{\Pr[\phi_{u_j}]}$ (even after the extension in Step 12). Therefore $\frac{\Pr[\sigma_u]}{\Pr[\phi_u]} = \frac{\Pr[\phi_{u_j}]}{\sum_\ell \Pr[\phi_{u_\ell}]} \cdot \frac{\Pr[\sigma_{u_j}]}{\Pr[\phi_{u_j}]} = \frac{\Pr[\sigma_{u_j}]}{\sum_\ell \Pr[\phi_{u_\ell}]} = \frac{\Pr[\sigma_{u_j}]}{\Pr[\phi_u]}$. This completes the proof of Theorem 4.8.

4.4.4 Classes of Queries Producing PFF

For an SPJUD query q with difference rank $\delta(q) = 1$, we call a *difference sub-query* to be the sub-query that *immediately* appears on the right hand side of a difference operation. Since we allow union, there may be more than one difference sub-query of a query q with $\delta(q) = 1$. It is well-known that the boolean provenance ϕ_t of answers t of a safe CQ$^-$ query on tuple-independent databases are read-once [137], furthermore, if ϕ_t is read-once, $\overline{\phi_t}$ is also read-once, and there is a poly-size d-DNNF for $\overline{\phi_t}$ which can be computed in poly-time (see Proposition 4.4). Consider Proposition 4.3. From the proof of this proposition it follows that only the boolean provenance β_i produced by difference sub-queries appear as $\overline{\beta_i}$ of the boolean provenance ϕ of result tuples of an SPJUD query q with $\delta(q) = 1$. Since these β_i-s are read-once when the corresponding difference sub-query is safe for CQ$^-$, we have the following corollary:

Corollary 4.10. *Given an SPJUD query q with $\delta(q) = 1$, if all the difference sub-query-s in q are safe for the class CQ$^-$, then for any probabilistic database instance I, the boolean provenance of all answers in $q(I)$ will have a PFF that can be computed in poly-time. Therefore there is an FPRAS for this class of queries on any instance I.*

Corollary 4.10 can be extended to the instance-by-instance approach taken in [152, 156]: for the query-instance pairs (q, I) such that for every difference sub-query q' of q the boolean provenance-s in $q'(I)$ are read-once, the probability of the answers in $q(I)$ can be approximated. On the other hand, a similar result can be obtained when the difference sub-queries are UCQ queries such that for all instances I, the boolean provenance-s have a poly-size OBDD (*ordered binary decision diagrams*) representation [101]; this can also be decided from the query. Since OBDD-s are closed under negation, and a d-DNNF of poly-size in OBDD can be constructed in poly-time [58, 101], the boolean provenance of the answer tuples for all such queries on all database instances will have PFF-s computable in poly-time, and therefore again an FPRAS can be obtained using our result in the previous section.

4.5 Related Work

There has been significant progress since 2004 in probabilistic databases. We have already mentioned the *dichotomy* results that have identified exactly the safe positive queries [54–57]. Ré and Suciu show a *trichotomy* result [146] for queries in CQ^{-} extended with *aggregate* operations. Such queries can be divided into safe [145] which have efficient exact computation, *approx-safe* which have an FPRAS, and *hazardous*, which are inapproximable. This is a stronger kind of result than our lower bounds because we do not show that *every* query for which we don't give an FPRAS is inapproximable, just that some such queries exist. Approximation techniques for queries on probabilistic databases are also studied in [109, 140]. We have also mentioned work explaining the tractability of safe queries through knowledge compilation applied to their boolean provenance [101, 137] and a recent work proposing a framework for exact and approximate probability evaluation for queries with difference, but without a guarantee of polynomial running time [77].

Exploiting boolean provenance through knowledge compilation has been used for unsafe queries, in an instance-by-instance approach for the class of CQ^{-}s [152, 156]. An earlier, completely different approach compiles queries *and* databases into *probabilistic graphical models* and then performs *inference* on these [154, 155], taking advantage of the considerable theoretical and practical arsenal developed in Machine Learning. Knowledge compilation is combined with Bayesian network inference in [100]. Significant work has also been done on top-k queries on probabilistic databases [117, 144, 159]. Finally, several systems for query processing on probabilistic databases have been developed, including MistiQ [25], Trio [17, 174], and MayBMS [10].

4.6 Conclusions

We have examined the theoretical difficulty of computing exactly, and of approximating, the answers to relational queries with difference on probabilistic databases. The obvious next step is to assess the practical validity of the algorithms that we have presented as part of our FPRAS.

Desirable extensions of this work include using more flexible probabilistic models such as disjoint-independent databases and continuing to improve the sharpness of the divisions we have discovered, including dichotomy or trichotomy results as discussed before.

Chapter 5

Provenance-based Approach for Dictionary Refinement in Information Extraction

We have studied the efficient computation of the uncertainty in the output given uncertain inputs using Boolean provenance in the previous two chapters. This chapter focuses on tracing errors in the output back to the input in order to find potential erroneous inputs using Boolean provenance. Source refinement, *i.e.* removing the erroneous inputs, help improve the quality of the output. We discussed our general framework for this purpose in Section 1.1.2. In this chapter, we present our technical results in the context of *dictionary refinement in information extraction*, where the erroneous dictionaries are modeled as relations and extraction rules are modeled as relational queries.

5.1 Overview

Information Extraction, the problem of extracting structured information from unstructured text, is an essential component of many important applications including business intelligence, social media analytics, semantic search and regulatory compliance. The success of these applications is tightly connected with the quality of the extracted results, as incorrect or missing results may often render the application useless.

Most information extraction systems use a set of *rules* and a set of *dictionaries* of terms and phrases (also known as *gazetteers*) to extract *entities* (*e.g.* Person, Organization, Location) and relations between entities (*e.g.* Person's birth date or phone number) from the text. It also uses a number of basic features including *syntactic features* (*e.g.* regular expressions) and *morphological features* (*e.g.* part of speech) to identify common patterns in text. Developing and maintaining high-quality entity or relation extractors is an extremely laborious process [45]. Typically, a developer starts by collecting an initial set of features and developing an initial set of rules. She then executes the extractor on a document collection, examines the causes of incorrect results, and refines the extractor in order to remove these incorrect results; this process is repeated until the developer is satisfied with the quality of the extractor.

Dictionaries are integral to any information extraction system. For example, an extractor for Person entities would make use of dictionaries of complete names (e.g., names of famous persons), as well as dictionaries of common first names and last names, and dictionaries of common titles (*e.g.* "Mr.", "Esq."). More sophisticated Person extractors may use a larger collection of more fine-grained dictionaries to improve accuracy (*e.g.* a dictionary of common first names that are also the names of major U.S. corporations). Dictionaries are even more relevant in the context of today's informal data sources (*e.g.* social media), which do not obey the constraints of formal language (*e.g.* syntactic features such as capitalization and punctuation are rarely present, and classic part of speech analyzers trained on formal text such as TreeBank have low accuracy [122]).

In this work, we initiate a formal study of the *dictionary refinement problem*: the problem of optimizing the quality of an extractor by removing selectively chosen entries from the dictionaries used in the extractor. Although many dictionaries are readily available from public or commercial sources (*e.g.* the U.S. Census Bureau [3] provides extensive lists of first and last names, while [4] provides lists of locations containing millions of entries with detailed geographical information) inherent ambiguity in language prevents such exhaustive dictionaries from being consumed directly, as they would lead to many false positives. Instead, these dictionaries require curation before they can be used effectively in an extractor, and the extent of the curation depends on the application, domain and language. For example, it has been observed that an extensive dictionary of location names is useful, whereas an extensive dictionary of person/organization names is not that effective [102, 126]. In addition, due to inherent overlap between dictionaries for different entities ("Chelsea" is a common first name, but also the name of a famous football club; "Victoria" or "Washington" can be a location or a person), it is important to derive dictionaries with high precision tailored to the particular domain of interest. As an example, while including "Chelsea" in a dictionary of names may work well in general, when the system is customized to process a corpus of sports news articles, it is highly likely that excluding "Chelsea" from the dictionary would lead to more accurate results. Finally, dictionaries may be generated automatically [150], or collected from noisy sources [113, 134] (*e.g.*, one may frequently encounter in dictionaries of person names highly ambiguous entries such as "San", "Francisco", "Hong", "Kong", "April", "Costa"). Clearly, such dictionaries must be further refined in order to improve the quality of the system.

To refine a dictionary, the system's supervisor would wish to examine a list of entries that with high likelihood are responsible for many false positives, and therefore whose removal would lead to a significant improvement of the system. In Section 1.1.2, we mentioned the issues to be addressed to achieve this goal. There are two main challenges: (1) an extractor typically usually makes use of multiple dictionaries that are combined in arbitrary ways via complex rules. Therefore, an output is determined by multiple dictionary entries and we need to gain information about individual dictionary entries from the labels of the output. (2) *Labeling* the outputs of extractors (as true and false positives) is an expensive task requiring substantial human effort. In practice, the labeling is incomplete and many of the outputs are unlabeled which makes the refinement even more challenging.

Dictionary refinement vs. rule refinement. One may naturally ask whether the alternative technique of *rule refinement*, *i.e.*adding more rules, or refining the existing rules, could be employed to improve the quality of an extractor. While this is a natural question, the decision to employ one technique over the other is usually carefully drawn on a case by case basis, by considering not only the overall improvement in accuracy, but also the ease of maintainability of the resulting extractor. For ambiguous phrases like "Chelsea did a splendid job today!" the entities may be recognized by complex rules and advanced natural language processing techniques from the broader context. However, maintaining such rules is a labor intensive process requiring expertise in the system's rule language. Building high precision dictionaries, especially for domain-specific applications, provides a low-overhead option requiring little or no knowledge of the system. Moreover, the two techniques are not mutually exclusive. For example, when customizing a Person extractor for the sports news domain, one may decide to remove "Chelsea" from the dictionary of first names, and at the same time add "Chelsea" to a new dictionary of ambiguous names, along with a new rule that marks occurrences of ambiguous names as candidate persons only if another strong contextual (e.g., a title) clue is present in the nearby text. However, prior to adding the new rule, one must first determine the ambiguous entries in the dictionary. Therefore dictionary refinement plays an important role to create high precision or domain-specific dictionaries, to curate noisy dictionaries, as an intermediate step for rule refinement and also as an low-overhead easily maintenable alternative to rule refinement [45, 75, 118, 123].

Summary of our results. We systematically study the dictionary refinement problem when some labels in the output may be missing. To balance the requirements of extraction *precision* (minimize false positives) and *recall* (avoid discarding correct answers), we maximize the standard objective of *F-score* (the harmonic mean of precision and recall) [171]. We study the F-score maximization problem under two natural constraints that a human supervisor is likely to use: a limit on the number of dictionary entries to remove (*size constraint*), or the maximum allowable decrease in recall (*recall constraint*). Our model and the theoretical results are applicable to general relational setting where erroneous source tuples are required to be refined to reduce the number of false positives in the output.

To understand the complexity of the dictionary refinement problem, we also discuss an important special case, called *single dictionary refinement*, where a *unique* dictionary entry is responsible for producing each output entity. Besides serving as a stepping stone for the study of the more general multiple dictionary case, the single dictionary case has several practical applications including the initial refinement of a noisy dictionary generated from various sources, and the curation of specialized high-precision dictionaries (such as lists of organizations in healthcare and banking). For the general case, called *multiple dictionary refinement*, we use Boolean provenance to model the complex dependencies between dictionary entries and final results.

We divide the dictionary refinement problem into two sub-problems: (a) *Label estimation* estimates the "fractional" labels of the unlabeled outputs assuming a statistical model for the labels[25]. (b) *Refinement optimization* takes the (exact or estimated) labels of the output tuples as input. It selects a set of dictionary entries to remove that maximizes the resulting F-score under size or recall constraint.

1. For label estimation, we give a method based on the well-known *Expectation - Maximization* (EM) algorithm [67]. Our algorithm takes Boolean provenance of the labeled outputs as input and estimates labels of the unlabeled outputs. Under certain independence assumptions, we show that our application has a closed-form expression for the update rules in EM which can be efficiently evaluated.

2. For refinement optimization, we show that the problem is *NP-hard* in the general multiple dictionary case, under both size and recall constraint, even when the extractor consists of just one rule involving two dictionaries[26]. This problem remains NP-hard in the single dictionary case under recall constraint[27]. However, the optimization problem becomes poly-time solvable for the single dictionary case under size constraint.

3. We conduct a comprehensive set of experiments on a variety of real-world information extraction rule-sets and competition datasets that demonstrate the effectiveness of our techniques in the context of information extraction systems.

Organization. We have already discussed our rule languages and provenance of output tuples in terms of the dictionary entries in Section 2.1.2. Here we formally define the

[25]The labels can appear on any subset of the outputs, *e.g.* we do *not* assume that a random subset of the output is labeled.

[26]This is equivalent to the example in Section 1.1.2, $q_1(x,y) :- R(x), T(x,y), S(y)$, where the relations R and S are potential sources of errors and T is trusted.

[27]This is equivalent to the example in Section 1.1.2, $q_2(x) :- R(x), T(x)$, where only the relation R is a potential source of errors and T is trusted.

dictionary refinement problem in Section 5.2. Estimation of labels for unlabeled results is discussed in Section 5.3. The optimization algorithms for single and multiple dictionary cases are presented in Section 5.4. Section 5.5 gives our experimental results. Finally, we discuss related work (Section 5.6) and conclude with directions for future work (Section 5.7).

5.2 Dictionary Refinement Problem

Let E be an extractor (for instance, Person or Organization extractor used in an information extraction system) and let A be the set of all dictionary entries used by E. Given a set of documents $\mathbb{D}$, the extractor E produces a set of results, some of which are true positives (`Good`) and some are false positives (`Bad`). An expected mention that is not identified by the extractor is called a *missing result* (false negative). The *precision*, or accuracy of the extractor is defined as the fraction of true positives among the total number of extracted results. The *recall*, or coverage of the extractor is defined as the fraction of true positives among the total number of expected results. An extractor with high recall misses very few expected results. Finally, the standard *F-score*, also known as *F_1-score* or *F-measure* [171], combines precision and recall into a single measure computed as the harmonic mean of precision and recall ($2PR/(P+R)$). We will use F-score as the quality measure of an extractor that has been extensively used in the literature (e.g. see [102]).

When a set of entries $S \subseteq A$ is removed from A, it results in another extractor E', which on the same set documents $\mathbb{D}$ will produce a subset of the earlier results. Let $\bar{S} = A \setminus S$. Then precision of E' is

$$P_{\bar{S}} = \frac{\text{No. of } \texttt{Good} \text{ results using } \bar{S}}{\text{No. of } \texttt{Good} \text{ results using } \bar{S} + \text{ No. of } \texttt{Bad} \text{ results using } \bar{S}} \tag{5.1}$$

whereas the recall of E' is

$$R_{\bar{S}} = \frac{\text{No. of } \texttt{Good} \text{ results using } \bar{S}}{\text{No. of } \texttt{Good} \text{ results using } A} \tag{5.2}$$

It is not hard to see that the recall of E' will be at most that of E (i.e. 1), whereas the precision and therefore the F-score of E' can be more or less than that of E depending on the set S. For instance, in the output Person table of Figure 2.4, all results except

"April Smith" are `Bad`, therefore the initial precision, recall and F-scores are $\frac{1}{4}$,1 and $\frac{2}{5}$ respectively. If the entry w_1 = "Chelsea" is removed from the dictionary *first_names.dict*, the recall will remain the same, whereas the precision and therefore the F-score will improve to $\frac{1}{2}$ and $\frac{2}{3}$ respectively.

The goal of the *dictionary refinement problem* is to compute a subset S whose removal results in an extractor E' having the maximum value of the F-score on $\mathbb{D}$. To understand the difficulty of the problem, we consider two cases varying the complexity of the extractor: (i) **single dictionary case** (where the extractor consists of a single extraction rule like the rule R_1 in Figure 2.4), and, (ii) **multiple dictionary case** (extractor can consist of more than one dictionaries and arbitrary complex rules)[28]. The single dictionary case also has its independent applications as mentioned in the introduction,

5.2.1 Estimating Labels of Results for Sparse Labeling

Even computing the F-score of an extractor requires knowing labels (`Good` or `Bad`) on the entire set of extraction results of E, while in practice often only a small fraction of the results is labeled. One possible approach is to ignore the unlabeled results altogether and try to maximize the F-score only using the labeled results. But this may lead to over-fitting and the solution may not work well for the entire result set. Therefore, for incomplete labeling, we do a pre-processing step of *label estimation* for the unlabeled results assuming a *statistical model* of the labels.

But why do we need a statistical model to estimate the labels? Consider the simple single dictionary case. Suppose we have observed that a dictionary entry w has produced 50 results whereas only 4 `Good` and 1 `Bad` results appear in the labeled dataset. Then we can assume that 80% of the result set is `Good`, or, for each unlabeled result, the label is 0.8 (note that each `Good` result counts 1 and each `Bad` result counts 0 in the calculation of the F-score). This we refer to as *entry-precision* p_w of an entry w (in the example, p_w = 0.8), which measures the correctness of entry w with respect to the given extractor, i.e

[28]The cases of "single dictionary" and "multiple dictionary" basically imply that the Boolean provenance of the results contain a single variable and (potentially) multiple variables respectively. The simple extraction rule in the single dictionary case can match entries from more than one dictionary, whereas the multiple dictionary case can combine more than one entries from a single dictionary.

the *probability* of a match of w being correct when w is used in producing a result[29]. If all results produced by an entry are unlabeled, we can assign a prior value to p_w (*e.g.*, 0.5).

However, in the multiple dictionary case, a dictionary entry can produce multiple results, and a result can be produced by multiple dictionary entries combined by an arbitrary Boolean provenance (*e.g.*, in Figure 2.4, $\texttt{Prov}(t_{10}) = w_5 + w_3 \cdot w_4$). For many equivalent results (*e.g.*, different occurrences of the result 'Victoria Island'), very few or zero labels will be available in practice. Assuming a fixed prior value of fractional label may no longer be realistic. For instance, if a result depends on the correctness of many dictionary entries (with provenance like $w_1 \cdot w_2 \cdot \cdots \cdot w_k$) it has a low likelihood of being a `Good` result. On the other hand, if there are more than one equivalent ways to produce a result (with provenance like $w_1 + w_2 + \cdots + w_k$), it has a higher likelihood of being a `Good` result. Further, the labels of different results (with same or different Boolean provenance) an entry produces must be taken into account while estimating the label of an unlabeled result produced by the entry.

We formalize the above intuition assuming a statistical model on the entry-precisions, in the presence of arbitrary Boolean provenance. Under certain independence assumptions, in Section 5.3, we give a method for estimating entry-precisions (and in turn the missing labels) based on the *Expectation - Maximization (EM)* algorithm. We also show that our estimated labels using EM reduces to empirical estimates for the case of single dictionary.

5.2.2 Formal Problem Definition

We are given b dictionaries $A_1, \ldots, A_b$ and let n denote the total number of entries in $A = \cup_{\ell=1}^{b} A_\ell$. Any occurrence τ is produced by matches of one or more dictionary entries combined by the rules in the extractor; all such dictionary entries w are said to be in *provenance* of τ. How the entries produce τ is captured by the provenance expression $\texttt{Prov}(\tau)$ of τ (see Figure 2.4 in Section 2.1.2) for all such entries w we say that $w \in \texttt{Prov}(\tau)$. $\texttt{Prov}(\tau)$ is a Boolean expression where the entries in $\texttt{Prov}(\tau)$ are treated as variables (every entry in A corresponds to a unique Boolean variable).

[29]This is *not* the precision of the extractor which combines the labels and the number of results produced by all dictionary entries.

In the single-dictionary case, every occurrence τ of an entry w had $\texttt{Prov}(\tau) = w$, and when w is deleted only those occurrences get deleted. However, in the multiple-dictionary case, if an entry w is deleted, some of the results τ such that $w \in \texttt{Prov}(\tau)$ can disappear, while some such results may survive. For instance, in Figure 2.4, when w_3 = "april" is deleted, the result t_8 = "April" gets deleted, but t_{10} = "April Smith" does not get deleted (although $w_3 \in \texttt{Prov}(t_{10}) = w_5 + w_3 \cdot w_4$). This example illustrates the following observation:

Observation 5.1. When a subset of entries $S \subseteq A$ is removed, a result τ disappears from the result set if and only if its provenance expression $\texttt{Prov}(\tau)$ evaluates to False by an assignment of False (resp. True) value to the variables corresponding to the entries in S (resp. $A \setminus S$).

Let $\texttt{surv}(S)$ denote the set of results τ that survive after a given set S is deleted. For example, given three results τ_1, τ_2, τ_3 with provenance expressions $uv, u + v, uw + uv$ respectively, when $S = \{u\}$ is deleted, the set $\texttt{surv}(S)$ will only contain τ_2. Let $\phi(\tau)$ denote the Boolean label for a result τ. When the entire result set is labeled, $\phi(\tau) = 1$ if τ is `Good`, and, = 0 if τ is `Bad`. Then rewriting the expressions for precision and recall from (5.1) and (5.2), when a subset of entries $S \subseteq A$ is deleted, the "residual" precision ($P_{\bar{S}}$), recall ($R_{\bar{S}}$) and their harmonic mean F-score ($F_{\bar{S}}$) respectively are

$$P_{\bar{S}} = \frac{\sum_{\tau \in \texttt{surv}(S)} \phi(\tau)}{|\texttt{surv}(S)|}, \qquad R_{\bar{S}} = \frac{\sum_{\tau \in \texttt{surv}(S)} \phi(\tau)}{\sum_{\tau} \phi(\tau)}, \qquad F_{\bar{S}} = \frac{2\sum_{\tau \in \texttt{surv}(S)} \phi(\tau)}{|\texttt{surv}(S)| + \sum_{\tau} \phi(\tau)} \tag{5.3}$$

For incomplete labeling on the result set, we extend the above definitions by allowing *fractional labels* $\phi(\tau)$, which intuitively denote the likelihood of a result being `Good`, and are returned by the label estimation step.

The refinement of an extractor is done with human supervision in practice. The supervisor may prefer to examine a small number of dictionary entries at a time, or, may want to ensure that not too many original true positives are removed by the refinement. Therefore, we maximize the residual F-score $F_{\bar{S}}$ under two constraints:

1. **Size constraint:** Given an integer $k \leq n$, find a subset S, that maximizes $F_{\bar{S}}$, where $|S| \leq k$.

2. **Recall constraint:** Given a fraction $\rho \leq 1$, find a subset S, that maximizes $F_{\bar{S}}$, where the residual recall $R_{\bar{S}} \geq \rho$.

We will also briefly discuss the optimization problem when no size or budget constraint is given. It can be observed that both numerator and the denominator of $F_{\bar{S}}$ are dependent on the set S being removed, which makes the optimization and analysis non-trivial.

In Section 5.4, we thoroughly study the complexity of the optimization problem. We show that the optimization problem is non-trivial even in the single dictionary case. Naturally, the optimization problem becomes harder for the multiple dictionary case.

5.3 Estimating Labels

In this section we discuss our statistical data model and algorithms to estimate the labels of the unlabeled results.

5.3.1 The Statistical Model

A natural approach to estimate labels of the results would be to estimate them empirically by grouping together the results with equal provenance (equivalent Boolean expressions). The problem with this approach is that the possible number of such provenance expressions is very large, even for a very simple rule like R_4 in Figure 2.4 where such an expression would involve only two variables, and it is likely that very few (if any) labels will be available for most of them. At the same time it is quite likely that the individual dictionary entries have similar entry-precisions, *i.e.*, the likelihood of being a correct match for the results, across results with different provenance expressions. Following the example given in the introduction, the candidate last name "'Island" has a low entry-precision. It can produce more than one `Bad` results like "Victoria Island" and "Baffin Island", and "Island" is a `Bad` match for a last name in both results. We represent this intuition by defining the model in the following way.

We assume that each entry w has a fixed (and unknown) entry-precision p_w. For any given result τ such that $w \in \texttt{Prov}(\tau)$, the *match*[30] of w for τ is correct with probability p_w and incorrect with probability $1 - p_w$ independent of the other results and other entries in $\texttt{Prov}(\tau)$. Further, we assume that the rules in the extractor are correct, *i.e.*, the label of τ is `Good` if and only if its provenance $\texttt{Prov}(\tau)$ evaluates to 1 with the matches of the dictionary entries in $\texttt{Prov}(\tau)$ ($\texttt{Good} \equiv 1$ and $\texttt{Bad} \equiv 0$). Next we discuss how we estimate the labels from the estimated entry-precisions, followed by our EM-based algorithm to estimate the entry-precisions from the available labeled data.

5.3.2 Estimating Labels from Entry-Precisions

Under our statistical model, the label $\phi(\tau)$ of a result can be estimated by evaluating the probability of its provenance expression $\texttt{Prov}(\tau)$ given the entry-precisions p_w for all entry $w \in \texttt{Prov}(\tau)$. Computing the probability of any Boolean expression ϕ given the probabilities of its constituent variables is in general *#P-hard* [169], and, the classes of queries for which the probability of the Boolean provenance can be efficiently computed have been extensively studied in the literature (*e.g.*, see [54]). However, our Boolean provenance involve a small number of variables (typically ≤ 10). So we can compute $\phi(\tau)$ given p_w-s by an exhaustive enumeration under the independence assumptions.

5.3.3 Estimating Entry-Precisions by EM

Here our goal is to estimate the values of entry-precisionp_w given a set of occurrences τ along with their labels and provenance expressions $\texttt{Prov}(\tau)$. We use the *Expectation - Maximization (EM)* algorithm to solve this problem. The EM algorithm [67] is a widely-used technique for the maximum likelihood estimation of parameters of a probabilistic model with hidden variables. This algorithm estimates the parameters iteratively either for a given number of steps or until some convergence criteria are met.

First, we introduce some notation to present the update rules of EM in terms of our problem. We index the entries arbitrarily as $w_1, \cdots, w_n$. Each entry w_i has an entry-precision $p_i = p_{w_i}$. There are N labeled occurrences $\tau_1, \cdots, \tau_N$. We assume that $\tau_1, \cdots, \tau_N$ also denote the labels of the occurrences $\phi(\tau_1), \cdots, \phi(\tau_N)$, so each τ_i is Boolean, where

[30]A match is the tuple in the intermediate relation extracted by the Dictionary operator.

$\tau_i = 1$ (resp. 0) if the label is `Good` (resp. `Bad`). If $w_i \in \texttt{Prov}(\tau_j)$, we say that $\tau_j \in \texttt{Succ}(w_i)$. For simplicity, we assume that entries from exactly b dictionaries are involved in the provenance expression $\phi_j = \texttt{Prov}(\tau_j)$ for each occurrence τ_j, although our implementation works for general cases. Hence each ϕ_j takes b inputs $y_{j1}, \cdots, y_{jb}$ and produces τ_j. Each $y_{j\ell}$ is Boolean, where $y_{j\ell} = 1$ (resp. 0) if the match of dictionary entry corresponding to $y_{j\ell}$ is correct (resp. incorrect) while producing the label for τ_j. The entry corresponding to $y_{j\ell}$ will be denoted by $\texttt{Prov}_{j\ell} \in \{w_1, \cdots, w_n\}$.

To illustrate the notation, let us consider the rule R_4 in Figure 2.4: the result is a person name if it is a match from the firstname dictionary, followed by a match from lastname dictionary. We refer to this rule as the *firstname-lastname* rule. In this example, $b = 2$ and for every occurrence τ_j, $\tau_j = \phi_j(y_{j1}, y_{j2}) = y_{j1}y_{j2}$. For a `Good` occurrence "April Smith", $\tau_j = 1$, $y_{j1} = 1$ (for "April"), and $y_{j2} = 1$ (for "Smith"), $\texttt{Prov}_{j1}$ = "April" and $\texttt{Prov}_{j2}$ = "Smith". For a `Bad` occurrence "Victoria Island", $\tau_j = 0$, $y_{j1} = 1$ (for "Victoria"), and $y_{j2} = 0$ (for "Island").

Parameters and hidden variables for EM. For our problem, the vector of labels of the occurrences $\vec{x} = \langle \tau_1, \cdots, \tau_N \rangle$ are the *observed variables*, the vector of vectors for the correctness of matches of individual entries for these occurrences $\vec{\vec{y}} = \langle y_{j\ell} \rangle_{j \in [1,N], \ell \in [1,b]}$ are the *hidden variables*, and the vector of entry-precisions $\vec{\theta} = \{p_1, \cdots, p_n\}$ is the vector of unknown *parameters*.

Update rules for EM. Let $\vec{\theta^t}$ be the parameter vector at iteration t. The log-likelihood of the observed variables is $q(\vec{x}; \vec{\theta}) = \log P(\vec{x}|\vec{\theta}) = \sum_{j=1}^{N} P(\tau_i|\vec{\theta})$. The complete information for the problem includes the observed variables $\vec{x} = \langle \tau_1, \cdots, \tau_N \rangle$ as well as the hidden variables $\vec{\vec{y}} = \langle y_{j\ell} \rangle_{j \in [1,N], \ell \in [1,b]}$. The expected log-likelihood of the complete information given the observed variables $\vec{x}$ and current parameter vector $\vec{\theta^t}$ is $\mathbb{E}[q(\vec{x}, \vec{\vec{y}}; \vec{\theta})|\vec{x}, \vec{\theta^t}] = K$ (say). Under the independence assumption K can be shown to be equal to:

$$K = \sum_{i \in [1,n]} \sum_{\substack{j \in [1,N], \tau_j \in \texttt{Succ}(w_i), \\ \texttt{Prov}(y_{j\ell}) = w_i}} c_{w_i,\tau_j,t} \log p_i + (1 - c_{w_i,\tau_j,t}) \log(1 - p_i)$$

where $c_{w_i,\tau_j,t} = \mathbb{E}[y_{j\ell}|\tau_j, \vec{\theta^t}]$, $\tau_j \in \texttt{Succ}(w_i)$ and $\texttt{Prov}_{j\ell} = w_i$. In the **E-step**, for every word w_i, and for every occurrence $\tau_j \in \texttt{Succ}(w_i)$, we compute $c_{w_i,\tau_j,t}$, the expectation of $y_{j\ell}$ (the ℓ-th bit of $\vec{y}_j$ where $\texttt{Prov}(y_{j\ell}) = w_i$) given the current parameter vector $\vec{\theta^t}$ using the probabilities $\Pr[\vec{y_j}|\tau_j, \vec{\theta^t}]$. Hence, after the E-step, for every occurrence τ_j we have a vector of real numbers of size b (if ϕ_j takes b inputs). In the **M-step**, we maximize the expression K w.r.t. parameter vector $\vec{\theta}$ to get the next guess of the parameters θ^{t+1}. Differentiating K w.r.t. each p_i, and equating $\delta K/\delta p_i$ to zero gives a closed-form expression for $p_i = \frac{C_1}{C_1+C_2}$, $i \in [1, n]$. Here $C_1 = \sum c_{w_i,\tau_j,t}$, $C_2 = \sum(1 - c_{w_i,\tau_j,t})$, and the sums are over $1 \le j \le N$ such that $\tau_j \in$ `Succ`(w_i). These parameter values are evaluated using the values of $c_{w_i,\tau_j,t}$ computed before and used as the estimation of the parameters in the $t+1$-th round. $\vec{\theta^{t+1}}$. The complete derivation of the update rules is given in the appendix (Section A.3.2).

Simplified update rules for the single dictionary case. Here we prove an interesting observation, which will be helpful in the optimization step.

Observation 5.2. For the case of single dictionary, the estimated entry-precision using EM reduces to its empirical entry-precision and EM converges in a single step.

For single dictionary, `Prov`$(\tau) = w$, where the result τ is an occurrence of an entry w. (*i.e.*, $b = 1$). Fix an arbitrary result τ_j, $j \in [1, N]$, and *the unique* entry w_i such that `Prov`$_{j,1} = w_i$. Then, at any time step t and for any values of the parameters $\vec{\theta^t}$, $c_{w_i,\tau_j,t} = \mathbb{E}[y_{j1}|\tau_j, \vec{\theta^t}] = \tau_j$. In other words, when the label of a result is given, whether or not the corresponding entry is a correct match for this result can be exactly inferred from the label of the result. Hence, the entry-precision p_i for any entry w_i is $p_i = \frac{C_1}{C_1+C_2} = \frac{\sum c_{w_i,\tau_j,t}}{\sum c_{w_i,\tau_j,t} + \sum(1-c_{w_i,\tau_j,t})}$ (the sums are over all τ_j such that $\tau_j \in$ `Succ`(w_i)) $= \frac{\sum \tau_j}{\sum \tau_j + \sum(1-\tau_j)}$ which equals the fraction of results produces by w_i that are `Good`, *i.e.*, the empirical entry-precision of w_i. Further, the EM algorithm will converge in a single step since this estimate is independent of the time-step t, and therefore this estimation step can be omitted altogether.

5.4 Refinement Optimization

Next we discuss the optimization problem of maximizing the residual F-score when (possibly fractional) labels of all occurrences in the result set are available for single and multiple dictionary cases (resp. Section 5.4.1 and 5.4.2).

5.4.1 Refinement Optimization for Single Dictionary

The following observation simplifies the expressions of precision, recall, and F-score for the single dictionary case.

Observation 5.3. A result $\tau \in \texttt{surv}(S)$ for single dictionary, if and only if the entry $w \notin S$ where $w = \texttt{Prov}(\tau)$.

Let us denote the *frequency* of an entry w (*i.e.*, the number of results τ such that $\texttt{Prov}(\tau) = w$) by f_w. Then for any entry $w \in A$ $|\texttt{surv}(S)| = \sum_{w \notin S} f_w$, and using Observation 5.2, $\sum_{\tau \in \texttt{surv}(S)} \phi(\tau) = \sum_{w \notin S} \sum_{\tau : w = \texttt{Prov}(\tau)} \phi(\tau) = \sum_{w \notin S} p_w f_w$. Here p_w is the (estimated) entry-precision of w[31]. Then the expressions for $P_{\bar{S}}, R_{\bar{S}}, F_{\bar{S}}$ given in (5.3) reduce to:

$$P_{\bar{S}} = \frac{\sum_{w \notin S} p_w f_w}{\sum_{w \notin S} f_w}, \qquad R_{\bar{S}} = \frac{\sum_{w \notin S} p_w f_w}{\sum_{w \in A} p_w f_w}, \qquad F_{\bar{S}} = 2\frac{\sum_{w \notin S} p_w f_w}{\sum_{w \in A} p_w f_w + \sum_{w \notin S} f_w}$$

In Section 5.4.1.1 we give an optimal poly-time algorithm to maximize the residual F-score under size constraint. For the recall constraint, in Section 5.4.1.2, we show that the exact optimization is NP-hard. Despite the hardness result, we give a simple poly-time algorithm that is provably nearly optimal and works well in our tests.

5.4.1.1 Size Constraint

Our goal is to maximize $F_{\bar{S}} = 2\frac{\sum_{w \notin S} p_w f_w}{\sum_{w \in A} p_w f_w + \sum_{w \notin S} f_w}$, where $|S| \le k$. The main idea of our algorithm is based on the fact that finding out whether there exists a dictionary with F-score of at least θ is a significantly simpler problem which overcomes the non-linearity of the objective function [74]. Accordingly, our algorithm *guesses* a value θ and then checks if θ is a feasible F-score for some S. The value of θ that maximizes $F_{\bar{S}}$ is then found by a binary search.

Checking if θ is a feasible F-score. We need to check whether there is a set S of entries such that $F_{\bar{S}} = 2\frac{\sum_{w \in A} p_w f_w - \sum_{w \in S} p_w f_w}{\sum_{w \in A} p_w f_w + \sum_{w \in A} f_w - \sum_{w \in S} f_w} \ge \theta$. Rearranging the terms, we need to find out whether there exists S such that

$$\sum_{w \in S} f_w(\theta - 2p_w) \ge \sum_{w \in A} f_w(\theta - (2-\theta)p_w)$$

[31]For unlabeled results τ-s such that $\texttt{Prov}(\tau) = w$, estimated label $\phi(\tau) = p_w$. For the labeled such results τ-s, the fraction having label 1 (`Good`) is p_w, the rest is 0 (`Bad`).

Note that the right hand side of the inequality is independent of S, so it suffices to select the highest (at most) k entries with non-negative value of $f_w(\theta - 2p_w)$ and check if the sum is at least $\sum_{w \in A} f_w(\theta - (2-\theta)p_w)$.

Clearly we want a subset S such that $F_{\bar{S}} \geq F_A$. Hence the guess θ is varied between F_A and 1. We present the algorithm in Algorithm 7 in terms of an accuracy parameter Δ; the value of Δ for the optimal F-score will be discussed later.

Algorithm 7 Algorithm for size constraint (given k and Δ)

```
1: – Let $\theta_{low} = F_A$ and $\theta_{high} = 1$
2: while $\theta_{high} - \theta_{low} > \Delta$ do
3:     Let $\theta = (\theta_{high} + \theta_{low})/2$ be the current guess.
4:     Sort the entries $w$ in descending order of $f_w(\theta - 2p_w)$.
5:     Let $S$ be the top $\ell \leq k$ entries in the sorted order such that $f_w(\theta - 2p_w) \geq 0$ for all
       $w \in S$.
6:     if $\sum_{w \in S}[f_w(\theta - 2p_w)] \geq \sum_{w \in A} f_w(\theta - (2-\theta)p_w)$ then
7:         $\theta$ is feasible, set $\theta_{low} = F_{\bar{S}}$ and continue.
8:     else
9:         $\theta$ is not feasible, set $\theta_{high} = \theta$ and continue.
10:    end if
11: end while
12: Output the set $S$ used to define the most recent $\theta_{low}$.
```

Running time. There is a linear time $O(n)$ time algorithm for the feasibility step 6: (i) Use the standard linear time selection algorithm to find the k-th highest entry, say u, according to $f_w(\theta - 2p_w)$, (ii) do a linear scan to choose the entries w such that $f_w(\theta - 2p_w) > f_u(\theta - 2p_u)$, and then choose entries such that $f_w(\theta - 2p_w) = f_u(\theta - 2p_u)$ to get k entries in total, (iii) discard the selected entries with negative values of $f_w(\theta - 2p_w)$ and output the remaining $\leq k$ entries as the set S. However, we can have simpler implementations of the verification step - using a min-heap gives $O(n + k \log n)$ time, whereas a simple sorting gives $O(n \log n)$ time. Since values of the guesses are between 0 and 1, and the algorithm stops when the upper and lower bounds are less than Δ away, at most $\log(1/\Delta)$ steps will be required. This means that there is an implementation of the algorithm with running time of $O(n \log(1/\Delta))$.

Value of Δ for optimal F-score. Let B be the number of bits to represent each p_w, f_w in the input. Consider any F-score $F_{\overline{S}} = 2\frac{\sum_{w \notin S} p_w f_w}{\sum_{w \in A} p_w f_w + \sum_{w \notin S} f_w}$. B-bit numbers between 0 and 1 can represent values $t.2^{-B}$, for $0 \leq t \leq 2^B - 1$. Multiply both numerator and denominator of $F_{\overline{S}}$ by 2^{2B} to get integer values in the numerator and denominator. Each of $p_w f_w$ and f_w in the denominator is at most 2^{2B} after this multiplication, and there are at most $2n$ of them. The denominator of the fraction representing difference between two unequal F-score values is at most $n2^{2B+1}$, whereas the numerator of the fraction is at least 1. Hence the difference is at least $\frac{1}{(n2^{2B+1})^2}$, and setting $\Delta = \frac{1}{(n2^{2B+1})^2}$ suffices. This leads to the following theorem:

Theorem 5.4. *There is an optimal algorithm for maximizing the residual F-score for single dictionary refinement under size constraint. The algorithm runs in time $O(n \cdot (\log n + B))$ where B is the number of bits used to represent each of the p_w and f_w values given to the algorithm.*

Greedy algorithm is not optimal. A natural question that may arise is whether selecting entries greedily (select the next entry that gives the maximum improvement in F-score and repeat for k steps) also gives an optimal solution. The following example answers this question in the negative.

Example 5.5. Let $n = 4$ and $k = 2$, $A = \{w_1, w_2, w_3, w_4\}$. The pair of precision and frequency (p_{w_i}, f_{w_i}), $1 \leq i \leq 4$ of these entries respectively are (0.0284,0.2374), (0.0050, 0.2846), (0.0040, 0.2485), (0.0033, 0.2295). Then the original F-score $F_A = 19.64 \times 10^{-3}$. Removing entries w_i, $1 \leq i \leq 4$ gives residual F-score 8.22×10^{-3}, 23.42×10^{-3}, 23.44×10^{-3}, and, 23.47×10^{-3} respectively. Hence greedy will choose entry w_4 in the first step. Given that w_4 is already chosen, removing w_1, w_2 and w_3 in addition gives resp. F-scores 8.90×10^{-3}, 31.21×10^{-3}, 30.70×10^{-3}. Hence the output of greedy is $\{w_4, w_2\}$. But the F-score after removing $\{w_2, w_3\}$ is 31.46×10^{-3}, which is better than the solution of greedy, and in this case also is the optimal solution. □

It can be verified that this example also shows non-optimality of other obvious choices like choosing entries in *increasing order of precision* p_w (outputs w_4, w_3) or in decreasing order of *(relative) bad counts* $(1 - p_w) f_w$ (outputs w_4, w_1). This example and our experiments show that the performance of greedy and the proposed optimal algorithms may be comparable for some data sets, though our algorithm is more efficient.

5.4.1.2 Recall Constraint

First, we show the NP-hardness of exact optimization for recall constraint.

NP-hardness of the exact optimization. We prove the NP-hardness via a reduction from the *subset-sum problem* which is known to be NP-hard [80]. In the subset-sum problem the input is a sequence of positive integers $I = \langle x_1, \cdots, x_n \rangle$[32], and an integer C, and the goal is to decide if there is a subset $S \subseteq I$ such that $\sum_{x_i \in S} x_i = C$.

Our reduction creates an instance of the refinement problem in which every number in the subset-sum instance corresponds to an entry with fixed and low precision and frequency proportional to the number. In addition, we create a single high precision word. This word ensures that the highest F-score is achieved when the total frequency of the removed low precision words is the highest. Therefore maximum F-score can be achieved only when the recall budget is used exactly. By the properties of our reduction, this corresponds to having a subset with the desired sum in the subset-sum instance. The complete reduction and its proof of correctness appears in the appendix (Section A.3.1).

Nearly Optimal Algorithm. We now describe a simple and efficient algorithm that gives a nearly optimal solution when used on a large corpus where frequencies of individual entries are small. Our algorithm sorts the entries in increasing order of precisions p_w, and selects entries according to this order until the recall budget is exhausted or there is no improvement of F-score by selecting the next entry. The pseudocode of the algorithm is given in Algorithm 8.

Clearly the algorithm runs in time $O(n \log n)$. Formally, we prove the following theorem that gives a lower bound on the F-score of the solution produced by our algorithm.

Theorem 5.6. *Let $w_1, \cdots, w_n$ be the entries sorted by precision and $p_1 \leq \cdots \leq p_n$ be the corresponding precisions. Let S^* be the set of entries whose removal gives the optimal F-score such that $R_{\bar{S^*}} \geq \rho$. Let $r^* = \sum_{i \in \bar{S^*}} p_i f_i$ and let ℓ be the largest index for which $\sum_{i > \ell} p_i f_i \geq r^*$. Then the set*

[32]The subset-sum problem is NP-hard even for positive integers.

Algorithm 8 Algorithm for recall budget constraint (given ρ)

```
1:  – Sort the entries in increasing order of precisions $p_w$, let $w_1, \cdots, w_n$ be the entries in
    sorted order and $p_1 \le \cdots \le p_n$ be the corresponding precisions.
2:  – Let $S_\ell = \{w_i : i \le \ell\}$, and $S_0 = \emptyset$.
3:  – Initialize $i = 1$.
4:  while $i \le n$ do
5:    if $F_{\overline{S}_i} \ge F_{\overline{S}_{i-1}}$ then
6:      if $R_{\overline{S}_i} \ge \rho$ then
7:        $i = i + 1$, continue.
8:      else
9:        return $S_{i-1}$.
10:     end if
11:   end if
12: end while
```

S returned by our algorithm satisfies

$$F_{\bar{S}} \ge \frac{2\sum_{i\in\bar{S}^*} p_i f_i}{\sum_{i\in\bar{S}^*} f_i + \sum_i p_i f_i + f_{max}/p_{\ell+1}}.$$

Proof. The algorithms orders the elements according to their precision values and selects in this order until the recall budget is exhausted or there is no further improvement in F-score. Let $p_i = p_{w_i}$ and $f_i = f_{w_i}$, where $p_1 \le \cdots \le p_n$. Let $f_{max} = \max\{f_1, f_2, \ldots, f_n\}$, $S^i = \{w_j : 1 \le j \le i\}$ and $\overline{S} = A \setminus S$. Let $r^\ell = \sum_{i\in\overline{S^\ell}} p_i f_i = \sum_{i>\ell} p_i f_i$. By definition, $r^* + f_{max} \ge r^\ell \ge r^*$.

Note that recall $R_{\overline{S^\ell}} = r^\ell / \sum_i p_i f_i \ge r^* / \sum_i p_i f_i = R_{\overline{S^*}} \ge \rho$. Due to the monotonicity check, the algorithm will return a solution with F-score $\ge F_{\overline{S^\ell}}$. Hence it suffices to give a lower bound on $P_{\overline{S^\ell}}$.

To do this observe that (1) $\sum_{i\in\overline{S^\ell}\setminus\overline{S^*}} p_i f_i - \sum_{i\in\overline{S^*}\setminus\overline{S^\ell}} p_i f_i = \sum_{i\in\overline{S^\ell}} p_i f_i - \sum_{i\in\overline{S^*}} p_i f_i \le f_{max}$, and, (2) $\sum_{i\in\overline{S^\ell}\setminus\overline{S^*}} f_i \le \frac{\sum_{i\in\overline{S^\ell}\setminus\overline{S^*}} p_i f_i}{p_{\ell+1}} \le \frac{(\sum_{i\in\overline{S^*}\setminus\overline{S^\ell}} p_i f_i + f_{max})}{p_{\ell+1}} \le \sum_{i\in\overline{S^*}\setminus\overline{S^\ell}} f_i + \frac{f_{max}}{p_{\ell+1}}$. From (1) and (2), $\sum_{i\in\overline{S^\ell}} f_i \le \sum_{i\in\overline{S^*}} f_i + \frac{f_{max}}{p_{\ell+1}}$. Hence $P_{\overline{S^\ell}} = \frac{r^\ell}{\sum_{i\in\overline{S^\ell}} f_i} \ge \frac{r^*}{\sum_{i\in\overline{S^*}} f_i + f_{max}/p_{\ell+1}}$, and, $F_{\overline{S^\ell}} = \frac{2}{1/R_{\overline{S^\ell}} + 1/P_{\overline{S^\ell}}} \ge \frac{2}{1/R_{\overline{S^*}} + 1/P_{\overline{S^\ell}}} \ge \frac{2\sum_{i\in\overline{S^*}} p_i f_i}{\sum_{i\in\overline{S^*}} f_i + \sum_i p_i f_i + \frac{f_{max}}{p_{\ell+1}}}$ □

Note that the lower bound guaranteed by the algorithm differs from the optimal F-score $F_{\bar{S}^*}$ only by the addition of the error term $\frac{f_{max}}{p_{\ell+1}}$ to the denominator. Individual frequencies are likely to be small when the given corpus and the dictionary are large. At the same time ℓ and hence $p_{\ell+1}$ are determined solely by the recall budget. Therefore the error term $\frac{f_{max}}{p_{\ell+1}}$ is likely to be much smaller than the denominator for a large dictionary. Our experiments confirm this informal argument.

Optimal F-Score without constraints. Another surprising property of the algorithm we just described is that while it is not necessarily optimal in general, without the recall budget (i.e. with $\rho = 0$) this algorithm finds the solution with the globally optimal F-score. Naturally, the optimal solution can also be found using the slightly more involved Algorithm 7 with $k = n$. The proof of this claim can be found in [151].

5.4.2 Refinement Optimization for Multiple Dictionaries

The optimization problem becomes harder in the multiple dictionary case. In Section 5.4.2.1 we show that even for the simple firstname-lastname rule (rule R_4 in Figure 2.4) the optimization problem for size constraint is NP-hard; this problem was shown to be poly-time solvable for single dictionary. The case of recall constraint has already been shown to be NP-hard even for single-dictionary. Then in Section 5.4.2.2 we discuss some efficient algorithms that we evaluate experimentally.

5.4.2.1 NP-hardness for Size Constraint

We give a reduction from the *k'-densest subgraph problem in bipartite graphs* which has been proved to be NP-hard in [48][33]. Here the input is a bipartite graph $H(U, V, E)$ with n' vertices and m' edges, and, an integer $k' < n'$. The goal is to select a subset of vertices $W \subseteq U \cup V$ such that $|W| = k'$ and the subgraph induced on W has the maximum number of edges. We will denote the set of edges in the *induced subgraph* on W (every edge in the subgraph has both its endpoints in W) by $E(W)$.

For simplicity, first we prove a weaker claim: removing a subset S such that the the size of S is *exactly k* (as opposed to *at most k*) is NP-hard. Intuitively, the vertices correspond to

[33]The complexity of the unconstrained multiple dictionary case $k = n$ or recall budget = 0, is an interesting open problem

entries and the edges correspond to occurrences. We show that if the induced subgraph on a subset of vertices of size at most k' has a large number of edges, then removing entries in the *complement* of this subset results in this induced subgraph that gives a large residual F-score.

Given an instance of the k'-densest subgraph problem, we create an instance of the dictionary refinement problem as follows. The vertices in U and V respectively correspond to the entries in the firstname and lastname dictionaries in the firstname-lastname rule. Every edge $(u, v) \in E$ corresponds to a unique provenance expression $\phi_{u,v} = uv$, where the entries u and v are chosen from these two dictionaries respectively. For each $(u, v) \in E$, there is one result with label 1 (`Good`), and one with label 0 (`Bad`). The parameter k in the dictionary refinement problem is $k = n' - k'$. We show that there is a subset $W \subseteq U \cup V$, such that $|W| = k'$ and $E(W) \geq q$ if and only if there is a subset S for the dictionary refinement problem such that $|S| = k$ and $F_{\bar{S}} \geq \frac{2}{\frac{m'}{q}+2}$.

The residual precision in the above reduction is a constant for all choices of S, and therefore, the residual F-score is a monotone function of the residual recall. Hence the above reduction does not work for the relaxed constraint $|S| \leq k$ (the residual recall is always maximized at $S = \emptyset$, i.e. when $k = 0$, independent of the k'-densest subgraph solution).

Outline of reduction for $|S| \leq k$. To strengthen the above reduction to work for the relaxed constraint $|S| \leq k$, we do the following We retain the graph with a good and a bad occurrence for every edge as before. In addition, we add $s = mn$ `Good` results that are unrelated to anything. These results will make the differences in the recall between solutions tiny (while preserving monotonicity in the size of $E(W)$). For every original entry $u \in U \cup V$ corresponding to the vertices in the graph, we add s `Bad` results (u, u^i) $1 \leq i \leq s$ connected to it. The entries $\bigcup_{u \in U \cup V}\{u_i : 1 \leq i \leq s\}$ are called auxiliary-bad entries. This way the precision will be roughly equal to $\frac{1}{n-k}$ (since these results dominate the total count) and hence solutions with smaller number of entries removed will have noticeably lower precision. It is also true that removing any of the auxiliary bad entries will have a tiny effect, so any optimal solution for the refinement problem will always remove the entries corresponding to graph vertices $U \cup V$. The complete reduction is given in the appendix (Section A.3.3). This proves the following theorem:

Theorem 5.7. *Maximization of the residual F-score for multiple dictionary refinement under size constraint is NP-hard even for the simple firstname-lastname rule (*i.e.*, the rule R_4 in Figure 2.4).*

5.4.2.2 Refinement Algorithms

Since multiple dictionary refinement problem is NP-hard under both size and recall constraints, we propose and evaluate two simple and efficient algorithms. These algorithms take the label (actual labels for labeled results, and estimated labels for unlabeled results) and the Boolean provenance for every result as input, and produce a subset of entries to remove across all dictionaries.

We take the *greedy approach* to compute the set of entries: select the entry in the next step that maximizes (1) ΔF for size constraint, and (2) $\Delta F / \Delta R$ for recall constraint. Here $\Delta F, \Delta R$ denote the changes in F-score and recall by deleting one additional entry. The implementation of both algorithms involves repeated computation of the $\texttt{surv}(S)$ sets after each entry is included to S. These algorithms stop if no further improvement in F-score is possible by deleting any entry or when the given size or recall budget is exhausted. These algorithms are empirically evaluated in Section 5.5.2.2.

5.5 Experiments

In this section we present examples and results for three purposes. First, in Section 5.5.1, we report various statistics on the datasets used in the experiments. This provides some intuitive understanding of the characteristic of the problem. Second, in Section 5.5.2, we evaluate the refinement algorithms on fully labeled datasets and compare their performance and efficiency; these results supplement the theoretical results derived earlier. Finally, we consider the refinement algorithms in conjunction with the label estimation step in Section 5.5.3 in order to evaluate their performance on partially labeled datasets.

Experimental Settings. We used SystemT v1.0.0 [114, 148], the information extraction system developed at IBM ResearchAlmaden, enhanced with a framework to support provenance of the results [118], for our experiments. The rules are specified in AQL (Annotation Query Language), which is the rule language of SystemT. All experiments were implemented in Java with JDK 6.0, and run on a 64-bit Windows 7 machine with Intel(R) Core^{TM} i7-2600 processor (4 GB RAM, 3.40 GHz).

5.5.1 Evaluation Datasets

We used two datasets in our experiments: CoNLL2003 (*CONLL* in short) and ACE2005 (*ACE* in short). These datasets are realistic in practical scenarios, and both have been used in official Named Entity Recognition competitions [5, 167]. Each dataset contains labeled results or occurrences, which conceptually are of the form: "(occurrence id, label)", where the *occurrence id* is an abstraction of "(document id, begin offset, end offset)", and the *label* is either `Good` or `Bad` (for true and false positives respectively). In this work splitting of a corpus into documents, and the location of the occurrences within a document play no role. The details of these datasets for single and multiple dictionary refinements are described in Figure 5.1. We considered two Person extractors to evaluate the single and multiple dictionary cases. While the single dictionary case used only one rule (like the rule R_1 in Figure 2.4) involving a single dictionary "name.dict" containing both first and last names, the multiple dictionary case used a Person extractor with 14 AQL rules and two more specific dictionaries "StrictFirst.dict" and "StrictLast.dict" containing first and last names respectively, along with a high-precision salutation dictionary (containing entries like "Mr.", "Prof.", etc.) that has already been refined and does not participate in the refinement process[34].

In Figure 5.2, we present some statistics on these two datasets in terms of single dictionary refinement. Figure 5.2 (i) shows that around 200 dictionary entries have precisions 0.05 or less in these two datasets, and motivates the importance of dictionary refinement. Although relative frequencies of most of the dictionary entries are small (≤ 0.01, as shown in Figure 5.2 (i)), Figure 5.2 (ii) shows that many entries have produced 5 or more bad occurrences[35].

[34]The *dictionary size* in the figure denotes the total number of entries across all dictionaries producing at least one occurrence.

[35]The number of entries producing 5 or more bad occurrences is 69 for CONLL and 36 for ACE, and the maximum number of bad occurrences for an entry is 101 for CONLL (produced by "september") and 453 for ACE (produced by "re").

Corpus	CONLL		ACE	
Refinement	Single	Multiple	Single	Multiple
No. of documents	945		274	
No. of AQL rules	1	14	1	14
Dictionary size	1501	1357	704	664
No. of matches	6526	7448	4616	4619
No. of true positives	4721	6301	3327	3601
No. of false positives	1805	1147	1289	1018

Figure 5.1: Details of evaluation data sets

5.5.2 Dictionary Refinement with Full Labeling

Here we evaluate the algorithms proposed in Section 5.4 for single and multiple dictionary cases, both in terms of the resulting F-score after refinement and the runtime.

5.5.2.1 Refinement Algorithms for Single Dictionary Case

For **size constraint**, we compare Algorithm 7 with three other natural approaches on both CONLL and ACE datasets. The algorithms compared are (see Figure 5.3 (i)): (1) *K-Optimal*: the optimal algorithm presented in Algorithm 7 with $\Delta = 0.001$, (2) *K-Greedy*: the next entry is selected greedily that gives the maximum increment in F-score (*i.e.*, maximizes ΔF at each step), (3) *K-BadFraction*: chooses the top-k entries in the decreasing order of *fraction* of `Bad` results (or, increasing order of individual empirical precisions), (4) *K-BadCount*: chooses the top-k entries in the decreasing order of the *number* of `Bad` results produced by the entries.

For **recall constraint**, we compare our near-optimal algorithm discussed in Section 5.4.1.2) with three other natural choices. The algorithms compared are (see Figure 5.3 (ii)): (1) *RecBudget-Near-Opt*: the near-optimal algorithm that selects entries in increasing order of individual empirical precisions, (2) *RecBudget-Greedy*: the greedy algorithm similar to K-Greedy (that maximizes ΔF at each step), (3) *RecBudget-Greedy-FR*: the greedy algorithm that maximizes the incremental improvement in F-score relative to the decrease in recall

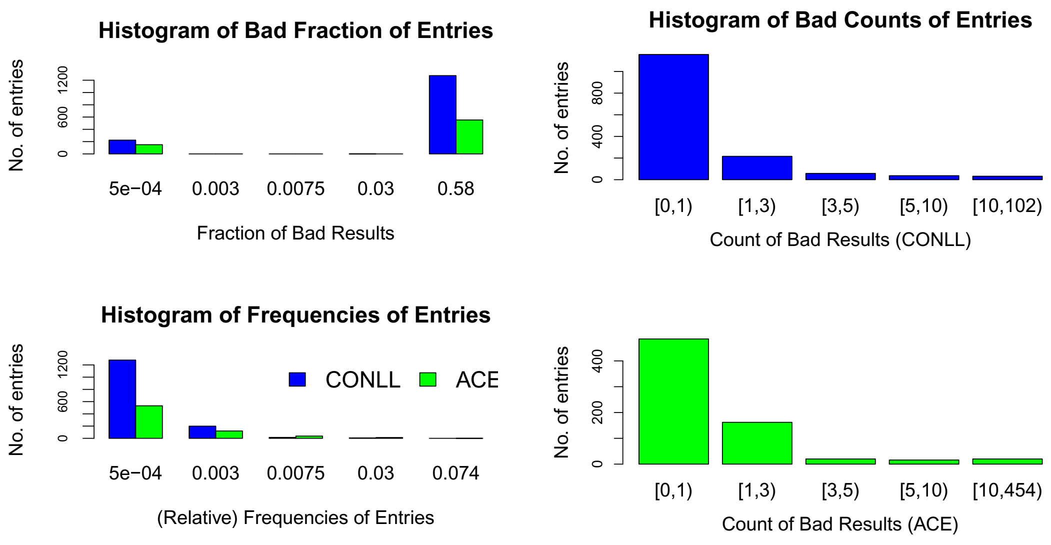


Figure 5.2: Histograms for statistics of the data : (i) Fraction of bad results produced by entries and relative frequencies of entries, (ii) Count of bad results produced by entries.

(*i.e.*, maximizes $\Delta F/\Delta R$ at each step), (4) *RecBudget-BadCount*: chooses the top-k entries in the decreasing order of the *number* of `Bad` results produced by the entries. Each algorithm is run until the given recall budget ρ is exhausted.

Observations. For size constraint, K-Optimal and K-Greedy perform better than selecting by the fraction or number of `Bad` results, and their performance gap is negligible on our datasets (Example 5.5 shows that K-Greedy may not give optimal results for some datasets). On the other hand, for recall constraint, RecBudget-Near-Opt and RecBudget-Greedy-FR have similar performance and are better than selecting entries by ΔF or the number of `Bad` results. In fact, it can be shown that the conditions for these two algorithms (sorting by fraction of bad results and selecting by $\Delta F/\Delta R$) are approximately equivalent assuming both F and R to be continuous functions of entry precisions (*i.e.*, approximating $\Delta F/\Delta R$ by dF/dR).

Figure 5.4 gives the running time of the best two algorithms for both size and recall constraints, averaged over 100 runs. This figure shows that the algorithms are efficient and therefore can be embedded in interactive tools. The greedy algorithms take more time than the (near) optimal algorithms at higher (lower) values of k (ρ). For size constraint, optimal and greedy respectively run in $O(n \log n)$ and $O(kn)$ time, n being the number

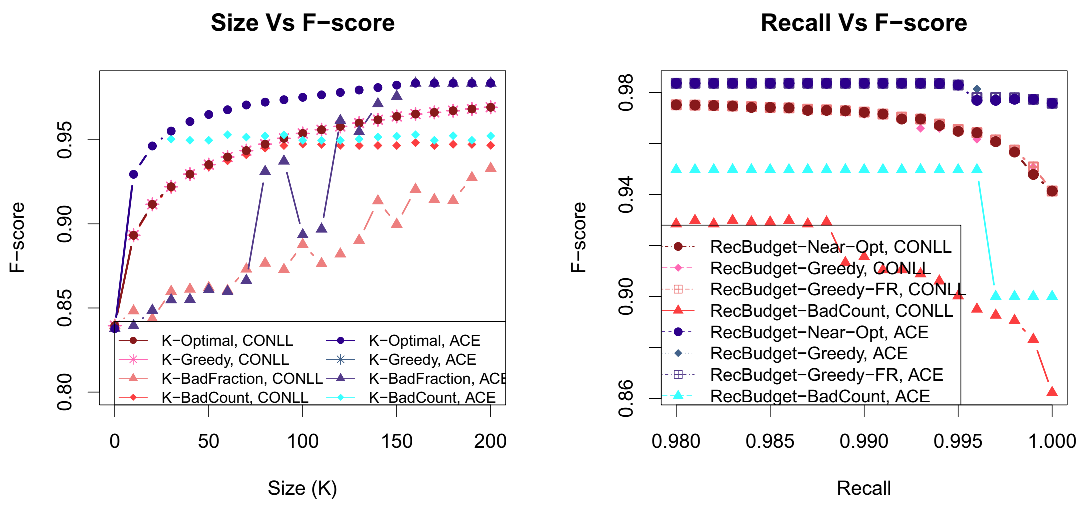


Figure 5.3: Performance of refinement optimization algorithms for single dictionary: (i) Size constraint, (ii) Recall constraint

of dictionary entries; the running times are similar for recall constraint. For $\Delta = 0.001$, K-Optimal converges in only 5 to 6 iterations. The benefit in running time is expected more for larger datasets, and therefore K-Optimal and RecBudget-Near-Opt should be used in practice.

5.5.2.2 Refinement Algorithms for Multiple Dictionary Case

For the multiple dictionary case, we compare the greedy algorithms proposed in Section 5.4.2 with other obvious approaches. The algorithms compared for size constraint are (see Figure 5.5 (i)): (1) *K-Greedy*: selects the next entry giving maximum ΔF, (2) *K-BadCount*: selects the next entry that has produced the maximum *number* of `Bad` results, (3) *K-BadFrac*: selects the next entry that has the maximum *fraction* of `Bad` results among the results it has produced.

The corresponding algorithms for recall constraint are (1) *RecBudget-Greedy-FR* (selects the next entry giving maximum $\Delta F/\Delta R$), (2) *RecBudget-BadCount*, and (3) *RecBudget-BadFrac* (see Figure 5.5 (ii)).

Observations. Figure 5.5 shows that the proposed greedy algorithms perform better than the other algorithms for both size and recall constraints. Figure 5.4 compares the

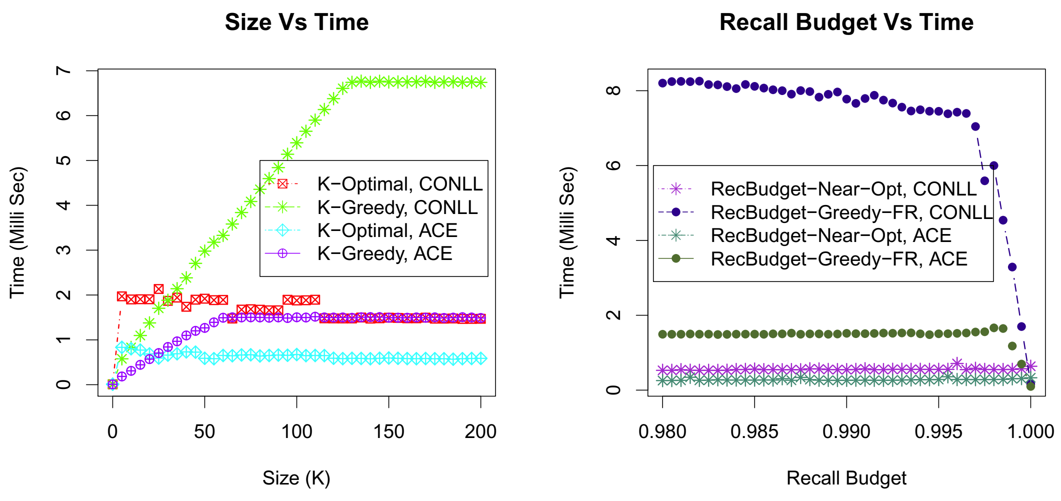


Figure 5.4: Running times of refinement algorithms for single dictionary : (i) Size constraint, (ii) Recall constraint

running time of the best two algorithms for both size and recall constraints (greedy and sorting by the fraction of bad results), averaged over 10 runs. Naturally, K-BadFrac and RecBudget-BadFrac have better running time than the greedy algorithms as they involve a single round of sorting of the entries. Then the entries are selected according to this order. The greedy algorithms needs to compute ΔF or $\Delta F/\Delta R$ at each step for every remaining entry w. This requires deciding whether the provenance expressions of the results produced by w evaluates to FALSE when w is selected in addition to the entries already chosen for deletion. The runtime for RecBudget-Greedy-FR is high even at $\rho = 1.0$ (unlike K-Greedy at $k = 0$) since even then many entries may get deleted that produce *no* `Good` results. K-Greedy and K-BadFrac give similar F-scores at higher values of k, so K-BadFrac may be used for some practical purposes at these values of k.

5.5.3 Dictionary Refinement with Incomplete Labeling

Now we evaluate our label estimation approaches for incomplete labeling, *i.e.*, when some of the results are *not* labeled as `Good` or `Bad`. The label of each unlabeled results will be estimated using the labeled results using the algorithms in Section 5.3; for the labeled results, their actual labels will be retained. Our proposed algorithms in Section 5.4 will

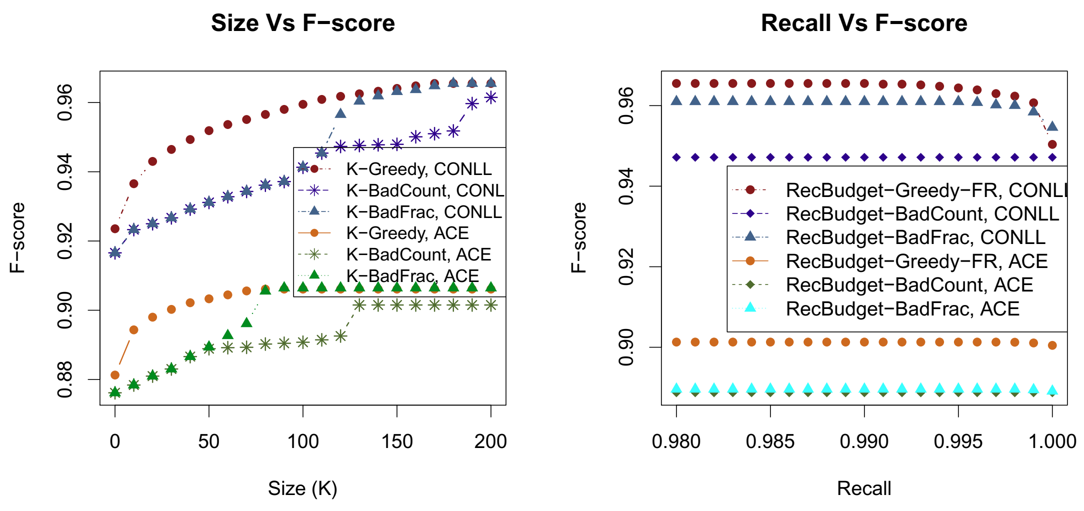


Figure 5.5: Refinement algorithms for multiple dictionary: (i) Size Vs. F-score, (ii) Recall budget Vs. F-score

be evaluated on the entire dataset. This we will refer to as the *ESTIMATED* approach in the figures (Figure 5.7 and 5.8). The ESTIMATED approach will be compared against the *NOT-ESTIMATED* approach, where the algorithms are run *only* on the labeled result set.

We vary the fraction of labeled results compared to the entire dataset in the figures. The set of deleted entries returned by each algorithm for both ESTIMATED and NOT-ESTIMATED approach are evaluated against the actual `Good`/`Bad` labels for the entire resultset to obtain the F-score after refinement. In each run, a random subset of the results is considered as the labeled resultset according to the desired fraction of labeled results. The algorithms are repeated for 10 runs, using different random subsets of labeled results (varying the initial random seed), and the mean F-scores are plotted in Figure 5.7 and 5.8.

5.5.3.1 Incomplete Labeling for Single Dictionary Case

As mentioned in Section 5.3.3 (Observation 5.2), the label of an occurrence produced by an entry is simply the empirical entry-precision of the entry (fraction of `Good` counts, and 0.5 if no occurrences are labeled), and therefore, the EM algorithm is not run for this case. Figure 5.7 (i) shows that, for size constraint, ESTIMATED approach gives better F-score over NOT-ESTIMATED and the benefit is higher when fewer results are labeled

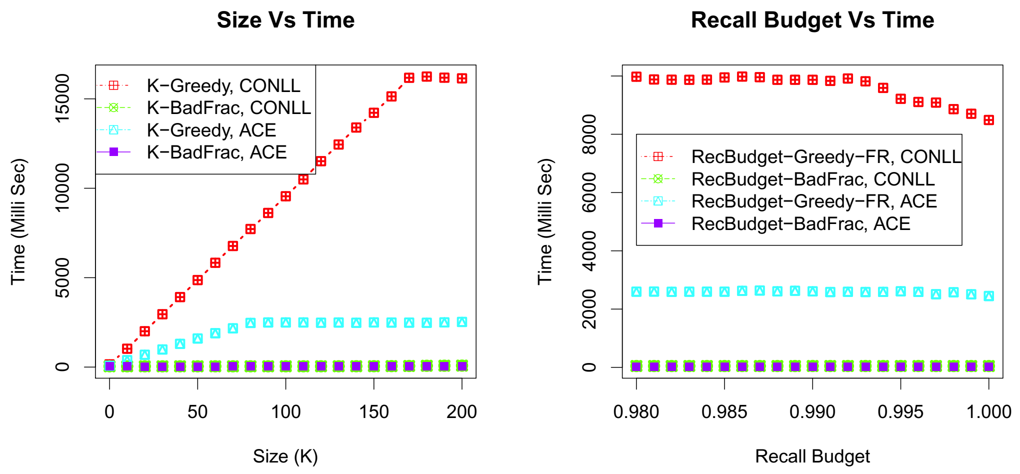


Figure 5.6: Running time for multiple dictionary (i) Size Vs. Time, (ii) Recall Budget Vs. Time

(both approaches are equivalent when fraction of labeled results = 1). However, for recall constraint and RecBudget-Near-Opt, both approaches give essentially the same result. This is as expected, since the entries are chosen according to the fraction of `Bad` results which is the same for both labeled and entire resultset.

5.5.3.2 Incomplete Labeling for Multiple Dictionary Case

We first estimate the entry-precisions by the EM algorithm that uses the provenance of the labeled results (ref. Section 5.3.3) and their labels. The EM algorithm is fast enough for practical applications (it converges in about 15-20 steps and takes about 2 seconds to run). Then the label of the unlabeled results are estimated by evaluating their provenance using the estimated entry-precisions (ref. Section 5.3.2). Figure 5.8 shows that ESTIMATED approach is better or at least as good as the NOT-ESTIMATED approach. However, the improvement is not as good as the single dictionary case due to the simplified independence assumptions. It will be an important future work to have a more accurate estimation of the labels incorporating correlations among entries.

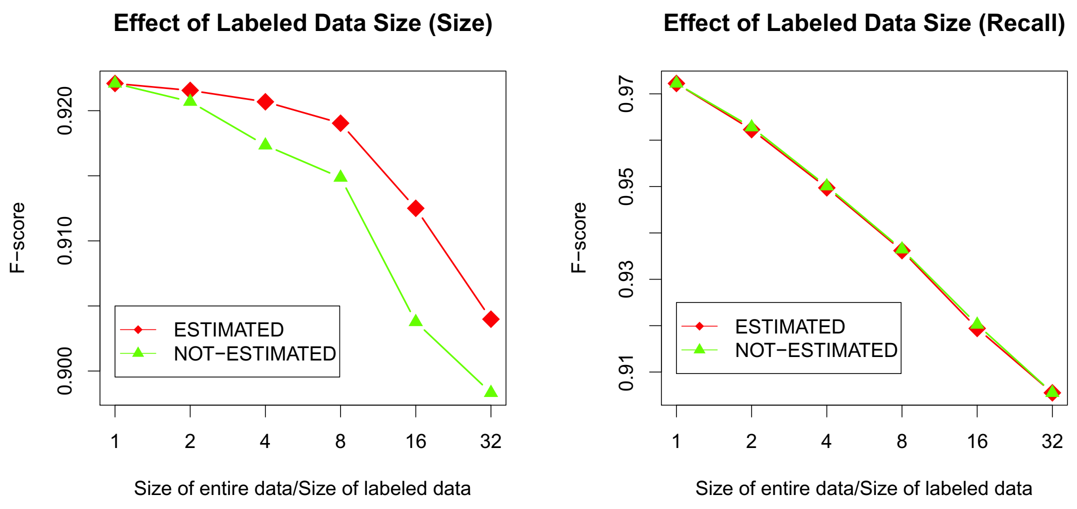


Figure 5.7: Effect of the labeled data size on F-score for single dictionary (i) Size constraint, (ii) Recall constraint.

5.5.3.3 Qualitative Evaluation

We conclude our experiments by qualitatively evaluating the entries returned by our algorithms on small labeled data. In Figure 5.9, we show some of the entries returned by our algorithms (the optimal algorithm K-Optimal for size constraint, and the near-optimal algorithm for recall-constraint RecBudget-Near-Opt) in the single dictionary case. This figure gives the first 10 entries in an arbitrary run of the algorithms when $1/10$-th of the results are labeled, and shows the counts of `Good` and `Bad` results for these entries in both labeled data and the entire data set. Clearly, these entries are incorrect or ambiguous as Person names. We see that the entries removed by the near-optimal algorithm for recall budget are the ones with fraction of `Bad` results being 1 (they improve precision and F-score without reducing recall). However, for the optimal under size constraint, entries with non-zero `Good` counts may also be chosen as the number of `Good` and `Bad` results also play an important role.

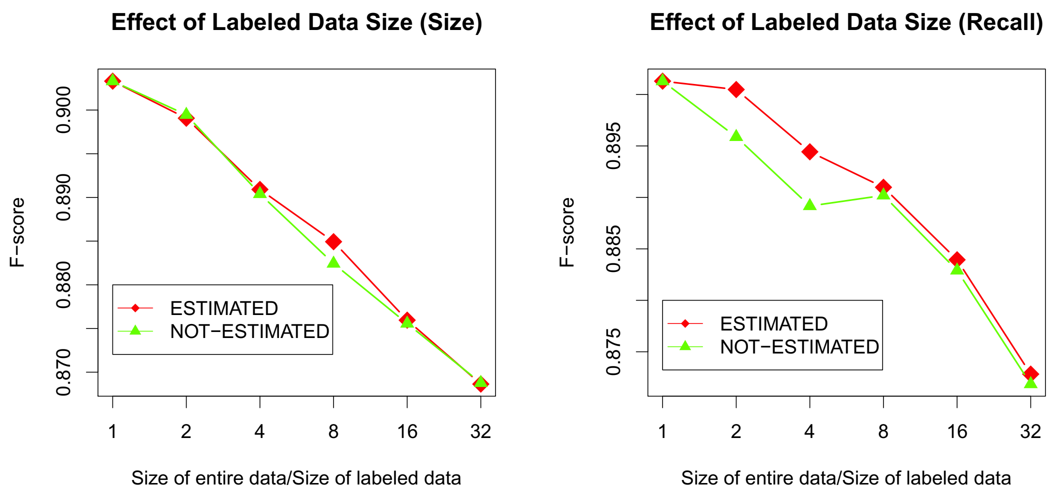


Figure 5.8: Effect of the labeled data size on F-score for multiple dictionary (i) Size constraint, (ii) Recall constraint.

5.6 Related Work

Previous work on entity extraction from lists on the web (*e.g.*, [73]), open information extraction (*e.g.*, [8, 177]) and dictionary learning (*e.g.*, [150]) addresses the problem of creating new dictionaries of terms from unstructured or semi-structured data sources by exploiting contextual patterns or the structure of web pages. Our work is complementary to these approaches, and can be used to further refine such automatically generated dictionaries.

Liu et al. [118] proposed a provenance-based framework for refining information extraction rules. They showed how to use provenance to compute *high-level changes*, a specific intermediate result whose removal from the output of an operator causes the removal of a false positive from the result, and how multiple high-level changes can be realized via a *low-level change*: a concrete change to the operator that removes one or more intermediate results from the output of the operator. While [118] focused on the general refinement framework based on high-level and low-level changes, it did not go into depth about the problems of computing specific types of low-level changes. In [151], we formally study one specific and important type of low-level change, that of refining

K-Optimal		
	$(good, bad)$ Count	
Entry	Labeled	Labeled + Unlabeled
china	(0, 12)	(0, 100)
kong	(0, 11)	(0, 70)
june	(0, 9)	(0, 97)
hong	(1, 10)	(2, 71)
september	(0, 8)	(0, 101)
king	(0, 5)	(6, 20)
louis	(1, 6)	(4, 33)
long	(0, 4)	(0, 66)
cleveland	(0, 4)	(0, 30)
april	(0, 4)	(0, 30)

RecBudget-Near-Opt		
	$(good, bad)$ Count	
Entry	Labeled	Labeled + Unlabeled
china	(0, 12)	(0, 100)
kong	(0, 11)	(0, 70)
june	(0, 9)	(0, 97)
september	(0, 8)	(0, 101)
king	(0, 5)	(6, 20)
long	(0, 4)	(0, 66)
cleveland	(0, 4)	(0, 30)
april	(0, 4)	(0, 42)
january	(0, 4)	(0, 28)
re	(0, 4)	(0, 54)

Figure 5.9: Top 10 entries output by the (near) optimal algorithms for size and recall constraints.

dictionaries used in an extractor, especially in the presence of sparse labeled data which was not considered in [118]. Moreover, our study of the single dictionary refinement problem is of independent interest: our solution is useful in refining manually or automatically generated dictionaries that can be used as basic features not only in rule-based, but also statistical (machine learning) information extraction systems.

Also related are recent studies on *causality* [125] and *deletion propagation* [35, 108]. In [125], the input consists of source tuples, an output Boolean vector and a ground truth Boolean vector, and the goal is to compute the *responsibility* of every source tuple using modifications to the source tuples such that the output is identical to the ground truth. In other words, the modifications should remove all incorrect results, while keeping all correct results. On the other hand, [35, 108] study the problem of deleting an incorrect answer tuple of a SPJU query while minimizing the *view-side-effect*, i.e. the number of other answer tuples deleted. We consider extractors consisting essentially of SPJU queries, where dictionary entries correspond to source tuples and extraction results correspond to outputs. As such, our work can be seen as an alternative objective for these related problems that tries to balance between deletion of incorrect tuples while keeping the correct tuples. In addition, we seek refinements that generalize well in practice, and do not restrict ourselves to obtaining the expected extraction result on the labeled data, which is not useful in our information extraction scenario. To the best of our knowledge, ours is the first formal study of dictionary refinement for rule-based information extraction.

5.7 Conclusions

In this work, we defined and studied one important aspect of building a high quality information extraction, that of refining dictionaries used in the extractor. We provided rigorous theoretical analysis and experimental evaluation of the optimization problems that such refinement entails. In addition, we proposed and evaluated statistical approaches for coping with the scarcity of labeled data. First and foremost, our experimental results show that dictionary refinement can significantly increase the quality (i.e. F-score) of the extraction results using even a small amount of labeled data. Further, our experimental results show which of the several natural baseline refinement algorithms perform better on real data sets. Our proposed algorithms and label estimation techniques perform better than the other approaches.

There are several interesting future directions. So far, we have only considered positive queries, whereas the difference operation in the extractor rules may be sometimes necessary. Since dictionaries are expected to contain large number of entries, efficient handling of large provenance graphs from an extractor with difference operators poses practical challenges. We would also like to explore a more detailed modeling of correlations between entries and occurrences, as well as analyze the choice of rule refinement. Finally, since manually labeling results is a time-consuming process, adaptively labeling a corpus for dictionary refinement given a budget on the number of labels is another important future direction.

Chapter 6

Provenance Views for Module Privacy

In the previous three chapters, we used Boolean provenance to derive uncertain information in the input or the output in the relational setting. In general, data provenance is useful to understand and debug an experiment or transaction in domains like scientific research and business processes. These applications involve more complex data (files) than tuples, and more complicated processes (sophisticated algorithms) than relational queries. In contrast to fine-grained Boolean provenance, coarse-grained workflow provenance is more useful in this regard, which simply records the data values and processes used in the generation of a result (see the discussion in Chapter 2). However, substantial investments in developing a workflow and its components, and the presence of proprietary and sensitive elements, restrict publishing and using provenance information in these domains. In particular, in this dissertation we focus on privacy of proprietary or commercial modules when they belong to a *workflow*, which is a key component in scientific research to capture the processes and data flows in an experiment. This chapter presents our first work in *module privacy in workflow provenance*, that shows how intentionally introducing uncertainty in the provenance information allows publishing provenance while ensuring module privacy.

6.1 Overview

As discussed in Section 2.2, the importance of data provenance has been widely recognized in the context of workflows, a set of modules inter-connected via data paths to model and formalize a task like scientific experiment or a business process. A module in a workflow can be abstracted as a function that takes an input and produces an output. Modules used in scientific workflows may be proprietary (*e.g.*, commercial software), where the owner of the module wants to ensure that the module functionality (*i.e.*, the output for a given input) is not revealed from the published provenance information. In [63] we focus on the problem of preserving the privacy of *module functionality*, i.e. the mapping between input and output values produced by the module (rather than the actual *algorithm* that implements it).

To ensure the privacy of module functionality, we extend the notion of ℓ-diversity [121] to our network setting[36]: A module with functionality m in a workflow is said to be Γ-private if for every input x, the actual value of the output $m(x)$ is indistinguishable from $\Gamma - 1$ other possible values w.r.t. the visible data values in the provenance relation. This is achieved by carefully selecting a subset of data items and hiding those values in *all* executions of the workflow – i.e. by showing the user a *view* of the provenance relation for the workflow in which the selected data items (attributes) are hidden. Γ-privacy of a module ensures that even with arbitrary computational power and access to the view for all possible executions of workflow, an adversary can not guess the correct value of $m(x)$ with probability $> \frac{1}{\Gamma}$.

To reflect the fact that some data may be more valuable to the user than other data, we assign a *cost* to each data item in the workflow, which indicates the utility lost to the user when the data value is hidden. Identical privacy guarantees can be achieved by hiding different subsets of data of different cost. Further, a workflow may have *data sharing* (i.e. computed data items from a module can be passed as input to more than one module in the workflow). Therefore hiding some data can be used to guarantee privacy for more than one module in the network.

[36]In the Related Work, we discuss why a stronger notion of privacy, like differential privacy, is not suitable here.

The problem we address in this work is the following: *Given a workflow W, cost of each data item in the workflow, and a privacy parameter Γ, minimize the cost of hidden data while guaranteeing that individual private modules in the given workflow are Γ-private.* We call this the *secure-view* problem. We formally define the problem, study its complexity, and offer algorithmic solutions. We summarize our contributions in this chapter below.

1. We formalize the notion of Γ-privacy of a private module when it is a standalone entity (*standalone privacy*) as well as when it is a component of a workflow (with arbitrary directed acyclic graph structure and arbitrary data sharing) interacting with other modules (*workflow privacy*).

2. For standalone modules, we analyze the computational and communication complexity of obtaining a minimal cost set of input/output data items to hide such that the remaining, visible attributes guarantee Γ-privacy (a *safe subset*). We call this the *standalone secure-view* problem. Although finding the optimal solution to this problem has an exponential lower bound on time complexity, we argue that the trivial exponential-time algorithms to find the optimal solutions or expert-knowledge from the module designers may be acceptable for a single module having much fewer inputs and outputs.

3. Then we study workflows in which all modules are *private*, i.e. modules for which the user has no a priori knowledge and whose behavior must be hidden. For such *all-private workflows*, we analyze the complexity of finding a minimum cost set of data items in the workflow, as a whole, to hide such that the remaining visible attributes guarantee Γ-privacy for all modules. We call this the *workflow secure-view* problem. Although the privacy of a module within a workflow is inherently linked to the workflow topology and functionality of other modules, we are able to show that guaranteeing workflow secure-views in this setting essentially reduces to implementing the standalone privacy requirements for each module. We then study two variants of the workflow secure-view problem, one in which module privacy is specified in terms of attribute sets (*set constraints*) and one in which module privacy is specified in terms of input/output cardinalities (*cardinality constraints*). Both

variants are easily shown to be NP-hard, and we give poly-time approximation algorithms for these problems. While the cardinality constraints version has an linear-programming-based $O(\log n)$-approximation algorithm, the set constraints version is much harder to approximate. However, both variants becomes more tractable when the workflow has *bounded data sharing*, i.e. when a data item acts as input to a small number of modules. In this case a constant factor approximation is possible, although the problem remains NP-hard even without any data sharing.

4. Then we consider *general workflows*, i.e workflows which contain private modules as well as modules whose behavior is known (*public* modules). Here we show that ensuring standalone privacy of private modules no longer guarantees their workflow privacy. However, by making some of the public modules private (*privatization*) we can attain workflow privacy of all private modules in the workflow. Since privatization has a cost, the optimization problem, becomes much harder: Even without data sharing the problem is $\Omega(\log n)$-hard to approximate. However, for both all-private and general workflows, there is an LP-based $\ell_{\max}$-approximation algorithm, where $\ell_{\max}$ is the length of longest requirement list for any module.

Organization. Section 6.2 formalizes the notions of Γ-privacy of a module, both when it is standalone and when it appears in a workflow. The secure-view problem for standalone module privacy is studied in Section 6.3. Section 6.4 then studies the problem for workflows consisting only of private modules, whereas Section 6.5 generalizes the results to general workflows consisting of both public and private modules. Finally we conclude and discuss directions for future work in Section 6.7.

6.2 Module Privacy

Here we formally define module privacy with respect to workflow provenance. We first consider the privacy of a single module, which we call *standalone module privacy*. Then we consider privacy when modules are connected in a workflow, which we call *workflow module privacy*.

6.2.1 Standalone Module Privacy

Our approach to ensuring standalone module privacy, for a module represented by the relation R, will be to hide a carefully chosen subset of R's attributes. In other words, we will project R on a restricted subset V of attributes (called the *visible attributes* of R), allowing users access only to the view $R_V = \pi_V(R)$. The remaining, non visible, attributes of R are called *hidden attributes*.

We distinguish below two types of modules. (1) *Public modules* whose behavior is fully known to users when the name of the module is revealed. Here users have a priori knowledge about the full content of R and, even if given only the view R_V, they are able to fully (and exactly) reconstruct R. Examples include reformatting or sorting modules. (2) *Private modules* where such a priori knowledge does not exist, even if the name of the module is revealed. Here, the only information available to users, on the module's behavior, is the one given by R_V. Examples include proprietary software, e.g. a genetic disorder susceptibility module.

Given a view (projected relation) R_V of a private module m, the *possible worlds* of m are all the possible full relations (over the same schema as R) that are consistent with R_V w.r.t the visible attributes. Formally,

Definition 6.1. *Let m be a private module with a corresponding relation R, having input and output attributes I and O, resp., and let $V \subseteq I \cup O$. The set of* possible worlds *for R w.r.t. V, denoted $\texttt{Worlds}(R,V)$, consist of all relations R' over the same schema as R that satisfy the functional dependency $I \to O$ and where $\pi_V(R') = \pi_V(R)$.*

Example 6.2. Consider the example workflow and provenance relations in Figure 2.5 which we will use as a running example in this section. Returning to module m_1, suppose the visible attributes are $V = \{a_1, a_3, a_5\}$ resulting in the view R_V in Figure 6.1a. For clarity, we have added $I \cap V$ (visible input) and $O \cap V$ (visible output) above the attribute names to indicate their role. Naturally, $R_1 \in \texttt{Worlds}(R_1, V)$. Figure 6.1 shows four additional sample relations $R_1^1, R_1^2, R_1^3, R_1^4$ in $\texttt{Worlds}(R_1, V)$, such that $\forall i \in [1,4], \pi_V(R_1^i) = \pi_V(R_1) = R_V$. (Overall there are sixty four relations in $\texttt{Worlds}(R_1, V)$). □

To guarantee privacy of a module m, the view R_V should ensure some level of uncertainly w.r.t the value of the output $m(\pi_I(\mathbf{t}))$, for tuples $t \in R$. To define this, we introduce

$I \cap V$	$O \cap V$	
a_1	a_3	a_5
0	0	1
0	1	0
1	1	0
1	1	1

(a) $R_V = \pi_V(R_1)$ for V=$\{a_1,a_3,a_5\}$

a_1	a_2	a_3	a_4	a_5
0	0	0	0	1
0	1	1	0	0
1	0	1	0	0
1	1	1	0	1

(b) R_1^1

a_1	a_2	a_3	a_4	a_5
0	0	0	1	1
0	1	1	1	0
1	0	1	0	0
1	1	1	0	1

(c) R_1^2

a_1	a_2	a_3	a_4	a_5
0	0	1	0	0
0	1	0	0	1
1	0	1	0	0
1	1	1	0	1

(d) R_1^3

a_1	a_2	a_3	a_4	a_5
0	0	1	1	0
0	1	0	1	1
1	0	1	0	0
1	1	1	0	1

(e) R_1^4

Figure 6.1: (a): Projected relation for m_1 when attribute $\{a_2,a_4\}$ are hidden, (b)-(d): $R_1^i \in$ `Worlds`(R_1,V), $i \in [1,4]$

the notion of Γ-standalone-privacy, for a given parameter $\Gamma \geq 1$. Informally, R_V is Γ-standalone-private if for every $t \in R$, the possible worlds `Worlds`(R,V) contain at least Γ distinct output values that could be the result of $m(\pi_I(\mathbf{t}))$. Note that, a view R_V "hides" some data values in the provenance relations by *access control*, and the user cannot see or access the hidden values.

Definition 6.3. *Let m be a private module with a corresponding relation R having input and output attributes I and O resp. Then m is* Γ-standalone-private *w.r.t a set of visible attributes V, if for every tuple* $\mathbf{x} \in \pi_I(R)$, $|\text{OUT}_{\mathbf{x},m}| \geq \Gamma$, *where* $\text{OUT}_{\mathbf{x},m} = \{\mathbf{y} \mid \exists R' \in$ `Worlds`$(R,V), \; \exists \mathbf{t}' \in R' \;\; s.t \;\; \mathbf{x} = \pi_I(\mathbf{t}') \wedge \mathbf{y} = \pi_O(\mathbf{t}')\}$.

If m is Γ-standalone-private *w.r.t. V, then we will call V a* safe subset *for m and* Γ.

Γ-standalone-privacy implies that for *any* input the adversary cannot guess m's output with probability $> \frac{1}{\Gamma}$, even if the module is executed an arbitrary number of times.

Example 6.4. It can be verified that, if $V = \{a_1, a_3, a_5\}$ then for all $\mathbf{x} \in \pi_I(R_1)$, $|\texttt{OUT}_{\mathbf{x}}| \geq 4$, so $\{a_1, a_3, a_5\}$ is safe for m_1 and $\Gamma = 4$. As an example, from Figure 6.1, when $\mathbf{x} = (0,0)$, $\texttt{OUT}_{\mathbf{x},m} \supseteq \{(0,\underline{0},1), (0,\underline{1},1), (1,\underline{0},0), (1,\underline{1},0)\}$ (hidden attributes are underlined). Also, hiding any two output attributes from $O = \{a_3, a_4, a_5\}$ ensures standalone privacy for $\Gamma = 4$, e.g., if $V = \{a_1, a_2, a_3\}$ (i.e. $\{a_4, a_5\}$ are hidden), then the input $(0,0)$ can be mapped to one of $(0,\underline{0},\underline{0}), (0,\underline{0},\underline{1}), (0,\underline{1},\underline{0})$ and $(0,\underline{1},\underline{1})$; this holds for other assignments of input attributes as well. But, $V = \{a_3, a_4, a_5\}$ (i.e. when only input attributes are hidden) is not safe for $\Gamma = 4$: for any input $\mathbf{x}$, $\texttt{OUT}_{\mathbf{x},m} = \{(0,1,1), (1,1,0), (1,0,1)\}$, containing only three possible output tuples. □

There may be several safe subsets V for a given module m and parameter Γ. Some of the corresponding R_V views may be preferable to others, e.g. they provide users with more useful information, allow to answer more common/critical user queries, etc. Let $\overline{V} = (I \cup O) \setminus V$ denote the attributes of R that do not belong to the view. If $\texttt{c}(\overline{V})$ denotes the penalty of hiding the attributes in $\overline{V}$, a natural goal is to choose a safe subset V that minimizes $\texttt{c}(\overline{V})$. To understand the difficulty of this problem, we study a version of the problem where the cost function is additive: each attribute a has some penalty value $\texttt{c}(a)$ and the penalty of hiding $\overline{V}$ is $\texttt{c}(\overline{V}) = \Sigma_{a \in \overline{V}} \texttt{c}(a)$. We call this optimization problem the *standalone* `Secure-View` *problem* and discuss it in Chapter 6.

6.2.1.1 Workflow Module Privacy

To define privacy in the context of a workflow, we first extend our notion of *possible worlds* to a workflow view. Consider the view $R_V = \pi_V(R)$ of a workflow relation R. Since the workflow may contain private as well as public modules, a possible world for R_V is a full relation that not only agrees with R_V on the content of the visible attributes, but is also consistent w.r.t the expected behavior of the public modules. In the following definitions, $m_1, \cdots, m_n$ denote the modules in the workflow W and F denotes the set of functional dependencies $I_i \rightarrow O_i$, $i \in [1,n]$ in the relation R.

Definition 6.5. *The set of* possible worlds *for the workflow relation R w.r.t. V, denoted also* `Worlds`(R,V)*, consists of all the relations R' over the same attributes as R that satisfy the functional dependencies in F and where (1) $\pi_V(R') = \pi_V(R)$, and (2) for every public module m_i*

in W and every tuple $\mathbf{t}' \in R'$, $\pi_{O_i}(\mathbf{t}') = m_i(\pi_{I_i}(\mathbf{t}'))$.

Note that when a workflow consists only of private modules, the second constraint does not need to be enforced. We call these *all-private workflows* and study them in Chapter 6. We discuss workflows with both public and private modules in Chapter 6 and 7, and argue that attaining privacy in the presence of public modules is fundamentally harder.

We are now ready to define the notion of Γ-workflow-privacy, for a given parameter $\Gamma \geq 1$. Informally, a view R_V is Γ-workflow-private if for every tuple $t \in R$, and every private module m_i in the workflow, the possible worlds $\texttt{Worlds}(R, V)$ contain at least Γ distinct output values that could be the result of $m_i(\pi_{I_i}(\mathbf{t}))$.

Definition 6.6. *A private module m_i in W is* Γ-workflow-private *w.r.t a set of visible attributes V, if for every tuple $\mathbf{x} \in \pi_{I_i}(R)$, $|\texttt{OUT}_{\mathbf{x},W}| \geq \Gamma$, where $\texttt{OUT}_{\mathbf{x},W} = \{\mathbf{y} \mid \exists R' \in \texttt{Worlds}(R, V)$, s.t., $\forall \mathbf{t}' \in R'\ \ \mathbf{x} = \pi_{I_i}(\mathbf{t}') \Rightarrow \mathbf{y} = \pi_{O_i}(\mathbf{t}')\}$.*

W is called Γ-private *if every private module m_i in W is Γ-workflow-private. If W (resp. m_i) is* Γ-private *(Γ-workflow-private) w.r.t. V, then we call V a* safe subset *for Γ-privacy of W (Γ-workflow-privacy of m_i).*

For simplicity, in the above definition we assumed that the privacy requirement of every module m_i is the same Γ. The results and proofs in Chapter 6 and 7 remain unchanged when different modules m_i have different privacy requirements Γ_i.

In these chapters, for a set of visible attributes $V \subseteq A$, $\overline{V} = A \setminus V$ will denote the hidden attributes in the workflow. The following proposition is easy to verify, and will be used in our proofs:

Proposition 6.7. *If V is a safe subset for Γ-workflow-privacy of a module m_i in W, then any V' such that $V' \subseteq V$ (or, $\overline{V'} \supseteq \overline{V}$) also guarantees Γ-workflow-privacy of m_i.*

As we illustrate later, given a workflow W and a parameter Γ there may be several incomparable (in terms of set inclusion) safe subsets V for Γ-privacy of W. Our goal is to choose one that minimizes the penalty $\texttt{c}(\overline{V}) = \Sigma_{a \in \overline{V}}\ \texttt{c}(a)$ of the hidden attributes $\overline{V}$. This we call the *workflow* `Secure-View` *problem*, or simply the `Secure-View` problem, and study this optimization problem in Chapter 6 and 7.

In our complexity results, we will see the implication of limited *data sharing*, *i.e.*, when the output of a module is fed as input to several modules in a workflow. We define the degree of data sharing in a workflow as follows:

Definition 6.8. *A workflow W is said to have γ-*bounded data sharing *if every attribute in W can appear in the left hand side of at most γ functional dependencies $I_i \rightarrow O_i$.*

In the workflow in Figure 2.5, $\gamma = 2$. Intuitively, if a workflow has γ-bounded data sharing then a data item can be fed as input to at most γ different modules.

6.3 Standalone Module Privacy

We start our study of workflow privacy by considering the privacy of a standalone module, which is the simplest special case of a workflow. Hence understanding it is a first step towards understanding the general case. We will also see that standalone-privacy guarantees of individual modules may be used as building blocks for attaining workflow privacy.

We analyze below the time complexity of obtaining (minimal cost) guarantees for standalone module privacy. Though the notion of Γ-standalone-privacy is similar to the well-known notion of ℓ-diversity [121], to the best of our knowledge the time complexity of this problem has not been studied.

Optimization problems and parameters. Consider a standalone module m with input attributes I, output attributes O, and a relation R. Recall that a visible subset of attributes V is called a *safe subset* for module m and privacy requirement Γ, if m is Γ-standalone-private w.r.t. V (see Definition 6.3). If each attribute $a \in I \cup O$ has cost $\mathtt{c}(a)$, *the standalone* `Secure-View` *problem* aims to find a safe subset V s.t. the cost of the hidden attributes, $\mathtt{c}(\overline{V}) = \sum_{a \in \overline{V}} \mathtt{c}(a)$, is minimized. The corresponding decision version will take a cost limit C as an additional input, and decide whether there exists a safe subset V such that $\mathtt{c}(\overline{V}) \leq C$.

One natural way of solving the optimization version of the standalone `Secure-View` problem is to consider all possible subsets $V \subseteq I \cup O$, check if V is safe, and return the safe subset V s.t. $\mathrm{c}(\overline{V})$ is minimized. This motivates us to define and study the simpler *Safe-View problem*, which takes a subset V as input and decides whether V is a safe subset.

To understand how much of the complexity of the standalone Secure-View problem comes from the need to consider different subsets of attributes, and what is due to the need to determine the safety of subsets, we study the time complexity of standalone Secure-View, with and without access to *an oracle for the Safe-Viewproblem*, henceforth called a *Safe-View oracle*. A Safe-Vieworacle takes a subset $V \subseteq I \cup O$ as input and answers whether V is safe. In the presence of a Safe-View oracle, the time complexity of the Safe-View problem is mainly due to the number of oracle calls, and hence we study the *communication complexity*. Without access to such an oracle, we also study the *computational complexity* of this problem.

In our discussion below, $k = |I| + |O|$ denotes the total number of attributes in the relation R, and N denotes the number of rows in R (i.e. the number of executions). Then $N \leq \prod_{a \in I} |\Delta_a| \leq \delta^{|I|} \leq \delta^k$ where Δ_a is the domain of attribute a and δ is the maximum domain size of attributes.

6.3.1 Lower Bounds

We start with lower bounds for the Safe-View problem. Observe that this also gives lower bounds for the standalone Secure-View problem without a Safe-View oracle. To see this, consider a set V of attributes and assume that each attribute in V has cost > 0 whereas all other attributes have cost zero. Then Safe-View has a positive answer for V iff the standalone Secure-View problem has a solution with cost $= 0$ (i.e. one that hides only the attributes $\overline{V}$).

Communication complexity of Safe-View. Given a visible subset $V \subseteq I \cup O$, we show that deciding whether V is safe needs $\Omega(N)$ time. Note that just to read the table as input takes $\Omega(N)$ time. So the lower bound of $\Omega(N)$ does not make sense unless we assume the presence of a *data supplier* (we avoid using the term "oracle" to distinguish it from Safe-View oracle) which supplies the tuples of R on demand: Given an assignment $\mathbf{x}$ of the input attributes I, the data supplier outputs the value $\mathbf{y} = m(\mathbf{x})$ of the output attributes O. The following theorem shows the $\Omega(N)$ communication complexity lower bound in terms of the number of calls to the data supplier; namely, that (up to a constant factor) one indeed needs to view the full relation.

Theorem 6.9. ***(`Safe-View` Communication Complexity)*** *Given a module m, a subset $V \subseteq I \cup O$, and a privacy requirement Γ, deciding whether V is safe for m and Γ requires $\Omega(N)$ calls to the data supplier, where N is the number of tuples in the relation R of m.*

Proof sketch. This theorem is proved by a reduction from the *set-disjointness problem*, where Alice and Bob hold two subsets A and B of a universe U and the goal is decide whether $A \cap B \neq \phi$. This problem is known to have $\Omega(N)$ communication complexity where N is the number of elements in the universe (details are in Appendix A.4.1). □

Computational Complexity of `Safe-View`: The above $\Omega(N)$ computation complexity of `Safe-View` holds when the relation R is given explicitly tuple by tuple. The following theorem shows that even when R is described implicitly in a succinct manner, there cannot be a poly-time (in the number of attributes) algorithm to decide whether a given subset V is safe unless $P = NP$ (proof is in Appendix A.4.2).

Theorem 6.10. ***(`Safe-View` Computational Complexity)*** *Given a module m with a poly-size (in $k = |I| + |O|$) description of functionality, a subset $V \subseteq I \cup O$, and a privacy requirement Γ, deciding whether V is safe w.r.t. m and Γ is co-NP-hard in k.*

Proof sketch. The proof of this theorem works by a reduction from the UNSAT problem, where given a Boolean CNF formula g on variables $x_1, \cdots, x_\ell$, the goal is to decide whether, for *all* assignments of the variables, g is not satisfiable. Here given any assignment of the variables $x_1, \cdots, x_\ell$, $g(x_1, \cdots, x_\ell)$ can be evaluated in polynomial time, which simulates the function of the data supplier. □

Lower Bound of Standalone `Secure-View` with a `Safe-View` Oracle: Now suppose we have access to a `Safe-View` oracle, which takes care of the "hardness" of the `Safe-View` problem given in Theorems 6.9 and 6.10, in constant time. The oracle takes a visible subset $V \subseteq I \cup O$ as input, and answers whether V is safe for module m and privacy requirement Γ. The following theorem shows that the decision version of standalone `Secure-View` remains hard (i.e. not solvable in poly-time in the number of attributes):

Theorem 6.11. *(Standalone* ***`Secure-View`*** *Communication Complexity, with* ***`Safe-View`*** *oracle)* *Given a `Safe-View` oracle and a cost limit C, deciding whether there exists a safe subset $V \subseteq I \cup O$ with cost bounded by C requires $2^{\Omega(k)}$ oracle calls, where $k = |I| + |O|$.*

Proof sketch. The proof of this theorem involves a novel construction of two functions, m_1 and m_2, on ℓ input attributes and a single output attribute, such that for m_1 the minimum cost of a safe subset is $\frac{3\ell}{4}$ whereas for m_2 it is $\frac{\ell}{2}$ ($C = \frac{\ell}{2}$). In particular, for both m_1 and m_2, all subsets of size $< \frac{\ell}{4}$ are safe and all other subsets are unsafe, except that for m_2, there is exactly one special subset of size $\frac{\ell}{2}$ such that this subset and all subsets thereof are safe.

We show that for an algorithm using $2^{o(k)}$ calls, there always remains at least one special subset of size $\frac{\ell}{2}$ that is consistent with all previous answers to queries. Hence after $2^{o(k)}$ calls, if the algorithm decides that there is a safe subset with cost $\leq C$, we choose m to be m_1; on the other hand, if it says that there is no such subset, we set $m = m_2$. In both the cases the answer of the algorithm is wrong which shows that there cannot be such an algorithm distinguishing these two cases with $2^{o(k)}$ calls (details appear in Appendix A.4.3). $\square$

6.3.2 Upper Bounds

The lower bound results given above show that solving the standalone `Secure-View` problem is unlikely in time sub-exponential in k or sub-linear in N. We now present algorithms for solving the `Secure-View` and `Safe-View` problems, in time polynomial in N and exponential in k.

First note that, with access to a `Safe-View` oracle, the standalone `Secure-View` problem can be easily solved in $O(2^k)$ time, by calling the oracle for all 2^k possible subsets and outputting the safe subset with minimum cost.

Without access to a `Safe-View` oracle, we first "read" relation R using N data supplier calls. Once R is available, the simple algorithm sketched below implements the `Safe-View` oracle (i.e. tests if a set V of attributes is safe) and works in time $O(2^k N^2)$: For a visible subset V, we look at all possible assignments to the attributes in $I \setminus V$. For each input value we then check if it leads to at least $\frac{\Gamma}{\prod_{a \in O \setminus V} |\Delta_a|}$ different values of the visible output attributes in $O \cap V$ (Δ_a is the domain of attribute a). This is a necessary and sufficient condition for guaranteeing Γ privacy, since by all possible $\prod_{a \in O \setminus V} |\Delta_a|$ extensions of the output attributes, for each input, there will be Γ different possible output values. The algorithm is simple and details can be found in [64]. We mention here also that essentially the same algorithms (with same upper bounds) can be used to output *all* safe attribute sets of a standalone module, rather than just one with minimum cost. Such exhaustive enumeration will be useful in the following sections.

Remarks. These results indicate that, in the worse case, finding a minimal-cost safe attribute set for a module may take time that is exponential in the number of attributes. Note, however, that the number of attributes of a single module is typically not large (often less than 10, see [2]), so the computation is still feasible. Expert knowledge of module designers, about the module's behavior and safe attribute sets may also be exploited to speed up the computation. Furthermore, a given module is often used in many workflows. For example, sequence comparison modules, like BLAST or FASTA, are used in many different biological workflows. We will see that safe subsets for individual modules can be used as building blocks for attaining privacy for the full workflow. The effort invested in deriving safe subsets for a module is thus amortized over all uses.

6.4 All-Private Workflows

We are now ready to consider workflows that consist of several modules. We first consider in this section workflows where all modules are *private* (called *all-private workflows*). Workflows with a mixture of private and public modules are then considered in Section 6.5.

As in Section 6.3, we want to find a safe visible subset V with minimum cost s.t. all the modules in the workflow are Γ-workflow-private w.r.t. V (see Definition 6.6). One option is to devise algorithms similar to those described for standalone modules in the previous section. However, the time complexity of those algorithms is now exponential in the *total number of attributes of all modules in the workflow* which can be as large as $\Omega(nk)$, n being the number of modules in the workflow and k the maximum number of attributes of a single module. To avoid the exponential dependency on n, the number of modules in the workflow,which may be large [2], and to exploit the safe attribute subsets for standalone modules, which may have been already computed, we attempt in this section to assemble workflow privacy guarantees out of standalone module guarantees. We first prove, in Section 6.4.1, that this is indeed possible. Then, in the rest of this section, we study the optimization problem of obtaining a safe view with minimum cost.

Let W be a workflow consisting of modules $m_1, \cdots, m_n$, where I_i, O_i denote the input and output attributes of m_i, $i \in [1, n]$, respectively. We use below R_i to denote the relation for the *standalone* module m_i. The relations $R = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$, with attributes $A = \bigcup_{1=1}^{n}(I_i \cup O_i)$, then describes the possible executions of W. Note that if one of the modules in W is not a one-to-one function then the projection $\pi_{I_i \cup O_i}(R)$ of the relation R on $I_i \cup O_i$ may be a subset of the (standalone) module relation R_i.

In this section (and throughout the rest of the chapter), for a set of visible attributes $V \subseteq A$, $\overline{V} = A \setminus V$ will denote the hidden attributes. Further, $V_i = (I_i \cup O_i) \cap V$ will denote the visible attributes for module m_i, whereas $\overline{V_i} = (I_i \cup O_i) \setminus V_i$ will denote the hidden attributes for m_i, for $i \in [1, n]$.

6.4.1 Standalone-Privacy vs. Workflow-Privacy

We show that if a set of visible attributes guarantees Γ-standalone-privacy for a module, then if the module is placed in a workflow where only a subset of those attributes is made visible, then Γ-workflow-privacy is guaranteed for the module in this workflow. In other words, in an all-private workflow, hiding the union of the corresponding hidden attributes of the individual modules guarantees Γ-workflow-privacy for all of them[37]. We formalize this next.

Theorem 6.12. *Let W be an all-private workflow with modules $m_1, \cdots, m_n$. Given a parameter $\Gamma \geq 1$, let $V_i \subseteq (I_i \cup O_i)$ be a set of visible attributes w.r.t which m_i, $i \in [1, n]$, is Γ-standalone-private. Then the workflow W is Γ-private w.r.t the set of visible attributes V s.t. $\overline{V} = \bigcup_{i=1}^{n} \overline{V_i}$.*

Before we prove the theorem, recall that Γ-standalone-privacy of a module m_i requires that for every input $\mathbf{x}$ to the module, there are at least Γ *potential outputs* of $\mathbf{x}$ in the

[37]By Proposition 6.7, this also means that hiding any superset of this union would also be safe for the same privacy guarantee.

possible worlds $\texttt{Worlds}(R_i, V_i)$ of the standalone module relation R_i w.r.t. V_i; similarly, Γ-workflow-privacy of m_i requires at least Γ potential outputs of $\mathbf{x}$ in the possible worlds $\texttt{Worlds}(R, V)$ of the workflow relation R w.r.t. V. Since $R = R_1 \bowtie \cdots \bowtie R_n$, a possible approach to prove Theorem 6.12 may be to show that, whenever the hidden attributes for m_i are also hidden in the workflow W, any relation $R'_i \in \texttt{Worlds}(R_i, V_i)$ has a corresponding relation $R' \in \texttt{Worlds}(R, V)$ s.t. $R'_i = \pi_{I_i \cup O_i}(R')$. If this would hold, then for $\overline{V} = \bigcup_{i=1}^{n} \overline{V_i}$, the set of possible outputs, for any input tuple $\mathbf{x}$ to a module m_i, will remain unchanged.

Unfortunately, Proposition 6.13 below shows that the above approach fails. Indeed, $|\texttt{Worlds}(R, V)|$ can be significantly smaller than $|\texttt{Worlds}(R_i, V_i)|$ even for very simple workflows.

Proposition 6.13. *There exist a workflow W with relation R, a module m_1 in W with (standalone) relation R_1, and a set of visible attributes V_1 that guarantees both Γ-standalone-privacy and Γ-workflow-privacy of m_1, such that the ratio of $|\texttt{Worlds}(R_1, V_1)|$ and $|\texttt{Worlds}(R, V_1)|$ is* doubly exponential *in the number of attributes of W.*

Proof sketch. To prove the proposition, we construct a simple workflow with two modules m_1, m_2 connected as a chain. Both m_1, m_2 are one-one functions with k Boolean inputs and k Boolean outputs (for example, assume that m_1 is an identity function, whereas m_2 reverses the values of its k inputs). The module m_1 gets initial input attribute set I_1, produces $O_1 = I_2$ which is fed to the module m_2 as input, and m_2 produces final attribute set O_2. Let V_1 be an arbitrary subset of O_1 such that $|\overline{V_1}| = \log \Gamma$ (we assume that Γ is a power of 2). It can be verified that, m_1 as a standalone module is Γ-standalone-private w.r.t. visible attributes V_1 and both m_1, m_2, being one-one modules, are Γ-workflow-private w.r.t. V_1.

We show that the one-one nature of m_1 and m_2 restricts the size of $\texttt{Worlds}(R, V_1)$ compared to that of $\texttt{Worlds}(R_1, V_1)$. Since both m_1 and m_2 are one-one functions, the workflow W also computes a one-one function. Hence any relation S in $\texttt{Worlds}(R, V_1)$ has to compute a one-one function as well. But when m_1 was standalone, any function consistent with V_1 could be a member of $\texttt{Worlds}(R_1, V_1)$. By a careful computation, the ratio can be shown to be doubly exponential in k (details appear in Appendix A.4.4). $\square$

Nevertheless, we show below that for every input $\mathbf{x}$ of the module, the set of its possible outputs, in these worlds, is exactly the same as that in the original (much larger number of) module worlds. Hence privacy is indeed preserved.

In proving Theorem 6.12, our main technical tool is Lemma 6.14, which states that given a set of visible attributes V_i of a standalone module m_i, the set of possible outputs for *every* input $\mathbf{x}$ to m_i remains unchanged when m_i is placed in an all-private workflow, provided the corresponding hidden attributes $\overline{V_i}$ remains hidden in the workflow.

Recall that OUT_{x,m_i} and $\text{OUT}_{x,W}$ denote the possible output for an input $\mathbf{x}$ to module m_i w.r.t. a set of visible attributes when m_i is standalone and in a workflow W respectively (see Definition 6.3 and Definition 6.6).

Lemma 6.14. *Consider any module m_i and any input $\mathbf{x} \in \pi_{I_i}(R)$. If $\mathbf{y} \in \text{OUT}_{x,m_i}$ w.r.t. a set of visible attributes $V_i \subseteq (I_i \cup O_i)$, then $\mathbf{y} \in \text{OUT}_{x,W}$ w.r.t. $V_i \cup (A \setminus (I_i \cup O_i))$.*

The above lemma directly implies Theorem 6.12:

of Theorem 6.12. We are given that each module m_i is Γ-standalone-private w.r.t. V_i, i.e., $|\text{OUT}_{x,m_i}| \geq \Gamma$ for all input $\mathbf{x}$ to m_i, for all modules m_i, $i \in [1,n]$ (see Definition 6.3). From Lemma 6.14, this implies that for all input $\mathbf{x}$ to all modules m_i, $|\text{OUT}_{x,W}| \geq \Gamma$ w.r.t $V' = V_i \cup (A \setminus (I_i \cup O_i))$. For this choice of V', $\overline{V'} = A \setminus V' = (I_i \cup O_i) \setminus V_i = \overline{V_i}$ (because, $V_i \subseteq I_i \cup O_i \subseteq A$). Now, using Proposition 6.7, when the visible attributes set V is such that $\overline{V} = \bigcup_{i=1}^{n} \overline{V_i} \supseteq \overline{V_i} = \overline{V'}$, every module m_i is Γ-workflow-private. $\square$

To conclude the proof of Theorem 6.12 we thus only need to prove Lemma 6.14. For this, we use the following auxiliary lemma.

Lemma 6.15. *Let m_i be a standalone module with relation R_i, let $\mathbf{x}$ be an input to m_i, and let $V_i \subseteq (I_i \cup O_i)$ be a subset of visible attributes. If $\mathbf{y} \in \text{OUT}_{x,m_i}$ then there exists an input $\mathbf{x}' \in \pi_{I_i}(R_i)$ to m_i with output $\mathbf{y}' = m_i(\mathbf{x}')$ such that $\pi_{V_i \cap I_i}(\mathbf{x}) = \pi_{V_i \cap I_i}(\mathbf{x}')$ and $\pi_{V_i \cap O_i}(\mathbf{y}) = \pi_{V_i \cap O_i}(\mathbf{y}')$.*

The statement of the lemma can be illustrated with the module m_1 whose relation R_1 appears Figure 2.5b. Its visible portion (for visible attributes a_1, a_3, a_5) is given in Figure 6.1a. Consider the input $\mathbf{x} = (0,0)$ to m_1 and the output $\mathbf{y} = (1,0,0)$. For $V = \{a_1, a_3, a_5\}$, $\mathbf{y} \in \text{OUT}_{x,m_1}$ (see Figure 6.1d). This is because there exists $\mathbf{x}' = (0,1)$, s.t. $\mathbf{y}' = m_1(\mathbf{x}') = (1,1,0)$, and, $\mathbf{x}, \mathbf{x}'$ and $\mathbf{y}, \mathbf{y}'$ have the same values of the visible attributes (a_1 and

$\{a_3, a_5\}$ respectively). Note that $\mathbf{y}$ does not need to be the actual output $m_1(\mathbf{x})$ on $\mathbf{x}$ or even share the same values of the visible attributes (indeed, $m_1(\mathbf{x}) = (0,1,1)$). We defer the proof of Lemma 6.15 to Appendix A.4.5 and instead briefly explain how it is used to prove Lemma 6.14.

Proof sketch of Lemma 6.14. Let us fix a module m_i, an input $\mathbf{x}$ to m_i and a candidate output $\mathbf{y} \in \texttt{OUT}_{x,m_i}$ for $\mathbf{x}$ w.r.t. visible attributes V_i. We already argued that, for $V = V_i \cup (A \setminus (I_i \cup O_i))$, $\overline{V} = A \setminus V = (I_i \cup O_i) \setminus V_i = \overline{V_i}$. We will show that $\mathbf{y} \in \texttt{OUT}_{x,W}$ w.r.t. visible attributes V by showing the existence of a possible world $R' \in \texttt{Worlds}(R, V)$, for $\overline{V} = \overline{V_i}$, s.t. $\pi_{I_i}(\mathbf{t}) = \mathbf{x}$ and $\pi_{O_i}(\mathbf{t}) = \mathbf{y}$ for some $\mathbf{t} \in R'$.

We start by replacing module m_i by a new module g_i such that $g_i(\mathbf{x}) = \mathbf{y}$ as required. But due to data sharing, other modules in the workflow can have input and output attributes from I_i and O_i. Hence if we leave the modules $m_j, j \neq i$, unchanged, there may be inconsistency in the values of the visible attributes, and the relation produced by the join of the standalone relations of the module sequence $\langle m_1, \cdots, m_{i-1}, g_i, m_{i+1}, \cdots, m_n \rangle$ may not be a member of $\texttt{Worlds}(R, V)$. To resolve this, we consider the modules in a topological order, and change the definition of all modules $m_1, \cdots, m_n$ to $g_1, \cdots, g_n$ (some modules may remain unchanged). In proving the above, the main idea is to use *tuple and function flipping* (formal definition in the appendix). If a module m_j shares attributes from I_i or O_i, the new definition g_j of m_j involves flipping the input to m_j, apply m_j on this flipped input, and then flipping the output again to the output value. The proof shows that by consistently flipping all modules, the visible attribute values remain consistent with the original workflow relation and we get a member of the possible worlds (details appear in Appendix A.4.6). □

It is important to note that the assumption of all-private workflow is crucial in proving Lemma 6.14 – if some of the modules m_j are public, we can not redefine them to g_j (the projection to the public modules should be unchanged - see Definition 6.6) and we may not get a member of $\texttt{Worlds}(R, V)$. We will return to this point in Section 6.5 when we consider workflows with a mixture of private and public modules.

6.4.2 The `Secure-View` Problem

We have seen above that one can assemble workflow privacy guarantees out of the standalone module guarantees. Recall however that each individual module may have several possible safe attributes sets (see, e.g., Example 6.4). Assembling different sets naturally lead to solutions with different cost. The following example shows that assembling optimal (cheapest) safe attributes of the individual modules may not lead to an optimal safe attributes set for the full workflow. The key observation is that, due to data sharing, it may be more cost effective to hide expensive shared attributes rather than cheap non-shared ones (though later we show that the problem remains NP-hard even without data sharing).

Example 6.16. Consider a workflow with $n+2$ modules, $m, m_1, \cdots, m_n, m'$. The module m gets an input data item a_1, with cost 1, and sends as output the same data item, a_2, with cost $1+\epsilon$, $\epsilon > 0$, to all the m_i-s. Each m_i then sends a data item b_i to m' with cost 1. Assume that standalone privacy is preserved for module m if either its incoming or outgoing data is hidden and for m' if any of its incoming data is hidden. Also assume that standalone privacy is preserved for each m_i module if either its incoming or its outgoing data is hidden. As standalone modules, m will choose to hide a_1, each m_i will choose to hide the outgoing data item b_i, and m' will choose to hide any of the b_i-s. The union of the optimal solutions for the standalone modules has cost $n+1$. However, a lowest cost solution for preserving workflow privacy is to hide a_2 and any one of the b_i-s. This assembly of (non optimal) solutions for the individual modules has cost $2+\epsilon$. In this case, the ratio of the costs of the union of standalone optimal solutions and the workflow optimal solution is $\Omega(n)$. $\square$

This motivates us to define the combinatorial optimization problem `Secure-View` (for workflow secure view), which generalizes the `Secure-View` problem studied in Section 6.3. The goal of the `Secure-View` problem is to choose, for each module, a safe set of attributes (among its possible sets of safe attributes) s.t. together the selected sets yield a minimal cost safe solution for the workflow. We define this formally below. In particular, we consider the following two variants of the problem, trading-off expressibility and succinctness.

Set constraints. The possible safe solutions for a given module can be given in the form of a list of hidden attribute sets. Specifically, we assume that we are given, for each module m_i, $i \in [1,n]$, a list of pairs $L_i = \langle(\overline{I}_i^1,\overline{O}_i^1), (\overline{I}_i^2,\overline{O}_i^2)\ldots(\overline{I}_i^{l_i},\overline{O}_i^{l_i})\rangle$. Each pair $(\overline{I}_i^j,\overline{O}_i^j)$ in the list describes one possible safe (hidden) solution for m_i: $\overline{I}_i^j \subseteq I_i$ (resp. $\overline{O}_i^j \subseteq O_i$) is the set of input (output) attributes of m_i to be hidden in this solution. l_i (the length of the list) is the number of solutions for m_i that are given in the list, and we use below $\ell_{\max}$ to denote the length of the longest list, i.e. $\ell_{\max} = \max_{i=1}^{n} \ell_i$.

When the input to the `Secure-View` problem is given in the above form (the candidate attribute sets are listed explicitly), we call it *the `Secure-View` problem with set constraints*.

Cardinality constraints. Some modules may have many possible candidate safe attribute sets. Indeed, their number may be exponential in the number of attributes of the module. This is illustrate by the following two simple examples.

Example 6.17. First observe that in any one-one function with k Boolean inputs and k Boolean outputs, hiding any k incoming or any k outgoing attributes guarantees 2^k-privacy. Thus listing all such subsets requires a list of length $\Omega(\binom{2k}{k}) = \Omega(2^k)$. Another example is majority function which takes $2k$ Boolean inputs and produces 1 if and only if the number of one-s in the input tuple is $\geq k$. Hiding either $k+1$ input bits or the unique output bit guarantee 2-privacy for majority function, but explicitly listing all possible subsets again leads to exponential length lists. $\square$

Note that, in both examples, the actual identity of the hidden input (resp. output) attributes is not important, as long as sufficiently many are hidden. Thus rather than explicitly listing all possible safe sets we could simply say what combinations of numbers of hidden input and output attributes are safe. This motivates the following variant of the `Secure-View` problem, called *the `Secure-View` problem with cardinality constraints*: Here for every module m_i we are given a list of pairs of numbers $L_i = \langle(\alpha_i^1,\beta_i^1)\ldots(\alpha_i^{l_i},\beta_i^{l_i})\rangle$, s.t. for each pair (α_i^j,β_i^j) in the list, $\alpha_i^j \leq |I_i|$ and $\beta_i^j \leq |O_i|$. The interpretation is that hiding any attribute set of m_i that consists of at least α_i^j input attributes and at least β_i^j output attributes, for some $j \in [1,\ell_i]$, makes m_i safe w.r.t the remaining visible attributes.

To continue with the above example, the list for the first module may consists of $(k,0)$

and $(0,k)$, whereas the list for the second module consists of $(k+1,0)$ and $(0,1)$.

It is easy to see that, for cardinality constraints, the lists are of size at most quadratic in the number of attributes of the given module (unlike the case of set constraints where the lists could be of exponential length)[38]. In turn, cardinality constraints are less expressive than set constraints that can specify arbitrary attribute sets. This will affect the complexity of the corresponding `Secure-View` problems.

Problem Statement. Given an input in one of the two forms, a *feasible* safe subset V for the workflow, for the version with set constraints (resp. cardinality constraints), is such that for each module m_i $i \in [1,n]$, $\overline{V} \supseteq (\overline{I}_i^j \cup \overline{O}_i^j)$ (resp. $|\overline{V} \cap I_i| \geq \alpha_i^j$ and $|\overline{V} \cap O_i| \geq \beta_i^j$) for some $j \in [1,\ell_i]$. The goal of the `Secure-View` problem is to find a safe set V where $\mathtt{c}(\overline{V})$ is minimized.

6.4.3 Complexity results

We present below theorems which give approximation algorithms and matching hardness of approximation results of different versions of the `Secure-View` problem. The hardness results show that the problem of testing whether the `Secure-View` problem (in both variants) has a solution with cost smaller than a given bound is NP-hard even in the most restricted case. But we show that certain approximations of the optimal solution are possible. Theorem 6.18 and 6.20 summarize the results for the cardinality and set constraints versions, respectively. Here we only sketch the proofs for brevity; full details appear in Appendix A.4.7.

Theorem 6.18. ***(Cardinality Constraints)*** *There is an $O(\log n)$-approximation of the `Secure-View` problem with cardinality constraints. Further, this problem is $\Omega(\log n)$-hard to approximate unless NP $\subseteq$ DTIME$(n^{O(\log\log n)})$, even if the maximum list size $\ell_{\max} = 1$, each data has unit cost, and the values of α_i^j, β_i^j-s are 0 or 1.*

Proof sketch. The proof of the hardness result in the above theorem is by a reduction from the set cover problem. The approximation is obtained by randomized rounding a carefully written linear program (LP) relaxation of this problem. A sketch is given below.

[38]In fact, if one assumes that there is no redundancy in the list, the lists become of at most of linear size.

Our algorithm is based on rounding the fractional relaxation (called the LP relaxation) of the integer linear program (IP) for this problem presented in Figure 6.2.

$$\text{Minimize } \textstyle\sum_{b \in A} c_b x_b \quad \text{subject to}$$

$$\sum_{j=1}^{\ell_i} r_{ij} \geq 1 \quad \forall i \in [1,n] \tag{6.1}$$

$$\sum_{b \in I_i} y_{bij} \geq r_{ij}\alpha_i^j \quad \forall i \in [1,n],\ \forall j \in [1,\ell_i] \tag{6.2}$$

$$\sum_{b \in O_i} z_{bij} \geq r_{ij}\beta_i^j \quad \forall i \in [1,n],\ \forall j \in [1,\ell_i] \tag{6.3}$$

$$\sum_{j=1}^{\ell_i} y_{bij} \leq x_b, \quad \forall i \in [1,n], \forall b \in I_i \tag{6.4}$$

$$\sum_{j=1}^{\ell_i} z_{bij} \leq x_b, \quad \forall i \in [1,n], \forall b \in O_i \tag{6.5}$$

$$y_{bij} \leq r_{ij}, \quad \forall i \in [1,n],\ \forall j \in [1,\ell_i],\ \forall b \in I_i \tag{6.6}$$

$$z_{bij} \leq r_{ij}, \quad \forall i \in [1,n],\ \forall j \in [1,\ell_i],\ \forall b \in O_i \tag{6.7}$$

$$x_b, r_{ij}, y_{bij}, z_{bij} \in \{0,1\} \tag{6.8}$$

Figure 6.2: IP for `Secure-View` with cardinality constraints

Recall that each module m_i has a list $L_i = \{(\alpha_i^j, \beta_i^j) : j \in [1,\ell_i]\}$, a feasible solution must ensure that for each $i \in [1,n]$, there exists a $j \in [1,\ell_i]$ such that at least α_i^j input data and β_i^j output data of m_i are hidden.

In this IP, $x_b = 1$ if data b is hidden, and $r_{ij} = 1$ if at least α_i^j input data and β_i^j output data of module m_i are hidden. Then, $y_{bij} = 1$ (resp., $z_{bij} = 1$) if both $r_{ij} = 1$ and $x_b = 1$, i.e. if data b contributes to satisfying the input requirement α_i^j (resp., output requirement β_i^j) of module m_i. Let us first verify that the IP indeed solves the `Secure-View` problem with cardinality constraints. For each module m_i, constraint (6.1) ensures that for some $j \in [1,\ell_i]$, $r_{ij} = 1$. In conjunction with constraints (6.2) and (6.3), this ensures that for some $j \in [1,\ell_i]$, (i) at least α_i^j input data of m_i have $y_{bij} = 1$ and (ii) at least β_i^j output data of m_i have $z_{bij} = 1$. But, constraint (6.4) (resp., constraint (6.5)) requires that whenever $y_{bij} = 1$ (resp., $z_{bij} = 1$), data b be hidden, i.e. $x_b = 1$, and a cost of c_b be added to the objective.

Thus the set of hidden data satisfy the privacy requirement of each module m_i and the value of the objective is the cost of the hidden data. Note that constraints (6.6) and (6.7) are also satisfied since y_{bij} and z_{bij} are 0 whenever $r_{ij} = 0$. Thus, the IP represents the `Secure-View` problem with cardinality constraints. In Appendix A.4.7 we show that simpler LP relaxations of this problem without some of the above constraints lead to unbounded and $\Omega(n)$ integrality gaps showing that an $O(\log n)$-approximation cannot be obtained from those simpler LP relaxations.

We round the fractional solution to the LP relaxation using Algorithm 9. For each $j \in [1, \ell_i]$, let I_{ij}^{min} and O_{ij}^{min} be the α_i^j input and β_i^j output data of m_i with minimum cost. Then, B_i^{min} represents $I_{ij}^{min} \cup O_{ij}^{min}$ of minimum cost.

Algorithm 9 Rounding algorithm of LP relaxation of the IP given in Figure 6.2, **Input**: An optimal fractional solution $\{x_b | b \in A\}$

```
1: Initialize $B = \phi$.
2: For each attribute $b \in A$ ($A$ is the set of all attributes in $W$), include $b$ in $B$ with probability $\min\{1, 16 x_b \log n\}$.
3: For each module $m_i$ whose privacy requirement is not satisfied by $B$, add $B_i^{min}$ to $B$.
4: Return $V = A \setminus B$ as the safe visible attribute.
```

The following lemma shows that step 2 satisfies the privacy requirement of each module with high probability:

Lemma 6.19. *Let m_i be any module in workflow W. Then with probability at least $1 - 2/n^2$, there exists a $j \in [1, \ell_i]$ such that $|I_i^h| \geq \alpha_i^j$ and $|O_i^h| \geq \beta_i^j$.*

Proof sketch. The LP solution returns a probability distribution on r_{ij}, and therefore on the pairs in list L_i. Let p be the index of the median of this distribution when list L_i is ordered by both α_i^j and β_i^j values, as described above. Our proof consists of showing that with probability $\geq 1 - 2/n^2$, $|I_i^h| \geq \alpha_{ip}$ and $|O_i^h| \geq \beta_{ip}$.

Note that since p is the median, the sum of y_{bij} over all incoming data of module v_i in the LP solution must be at least $\alpha_{ip}/2$ (from constraint (6.2)). Further, constraint (6.6) ensures that this sum is contributed to by at least $\alpha_{ip}/2$ different input data, and constraint (6.4) ensures that x_b for any input data b must be at least its contribution to this sum, i.e. $\sum_j y_{bij}$. Thus, at least $\alpha_{ip}/2$ different input data have a large enough value of x_b, and randomized rounding produces a good solution. An identical argument works for the output data of m_i (details appear in Appendix A.4.7). □

Since the above lemma holds for every module, by standard arguments, the $O(\log n)$-approximation follows. □

We next show that the richer expressiveness of set constraints increases the complexity of the problem.

Theorem 6.20. ***(Set Constraints)*** *The* `Secure-View` *problem with set constraints cannot be approximated to within a factor of $\ell_{\max}^{\epsilon}$ for some constant $\epsilon > 0$ (also within a factor of $\Omega(2^{\log^{1-\gamma} n})$ for all constant $\gamma > 0$) unless $NP \subseteq DTIME(n^{polylog\, n})$. The hardness result holds even when the maximum list size $\ell_{\max}$ is a (sufficiently large) constant, each data has unit cost, and the subsets $\overline{I}_i^j, \overline{O}_i^j$-s have cardinality at most 2. Finally, it is possible to get a factor $\ell_{\max}$-approximation in polynomial time.*

Proof sketch. When we are allowed to specify arbitrary subsets for individual modules, we can encode a hard problem like *label-cover* which is known to have no poly-logarithmic approximation given standard complexity assumptions. The corresponding approximation is obtained by an LP rounding algorithm which shows that a good approximation is still possible when the number of specified subsets for individual modules is not too large. Details can be found by Appendix A.4.8. □

The hardness proofs in the above two theorems use extensively data sharing, namely the fact that an output attribute of a given module may be fed as input to several other modules. Recall that a workflow is said to have γ-bounded data sharing if the maximum number of modules which takes a particular data item as input is bounded by γ. In real life workflows, the number of modules where a data item is sent is not very large. The following theorem shows that a better approximation is possible when this number is bounded.

Theorem 6.21. ***(Bounded Data Sharing)*** *There is a $(\gamma + 1)$-approximation algorithm for the* `Secure-View` *problem (with both cardinality and set constraints) when the workflow has γ-bounded data sharing. On the other hand, the cardinality constraint version (and consequently* *also the set constraint version) of the problem remain APX-hard even when there is* no *data sharing (i.e. $\gamma = 1$), each data has unit cost, the maximum list size $\ell_{\max}$ is 2, and the values of α_i^j, β_i^j-s are bounded by 3.*

Proof sketch. The APX-hardness in the above theorem is obtained by a reduction from vertex-cover in cubic graphs. This reduction also shows that the NP-completeness of this problem does not originate from data-sharing, and the problem is unlikely to have an exact solution even without any data sharing. The $\gamma + 1$-approximation is obtained by a greedy algorithm, which chooses the least cost attribute subsets for individual modules, and outputs the union of all of them. Since any attribute is produced by a unique module and is fed to at most γ modules, in any optimal solution, a single attribute can be used to satisfy the requirement of at most $\gamma + 1$ modules. This gives a $\gamma + 1$-approximation. Observe that when data sharing is not bounded, γ can be $\Omega(n)$ and this greedy algorithm will not give a good approximation to this problem. Details appear in Appendix A.4.9. □

6.5 Public Modules

In the previous section we restricted our attention to workflows where all modules are private. In practice, typical workflows use also public modules. Not surprisingly, this makes privacy harder to accomplish. In particular, we will see below that it becomes harder to assemble privacy guarantees for the full workflow out of those that suffice for component modules. Nevertheless a refined variant of Theorem 6.12 can still be employed.

6.5.1 Standalone vs. Workflow Privacy (Revisited)

We have shown in Section 6.4.1 (Theorem 6.12) that when a set of hidden attributes guarantees Γ-standalone-privacy for a private module, then the same set of attributes can be used to guarantee Γ-workflow-privacy *in an all-private network*. Interestingly, this is no longer the case for workflows with public modules. To see why, consider the following example.

Example 6.22. Consider a private module m implementing a one-one function with k Boolean inputs and k Boolean outputs. Hiding any $\log \Gamma$ input attributes guarantees Γ-standalone-privacy for m even if all output attributes of m are visible. However, if m gets all its inputs from a public module m' that computes some constant function (i.e. $\forall \mathbf{x}, m'(\mathbf{x}) = \mathbf{a}$, for some constant $\mathbf{a}$), then hiding $\log \Gamma$ input attributes no longer guarantees Γ-workflow-privacy of m – this is because it suffices to look at the (visible) output attributes of m to know the value $m(\mathbf{x})$ for $x = \mathbf{a}$.

In an analogous manner, hiding any $\log \Gamma$ output attributes of m, leaving all its input attributes visible, also guarantees Γ-standalone-privacy of m. But if m sends all its outputs to another public module m'' that implements a one-one invertible function, and whose output attributes happen to be visible, then for any input $\mathbf{x}$ to m, $m(\mathbf{x})$ can be immediately inferred using the inverse function of m''. $\square$

Modules that compute a constant function (or even one-one invertible function) may not be common in practice. However, this simple example illustrates where, more generally, the proof of Theorem 6.12 (or Lemma 6.14) fails in the presence of public modules: when searching for a possible world that is consistent with the visible attributes, one needs to ensure that the functions defined by the public modules remain unchanged. So we no longer have the freedom of freely changing the values of the hidden input (resp. output) attributes, if those are supplied by (to) a public module.

One way to overcome this problem is to "privatize" such problematic public modules, in the sense that the name of the public module is not revealed to users (either in the workflow specification or in its execution logs). Here we assume that once we rename a module the user loses all knowledge about it (we discuss other possible approaches in the conclusion). We refer to the public modules whose identity is hidden (resp. revealed) as *hidden* (*visible*) public modules. Observe that now, since the identity of the hidden modules is no longer known to the adversary, condition (2) in Definition 6.5 no longer needs to be enforced for them, and a larger set of possible words can be considered. Formally,

Definition 6.23. *(Definition 6.5 revisited) Let P be a subset of the public modules, and, as before, let V be a set of the visible attributes. Then, the set of* possible worlds *for the relation R w.r.t.*

V and P, denoted `Worlds`(R,V,P)*, consists of all relations R' over the same attributes as R that satisfy the functional dependencies in F and where (1) $\pi_V(R') = \pi_V(R)$, and (2) for every public module $m_i \in P$ and every tuple $\mathbf{t}' \in R'$, $\pi_{O_i}(\mathbf{t}') = m_i(\pi_{I_i}(\mathbf{t}'))$.*

The notion of Γ-privacy for a workflow W, with both private and public modules (w.r.t a set V of visible attributes and a set P of visible public modules) is now defined as before (Definition 6.6), except that the set of possible worlds that is considered is the refined one from Definition 6.23 above. Similarly, if W is Γ-*private* w.r.t. V and P, then we will call the pair (V,P) a *safe subset* for Γ-privacy of W.

We can now show that, by making visible only public modules whose input and output attribute values need not be masked, one can obtain a result analogous to Theorem 6.12. Namely, assemble the privacy guarantees of the individual modules to form privacy guarantees for the full workflow. Wlog., we will assume that $m_1, m_2, \cdots, m_K$ are the private modules and $m_{K+1}, \cdots, m_n$ are the public modules in W.

Theorem 6.24. *Given a parameter $\Gamma \geq 1$, let $V_i \subseteq (I_i \cup O_i)$, $i \in [1,K]$, be a set of visible attributes w.r.t which the private module m_i is Γ-standalone-private. Then the workflow W is Γ-private w.r.t the set of visible attributes V and any set of visible public modules $P \subseteq \{m_{K+1}, \cdots, m_n\}$, s.t. $\overline{V} = \bigcup_{i=1}^{K} \overline{V_i}$ and all the input and output attributes of modules in P are visible and belong to V.*

Proof sketch. The proof is similar to that of Thm. 6.12. Here we additionally show in a lemma analogous to Lemma 6.14 (see Appendix A.4.10) that, if a public module $m_j, j \in [K+1, n]$ is redefined to g_j, then m_j is hidden. In other words, the visible public modules in P are never redefined and therefore condition (2) in Definition 6.23 holds. □

Example 6.25. Consider a chain workflow with three modules $m' \to m \to m''$, where m' is a public module computing a constant function, m is a private module computing a one-one function and m'' is another public module computing an invertible one-one function. If we hide only a subset of the input attributes of m, m' should be hidden, thus $P = \{m''\}$. Similarly, if we hide only a subset of the output attributes of m, m'' should be hidden. Finally, if we hide a combination of input and output attributes, both m', m'' should be hidden and in that case $P = \phi$. □

6.5.2 The `Secure-View` Problem (Revisited)

The `Secure-View` optimization problem in general workflows is similar to the case of all-private workflows, with an additional cost due to hiding (privatization) of public modules: when a public module m_j is hidden, the solution incurs a cost $\texttt{c}(m_j)$. Following the notation of visible and hidden attributes, V and $\overline{V}$, we will denote the set of hidden public modules by $\overline{P}$. The total cost due to hidden public modules is $\texttt{c}(\overline{P}) = \sum_{m_j \in \overline{P}} \texttt{c}(m_j)$, and the total cost of a safe solution (V, P) is $\texttt{c}(\overline{V}) + \texttt{c}(\overline{P})$. The definition of the `Secure-View` problem, with cardinality and set constraints, naturally extends to this refined cost function and the goal is to find a safe solution with minimum cost. This generalizes the `Secure-View` problem for all-private workflows where $\overline{P} = \phi$ (and hence $\texttt{c}(\overline{P}) = 0$).

Complexity Results (details appear in Appendix A.4.9.2). In Section 6.4.3 we showed that the `Secure-View` problem has an $O(\log n)$-approximation in an all-private workflow even when the lists specifying cardinality requirements are $\Omega(n)$-long and when the workflow has arbitrary data sharing. But, we show (in Appendix A.4.13) by a reduction from the label-cover problem that the cardinality constraints version in general workflows is $\Omega(2^{\log^{1-\gamma} n})$-hard to approximate (for all constant $\gamma > 0$), and thus unlikely to have any polylogarithmic-approximation. In contrast, the approximation factor for the set constraints version remains the same and Theorem 6.20 still holds for general workflows by a simple modification to the proof. However, γ-bounded data sharing no longer give a constant factor approximation any more for a constant value of γ. By a reduction from the set-cover problem, we prove in Appendix A.4.11 that the problem is $\Omega(\log n)$-hard to approximate even when the workflow has no data sharing, and when the maximum size of the requirement lists and the individual cardinality requirements in them are bounded by 1.

6.6 Related Work

In literature, workflow privacy mostly has been studied in the context of access control. Fine-grained access control languages for provenance have been developed [37, 38, 135, 164], and a graph grammar approach for rewriting redaction policies over provenance [39]. The approach in [21] provides users with informative graph query results using *surrogates*, which give less sensitive versions of nodes/edges, and proposes a utility measure for the result. In [42], the authors discuss a framework to output a *partial* view of a workflow that conforms to a given set of access permissions on the connections between modules and data on input/output ports. The problem of ensuring the *lawful use* of data according to specified privacy policies has been considered in [83, 84]. The focus of the work is a policy language for specifying relationships among data and module sets, and their properties relevant to privacy. Although all these papers address workflow privacy, the privacy notions are somewhat informal and no guarantees on the quality of the solution are provided in terms of privacy and utility. Furthermore, our work is the first, to our knowledge, to address module privacy rather than data privacy.

Secure provenance for workflows has been studied in [28, 94, 120]. The goal is to ensure that provenance information has not been forged or corrupted, and a variety of cryptographic and trusted computing techniques are proposed. In contrast, we assume that provenance information has not been corrupted, and focus on ensuring module privacy.

In [127], the authors study information disclosure in data exchange, where given a set of public views, the goal is to decide if they reveal any information about a private view. This does not directly apply to our problem, where the private elements are the $(\mathbf{x}, m(\mathbf{x}))$ relations. For example, if all $\mathbf{x}$ values are shown without showing any of the $m(\mathbf{x})$ values for a module m, then information is revealed in their setting but not in our setting.[39]

Privacy-preserving data mining has received considerable attention (see surveys [6, 172]). The goal is to hide individual data attributes while retaining the suitability of data for mining patterns. For example, the technique of *anonymizing data* makes each record indistinguishable from a large enough set of other records in certain identifying attributes [7, 121, 163]. Privacy preserving approaches were studied for *social networks* [13, 40, 82, 112, 142], *auditing queries* [130, 133], networks and programming languages [89, 93, 147], and in other contexts.

Our notion of *standalone* module privacy is close to that of ℓ-diversity [121], in which the values of *non-sensitive attributes* are generalized so that, for every such generalization,

[39]In contrast, it can be shown that showing all $m(\mathbf{x})$ values while hiding the $\mathbf{x}$'s, may reveal information in our setting.

there are at least ℓ different values of *sensitive attributes*. We extend this work in two ways: First, we place modules (relations) in a network of modules, which significantly complicates the problem, Second, we analyze the complexity of attaining standalone as well as workflow privacy of modules.

Another widely used technique is that of *data perturbation* where some noise (usually random) is added to the the output of a query or to the underlying database. This technique is often used in *statistical databases*, where a query computes some aggregate function over the dataset [70] and the goal is to preserve the privacy of data elements. In contrast, in our setting the private elements are $(\mathbf{x}, m(\mathbf{x}))$ pairs for a private module m and the queries are select-project-join style queries over the provenance relation rather than aggregate queries.

Privacy in *statistical databases* is typically quantified using *differential privacy*, which requires that the output distribution is *almost* invariant to the inclusion of any particular record (see survey [71] and the references therein). Although this is an extremely strong notion of privacy, *no* deterministic algorithm can guarantee differential privacy. Since provenance is used to ensure reproducibility of experiments (and therefore data values must be accurate), adding random noise to provenance information may render it useless. Thus standard mechanisms for differential privacy are unsuitable for our purpose. Our approach of outputting a safe view allows the user to know the name of all data items and the exact values of data that is visible. The user also does not lose any utility in terms of *connections* in the workflow, and can infer exactly which module produced which visible data item or whether two visible data items depend on each other.

6.7 Conclusions

This chapter proposes the use of provenance views for preserving the privacy of module functionality in a workflow. Our model motivates a natural optimization problem, `Secure-View` , which seeks to identify the smallest amount of data that needs to be hidden so that the functionality of every module is kept private. We give algorithms and hardness results that characterize the complexity of the problem.

In our analysis, we assume that users have two sources of knowledge about module functionality: the module name (identity) and the visible part of the workflow relation. Module names are informative for public modules, but the information is lost once the module name is hidden/renamed. Names of private modules are non-informative, and users know only what is given in the workflow view. However, if users have some additional prior knowledge about the behavior of a private module, we may hide their identity by renaming them, and then run our algorithms.

Our work suggests several promising directions for future research. First, a finer privacy analysis may be possible if one knows what *kind* of prior knowledge the user has on a private module, e.g. the distribution of output values for a specific input value, or knowledge about the types and names of input/output attributes (certain integers may be illegal social security numbers, certain character sequences are more likely to represent gene sequences than others, etc). Our definitions and algorithms currently assume that all data values in an attribute domain are equally possible, so the effect of knowledge of a possibly non-uniform prior distribution on input/output values should be explored. Second, some additional sources of user knowledge on functionality of public modules (e.g. types of attributes and connection with other modules) may prohibit hiding their functionality using privatization (renaming), and we would like to explore alternatives to privatization to handle public modules. A third direction to explore is an alternative model of privacy. As previously mentioned, standard mechanisms to guarantee differential privacy (e.g. adding random noise to data values) do not seem to work for ensuring module privacy w.r.t. provenance queries, and new mechanisms suitable to our application have to be developed. Other natural directions for future research include considering *non-additive cost functions*, in which some attribute subsets are more useful than others, efficiently handling *infinite or very large domains* of attributes, and exploring alternate objective functions, such as *maximizing utility* of visible data instead of minimizing the cost of hidden data.

Chapter 7

A Propagation Model for Module Privacy

In the previous chapter, we introduced a model for module privacy in workflow provenance. In this chapter, we extend the work presented in the previous chapter by introducing a *propagation model* for module privacy in workflows that contain both proprietary modules with unknown functionality (private modules) and modules with known functionality (public modules).

7.1 Overview

This chapter focuses on privacy of module functionality, in particular in the general – and common – setting in which proprietary (*private*) modules are used in workflows which also contain non-proprietary (*public*) modules, whose functionality is assumed to be known by users. As an example of a public module, consider one which sorts its inputs. Even if the exact algorithm used by the module is not known by users (e.g. Merge sort vs Quick sort), given an input to the module a user can construct the output. In contrast, the functionality of private modules (i.e. what result will be output for a given input) is not known and should not be revealed by the visible provenance information. Examples of private modules include proprietary gene sequencing and medical diagnosis modules.

We use the same notion of Γ-privacy defined in Chapter 6. In Chapter 6, we showed that in a workflow with only private modules (an *all-private workflow*) the problem has a simple, elegant solution: If a set of hidden input/output data guarantees Γ-standalone-privacy for a private module, then if the module is placed in an all-private workflow where a superset of that data is hidden, then Γ-workflow-privacy is guaranteed for that module in the workflow. In other words, in an all-private workflow, hiding the union of the corresponding hidden data of the individual modules guarantees Γ-workflow-privacy for all of them. Unfortunately, this does not hold when the private module is placed in a workflow which contains public and private modules (a *public/private workflow*).

As an example, consider a private module m_2, which we assume is non-constant. Clearly, when executed in isolation as a *standalone* module, then either hiding all its inputs or hiding all its outputs over all executions guarantees privacy for the maximum privacy parameter Γ. However, suppose m_2 is embedded in a simple chain workflow $m_1 \longrightarrow m_2 \longrightarrow m_3$, where both m_1 and m_3 are public, equality modules. Then even if we hide *both* the input and output of m_2, their values can be retrieved from the input to m_1 and the output from m_3. Note that the same problem would arise if m_1 and m_3 were invertible functions, e.g. reformatting modules, a common case in practice.

In Chapter 6, we therefore explored *privatizing* public modules, i.e. hiding the names of carefully selected public modules so that their function is no longer known, and then hiding subsets of input/output data to ensure their Γ-privacy. Returning to the example above, if it were no longer known that m_1 was an equality module then hiding the input to m_2 (output of m_1) would be sufficient, under the assumption that the users have no other prior knowledge about m_1. Similarly, if m_3 was privatized then hiding the output of m_2 (input to m_3) would be sufficient.

Although privatization is a reasonable approach in some cases, there are many practical scenarios where it cannot be employed. For instance, when the workflow specification (the module names and connections) is known to the users, or when the identity of the privatized public module can be discovered through the structure of the workflow and the names or types of its inputs/outputs.

To overcome this problem we propose in this work an alternative novel solution, based on the *propagation of data hiding through public modules.* Returning to our example, if the input to m_2 were hidden then the input to m_1 would also be hidden, although the user would still know that m_1 were the equality function. Similarly, if the output of m_2 were hidden then the output of m_3 would also be hidden; again, the user would still know that m_3 was the equality function. While in this example things appear to be simple, several technically challenging issues must be addressed when employing such a propagation model in the general case: 1) whether to propagate hiding upward (e.g. to m_1) or downward (e.g. to m_3); 2) how far to propagate data hiding; and 3) which data of public modules must be hidden. Overall the goal is to guarantee that the functionality of private modules is not revealed while minimizing the amount of hidden data.

Our contributions. In this work we focus on *downward* propagation, for reasons that will be discussed in Section 7.2. *Using a downward propagation model, we show the following strong results*: For a special class of common workflows, *single(private)-predecessor workflows*, or simply *single-predecessor workflows* (which include the common tree and chain workflows), taking solutions for Γ-standalone-privacy of each private module (*safe subsets*) augmented with specially chosen input/output data of public modules in their *public closure* (up to a successor private module) that is rendered *upstream-downstream safe (UD-safe)* by the data hiding, and hiding the union of data in the augmented solutions for each private module will ensure Γ-workflow privacy for all private modules. We define these notions formally in Section 7.2 and go on to show that single-predecessor workflows is the largest class of workflows for which propagation of data hiding only within the public closure suffices.

Since data may have different *costs* in terms of hiding, and there may be many different safe subsets for private modules and UD-safe subsets for public modules, the next problem we address is finding a minimum cost solution – the *optimum view problem*. Using the result from above, we show that for single-predecessor workflows the optimum view problem may be solved by first identifying safe and UD-safe subsets for the private and public modules, resp., then assembling them together optimally. The complexity of identifying safe subsets for a private module was studied in [63] and the problem was shown to be NP-hard (EXP-time) in the number of module attributes. We show here that identifying UD-safe subsets for public modules is of similar complexity: Even deciding whether a given subset is UD-safe for a module is coNP-hard in the number of input/output data. We note however that this is not as negative as it might appear, since the number of inputs/outputs of individual modules is not high; furthermore, the computation may be performed as a pre-processing step with the cost being amortized over possibly many uses of the module in different workflows. In particular we show that, given the computed subsets, for chain and tree-shaped workflows, the optimum view problem has a polynomial time solution in the size of the workflow and the maximum number of safe/UD-safe subsets for a private/public modules. Furthermore, the algorithm can be applied to general single-predecessor workflows where the public closures have chain or tree shapes. In contrast, when the public closure has an arbitrary DAG shape, the problem becomes NP-hard (EXP-time) in the size of the public closure.

We then consider general workflows, and give a sufficient condition to ensure Γ-privacy that is not the trivial solution of hiding all data in the workflow. In contrast to single-predecessor workflows, hiding data within a public closure no longer suffices; data hiding must continue through other private modules to the entire downstream workflow. In return, the requirement from data hiding for public modules is somewhat weaker here: hiding must only ensure that the module is *downstream-safe* (D-safe), which typically involves fewer input/output data than upstream-downstream-safety (UD-safe).

Organization. Throughout this chapter we will use the workflow model given in Section 2.2 and the notions of standalone- and workflow-module privacy defined in Section 6.2. We have already discussed several related work in Section 6.6. The remainder of the chapter is organized as follows: Section 7.2 describes our propagation model, defines upstream-downstream-safety and single-predecessor workflows, and states the privacy theorem. Section 7.3 discusses the proof of the privacy theorem, and the necessity of the upstream-downstream-safety condition as well as the single-predecessor restriction. The optimization problem is studied in Section 7.4. We then discuss general public/private workflows in Section 7.3, before concluding in Section 7.6.

7.2 Privacy via propagation

Our goal here is to optimally choose attributes to hide in the common case of workflows that contain both private and public modules. We call them public/private workflows. We have seen in the previous section that when a set of hidden attributes guarantees Γ-standalone-privacy for a private module, then the same set of attributes can be used to guarantee Γ-workflow-privacy *in an all-private workflow*. Unfortunately this is no longer the case for general workflows. To see why, we revisit the following example mentioned in the introduction.

Example 7.1. Consider a private module m_2 implementing a one-one function with k boolean inputs and k boolean outputs. Hiding any $\log \Gamma$ input attributes guarantees Γ-standalone-privacy for m_2 even if all output attributes of m_2 are visible. However, if m_2 gets all its inputs from a public module m_1 that implements an equality function (in general, any one-one function) then for any value y and every input $\mathbf{x} = m_1(y)$ to m_2, $m_2(\mathbf{x})$ is revealed.

Similarly, hiding any $\log \Gamma$ output attributes of m_2 guarantees Γ-standalone-privacy for it, even if all input attributes of m_2 are visible. But if m_2 sends its outputs to a public module m_3 that also implements the equality function (in general, a one-one invertible function), and whose output attributes happen to be visible, then for any input $\mathbf{x}$ to m_2, $m_2(\mathbf{x})$ can be immediately inferred (using the inverse function of m_3). $\square$

One intuitive way to overcome this problem is to propagate the hiding of data through the problematic public modules, i.e., hide the attributes of public models that may disclose information about hidden attributes of private modules. To continue with the above example, if we choose to protect the privacy of m_2 by hiding $\log \Gamma$ of its input (resp. output) attributes, then we also need to propagate the hiding *upward* (resp. *downward*) to the public module supplying the input (receiving the output) and hide the corresponding input (output) attributes of the equality module m_1 (m_3).

Three main issues arise when employing such a propagation model: (1) upward vs. downward propagation; (2) recursive propagation; and (3) which attributes to hide. We discuss these issues next.

7.2.1 Upward vs. downward propagation

Whether or not propagation can be used depends on the safe subsets chosen for the private modules as well as the properties of the public modules. To see this, consider again Example 7.1, and assume now that public module m_1 computes some constant function (i.e. $\forall \mathbf{x}, m_1(\mathbf{x}) = \mathbf{a}$, for some constant $\mathbf{a}$). Then even upward propagation that hides *all* the input attributes of m_1 no longer preserves the Γ-workflow-privacy of m_2 w.r.t its hidden input attributes for any non-trivial value of $\Gamma > 1$. This is because it suffices to look at the (visible) output attributes of m_2 to know the value $m_2(\mathbf{x})$ for $x = \mathbf{a}$. More generally, we show in Appendix A.5.1 that if hiding a subset of input attributes gives Γ-standalone privacy for a standalone module m_2, then hiding the same subset of input attributes in the simple chain workflow ($m_1 \longrightarrow m_2$) may not give Γ-workflow-privacy for m_2 unless m_1 corresponds to an *onto function*[40].

This unfortunately is not very common for public modules (e.g. some output values like an invalid gene sequence may be well-known to be non-existent). In contrast, we will show below that when the privacy of a private module m_2 is achieved by *hiding output attributes only*, downward propagation that achieves the same privacy guarantees is possible without imposing any restrictions on the public modules. *We therefore focus in the rest of this chapter on safe subsets that contain only output attributes.*

Observe that such safe subsets always exist for all private modules – one can always hide all the output attributes. They may incur higher cost than that of an optimal subset of both input and output attributes, but, in terms of privacy, by hiding only output attributes one does not harm the maximum achievable privacy guarantee of the private module. In particular it is not hard to see that hiding all input attributes can give a maximum of Γ_1-workflow-privacy, where Γ_1 is the size of the range of the module. On the other hand hiding all output attributes can give a maximum of Γ_2-workflow-privacy, where Γ_2 is the size of the co-domain of the module, which can be much larger than the actual range.

[40]A function $f : D \to C$ is called *onto* or *surjective* if for every element $y \in C$, there is an element $x \in D$ such that $f(x) = y$.

7.2.2 Recursive propagation

Consider again Example 7.1, and assume now that public module m_3 sends its output to another public module m_4 that also implements an equality function (or a one-one invertible function). Even if the output of m_3 is hidden as described above, if the output of m_4 remains visible, the privacy of m_2 is again jeopardized since the output of m_3 can be inferred using the inverse function of m_4. We thus need to propagate the attribute hiding to m_4 as well. More generally, we will see below that we need to propagate the attributes recursively, through all adjacent public modules, until we reach another private module.

To formally define the *closure* of public modules to which attributes hiding needs to be propagated, we use the notion of an (un)directed *public path*. Intuitively, there is an (un)directed public path from a public module m_i to a public module m_j if we can reach m_j from m_i by an (un)directed path comprising only public modules. Recall that $A_i = I_i \cup O_i$ denotes the set of input and output attributes of module m_i.

Definition 7.2. *A public module m_1 has* a directed (resp. an undirected) public path *to a public module m_2 if there is a sequence of public modules $m_{i_1}, m_{i_2}, \cdots, m_{i_j}$ such that $m_{i_1} = m_1$, $m_{i_j} = m_2$, and for all $1 \leq k < j$, $O_{i_k} \cap I_{i_{k+1}} \neq \emptyset$ (resp. $A_{i_k} \cap A_{i_{k+1}} \neq \emptyset$).*

This notion naturally extends to module attributes. We say that an input attribute $a \in I_1$ of a public module m_1 has an (un)directed public path to a public module m_2 (resp. to its output attribute $b \in O_2$), if there is an (un)directed public path from m_1 to m_2. The set of public modules to which attribute hiding will be propagated can now be defined as follows.

Definition 7.3. *Given a private module m_i and a set of hidden output attributes $h_i \subseteq O_i$ of m_i, the* public-closure *$C(h_i)$ of m_i w.r.t. h_i, is the set of public modules consisting of (1) all public modules m_j s.t. $h_i \cap I_j \neq \emptyset$, and (2) all public modules to which there exists a undirected public path from the modules in $C(h_i)$.*

Example 7.4. We illustrate these notions using Figure 7.1. The public module m_4 has an undirected public path to the public module m_6 through the modules m_7 and m_3. For the private module m_2 and hidden output attributes $h_2 = \overline{V_2}$ that is a subset of $\{a_2, a_3, a_5\}$, the public closure $C(\overline{V_2}) = \{m_3, m_4, m_6, m_7\}$, whereas for $\overline{V_2} = \{a_4\}$ or $\{a_4, a_5\}$, $C(\overline{V_2}) =$

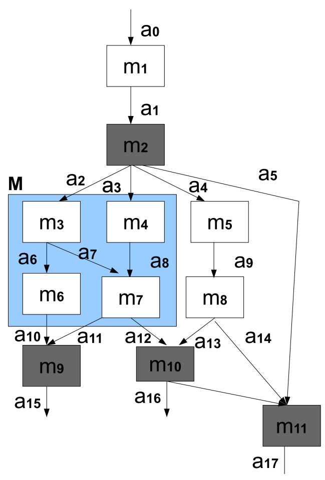


Figure 7.1: A single-predecessor workflow. White modules are public, grey are private; the box denotes the composite module M for $\overline{V_2} = \{a_2\}$.

$\{m_5, m_8\}$. In our subsequent analysis it will be convenient to view the public-closure as a virtual *composite module* that encapsulates the sub-workflow and behaves like it. For instance, the box in Figure 7.1 denotes the composite module M representing $C(\{a_2\})$, that has input attributes a_2, a_3, and output attributes a_{10}, a_{11} and a_{12}. $\square$

7.2.3 Selection of hidden attributes

In Example 7.1 it is fairly easy to see which attributes of m_1 or m_3 need to be hidden to preserve the privacy of m_2. For the general case, where the public modules are not as simple as equality functions, to determine which attributes of a given public module need to be hidden we use the notion of *upstream* and *downstream* safety. To define them we use the following notion of tuple equivalence w.r.t a given view (set of visible attributes). Recall that A denotes the set of all attributes in the workflow.

Definition 7.5. *Given two tuples* $\mathbf{x}$ *and* $\mathbf{y}$ *on a subset of attributes* $B \subseteq A$*, and a subset of visible attributes* $V \subseteq A$*, we say that* $\mathbf{x} \equiv_V \mathbf{y}$ *iff* $\pi_{V \cap B}(\mathbf{x}) = \pi_{V \cap B}(\mathbf{y})$.

Definition 7.6. *Given a subset of visible attributes* $V \subseteq A_i$ *of a public module* m_i*,* m_i *is called*

- downstream-safe w.r.t. V *if for any two equivalent input tuples* $\mathbf{x}, \mathbf{x}'$ *to* m_i *w.r.t.* V *their outputs are also equivalent:*

$$[\mathbf{x} \equiv_V \mathbf{x}'] \Rightarrow [m_i(\mathbf{x}) \equiv_V m_i(\mathbf{x}')],$$

- upstream-safe w.r.t. *V if for any two equivalent outputs $\mathbf{y}, \mathbf{y}'$ of m_i w.r.t. V all of their preimages are also equivalent:*

$$\left[(\mathbf{y} \equiv_V \mathbf{y}') \wedge (m_i(\mathbf{x}) = \mathbf{y}, m_i(\mathbf{x}') = \mathbf{y}')\right] \Rightarrow \left[\mathbf{x} \equiv_V \mathbf{x}'\right],$$

- upstream-downstream-safe (in short UD-safe) w.r.t. *V if it is both upstream-safe and downstream-safe w.r.t. V.*

Note that if $V = \emptyset$ (i.e. all attributes are hidden) then m_i is clearly UD-safe w.r.t to V. We call this the *trivial* UD-safe subset for m_i.

Example 7.7. Figure 7.2 shows some example module relations. For an (identity) module having relation R_1 in Figure 7.2a, two UD-safe visible subsets are $\{a_1, a_3\}$ and $\{a_2, a_4\}$. Note that $\{a_2, a_3\}$ is not a UD-safe subset: for tuples having the same values of a_2, say 0, the values of a_3 are not the same. For a module having relation R_2 in Figure 7.2b, a UD-safe visible subset is $\{a_1, a_3, a_4\}$, but there is no UD-safe subset that includes a_2. It can also be checked that the module m_1 in Figure 2.5b does not have any non-trivial UD-safe subset. □

a_1	a_2	a_3	a_4
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

(a) R_1

a_1	a_2	a_3	a_4
0	0	1	0
0	1	1	0
1	0	0	1
1	1	0	1

(b) R_2

Figure 7.2: UD-safe solutions for modules

The first question we attempt to answer is whether there is a theorem analogous to Theorem 6.12 that works in the presence of public modules. In particular, we will show that for a class of workflows called *single-predecessor workflows* one can construct a private solution for the whole workflow by taking safe standalone solutions for the private modules, and then ensuring the UD-safe properties of the public modules in the corresponding public-closure. Next we define this class of workflows:

Definition 7.8. *A workflow W is called a* single-predecessor workflow, *if*

1. *W has no data-sharing, i.e. for $m_i \neq m_j$, $I_i \cap I_j = \emptyset$, and,*

2. *for every public module m_j that belongs to a public-closure w.r.t some output attribute(s) of a private module m_i, m_i must be the unique private module that has a directed public path to m_j (i.e. m_i is the single private predecessor of m_j having a directed public path to it).*

Example 7.9. Again consider Figure 7.1 which shows a single-predecessor workflow. Modules m_3,m_4, m_6,m_7 have undirected public paths from $a_2 \in O_2$ (output attribute of m_2), whereas m_5,m_8 have a (un)directed public path from $a_4 \in O_2$; also m_1 is the unique private-predecessor of m_3,...,m_8. The public module m_1 does not have any private predecessor, but m_1 does not belong to the public-closure w.r.t the output attributes of any private module. $\square$

Single-predecessor workflows include common workflow structures such as chains and trees. Although they are more restrictive than general workflows, the above example illustrates that they can still capture fairly intricate workflow structures. We will focus first on single-predecessor workflows, then explain in Section 7.5 how general workflows can be handled.

Let M^+ be the set of indices for public modules ($M^+ = \{i : m_i \text{ is public }\}$) and M^- be the set of indices for private modules. Recall that I_i,O_i denote the subset of input and output attributes of module m_i and $A_i = I_i \cup O_i$; if $V_i \subseteq A_i$, then $\overline{V_i} = A_i \setminus V_i$.

Theorem 7.10. ***(Privacy Theorem for single-predecessor workflows)*** *Let W be a single-predecessor workflow. For a private module m_i in W, let V_i be a safe subset for Γ-standalone-privacy s.t. only output attributes of m_i are hidden (i.e. $\overline{V_i} \subseteq O_i$). Let $C(\overline{V_i})$ be the public-closure of m_i w.r.t. hidden attributes $\overline{V_i}$. Let H_i be a set of hidden attributes s.t. $\overline{V_i} \subseteq H_i \subseteq O_i \cup \bigcup_{k \in C(\overline{V_i})} A_k$ and where for every public module $m_j \in C(\overline{V_i})$, m_j is UD-safe w.r.t. $A_j \setminus H_i$. Then the workflow W is Γ-private w.r.t the set of visible attributes $V = A \setminus (\bigcup_{i \in M^-} H_i)$.*

In other words, in a single-predecessor workflow, taking solutions for the standalone private modules, expanding them to the public-closure of the modules, following the UD-safe safety requirements, then hiding the union of the attributes in these sets, guarantees Γ-workflow-privacy for all of the private modules.

The next proposition shows that single-predecessor workflows constitute in a sense the largest class of workflows for which such assembly is guaranteed to succeed.

Proposition 7.11. *There is a workflow W that is not a single-predecessor workflow (either because it has data sharing or because more (or fewer) than one such private-predecessor exists for some public module), and a private module m_i in W, where even hiding all output attributes O_i of m_i and all attributes A_j of the public modules $m_j \in C(O_i)$ does not give Γ-privacy for any $\Gamma > 1$.*

The next section is dedicated to proving these two results. In Section 7.4 we then show how this refined privacy theorem can be used to choose hidden attributes optimally in single-predecessor workflows. Solutions for workflows that are not single-predecessor are considered in Section 7.5.

7.3 Privacy for Single-Predecessor workflows

We start in Section 7.3.1 by proving Theorem 7.10 and explaining the role of the UD-safe requirement. Then, in Section 7.3.2, we prove Proposition 7.11 by illustrating the necessity of the restriction to single-predecessor workflows in Theorem 7.10.

7.3.1 Proof of the Privacy Theorem

Here we prove Theorem 7.10. To prove Γ-privacy, we need to show the existence of many possible outputs for each input to each public module, originating from the possible worlds of the workflow relation w.r.t. the visible attributes. First we present a crucial lemma which relates privacy of standalone modules to their privacy in a workflow when the conditions in Theorem 7.10 are satisfied.

Lemma 7.12. *Consider a single-predecessor workflow W; any private module m_i in W and any input $\mathbf{x} \in \pi_{I_i}(R)$; and any $\mathbf{y} \in \texttt{OUT}_{\mathbf{x},m_i}$ w.r.t. a set of visible attributes V_i. Given a set of hidden attributes $H_i \subseteq A$, such that (i) the hidden attributes $\overline{V_i} \subseteq H_i$, (ii) only output attributes from O_i are included in $\overline{V_i}$ (i.e. $\overline{V_i} \subseteq O_i$), and (iii) every module m_j in the public-closure $C(\overline{V_i})$ is UD-safe w.r.t. $A_j \setminus H_i$. Then $\mathbf{y} \in \texttt{OUT}_{\mathbf{x},W}$ w.r.t. visible attributes $V = A \setminus H_i$.*

First we show that the above lemma implies Theorem 7.10. Then we can focus on a single private module m_i, as given in the lemma, in the remainder of this subsection.

Proof of Theorem 7.10.

Proof. We first argue that if H_i satisfies the conditions in Theorem 7.10 then $H'_i = \bigcup_{\ell \in M^-} H_\ell$ satisfies the conditions in Lemma 7.12. Clearly (i) $\overline{V_i} \subseteq H_i \subseteq \bigcup_{\ell \in M^-} H_\ell = H'_i$; (ii) $\overline{V_i} \subseteq O_i$ is unchanged. Next we argue that the third requirement in the lemma, (iii) every module in the public-closure $C(\overline{V_i})$ is UD-safe w.r.t. H'_i, also holds.

To see (iii), observe that the Theorem 7.10 has an additional condition on H_i: $H_i \subseteq O_i \cup \bigcup_{j \in C(\overline{V_i})} A_j$. Since W is a single-predecessor workflow, for two private modules $m_i, m_\ell \in M^-, i \neq \ell$, the public closures $C(\overline{V_i}) \cap C(\overline{V_\ell}) = \emptyset$ (this follows directly from the definition of single-predecessor workflows). Further, since W is single-predecessor, W has no data-sharing by definition, so for any two modules m_i, m_j in W (public or private), the set of attributes $A_i \cap A_j = \emptyset$. Clearly, m_i being a private module, $m_i \notin C(\overline{V_\ell})$ and vice versa. Hence

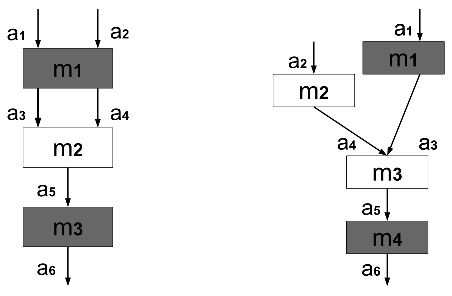
$$\Big(O_i \cup (\cup_{k \in C(\overline{V_i})} A_k)\Big) \cap \Big(O_\ell \cup (\cup_{k \in C(\overline{V_\ell})} A_k)\Big) = \emptyset.$$

In particular, for a public module $A_j \in C(V_i)$, and for any private module $m_\ell \in M^-, \ell \neq i$, $A_j \cap H_\ell = \emptyset$. Therefore, $A_j \setminus H'_i = A_j \setminus (\bigcup_{\ell \in M^-} H_\ell) = A_j \setminus H_i$. Since m_j is UD-safe w.r.t. $A_j \setminus H_i$ from the condition in the theorem, m_j is also UD-safe w.r.t. $A_j \setminus H'_i$. Hence H'_i satisfies the conditions stated in the lemma.

Now Theorem 7.10 also states that each private module m_i ($i \in M^-$) is Γ-standalone-private w.r.t. visible attributes V_i, i.e., $|\text{OUT}_{\mathbf{x},m_i}| \geq \Gamma$ for all input $\mathbf{x}$ to m_i (see Definition 6.3). From Lemma 7.12, using H'_i in place of H_i, this implies that for all input $\mathbf{x}$ to private modules m_i, $|\text{OUT}_{\mathbf{x},W}| \geq \Gamma$ w.r.t $V = A \setminus H'_i = A \setminus \bigcup_{\ell \in M^-} H_\ell$. From Definition 6.6, this implies that each private module m_i is Γ-workflow-private w.r.t. $V = A \setminus \bigcup_{i \in M^-} H_i$; equivalently W is Γ-private w.r.t. V. □

We can thus focus from now on a single private module m_i and only need to prove Lemma 7.12 for m_i. But before we do so, let us first discuss the necessity of the UD-safe assumption in Theorem 7.10.

Necessity of UD-safe condition. Example 7.1 suggests that the downward-safety condition is necessary and natural. Otherwise, if there is a subsequent equality module, the value of the hidden attributes $\overline{V_i} \subseteq O_i$ may be revealed. But it may not be obvious why we need the upward-safety condition, since we restrict the hidden attributes to be a subset of the output attributes O_i. The following proposition illustrates the necessity of having both upstream and downstream conditions in Theorem 7.10.



(a) Necessity of UD-safe condition (b) Necessity of single predecessor

Figure 7.3: White modules are public, grey are private.

Proposition 7.13. *There is a workflow W, a private module m_i, and a safe-subset V_i that guarantees Γ-standalone-privacy for m_i for some $\Gamma > 1$, such that satisfying only the downstream-safety condition for the public modules in $C(\overline{V_i})$ does not give Γ-workflow-privacy for m_i for* all $\Gamma > 1$.

In the proof of this proposition and in other places in this chapter we will consider the different possible worlds of the workflow view and focus on the behavior (input-to-output mapping) $\widehat{m}_i$ of the module m_i, as seen in these world. This may be different than its true behavior, recorded in the actual workflow relation R, and we will say that m_i is *redefined* as $\widehat{m}_i$ in the given world. Note that m_i and $\widehat{m}_i$, viewed as relations, agree on the visible attributes of the the view but may differ in the non visible ones.

Proof of Proposition 7.13.

Proof. Consider a chain workflow as given in Figure 7.3a, with three modules m_1, m_2, m_3 defined as follows. (i) $m_1(a_1, a_2) = (a_3 = a_1, a_4 = a_2)$, (ii) $a_5 = m_2(a_3, a_4) = a_3 \vee a_4$ (OR), (iii) $a_6 = m_3(a_5) = a_5$. m_1, m_3 are private whereas m_2 is public. All attributes take values in $\{0,1\}$. Clearly hiding output a_3 of m_1 gives Γ-standalone privacy for $\Gamma = 2$. Now suppose a_3 is hidden in the workflow. Since the m_2 is public (known to be OR function), a_5 must be hidden (downstream-safety condition). Otherwise from visible output a_5 and input a_4, some values of hidden input a_3 can be uniquely determined (eg. if $a_5 = 0, a_4 = 0$, then $a_3 = 0$ and if $a_5 = 1, a_4 = 0$, then $a_3 = 1$). On attributes $(a_1, a_2, \underline{a_3}, a_4, \underline{a_5}, a_6)$, the original relation R is shown in Table 7.1 (the hidden attributes and their values are underlined in the text and in grey in the table).

a_1	a_2	a_3	a_4	a_5	a_6
0	0	0	0	0	0
0	1	0	1	1	1
1	0	1	0	1	1
1	1	1	1	1	1

Table 7.1: Relation R for workflow given in Figure 7.3a

Let us first consider an input (0,0) to m_1. When a_3 is hidden, a possible candidate output y of input tuple $x = (0,0)$ to m_1 is $(\underline{1},0)$. So we need to have a possible world where m_1 is redefined as $\widehat{m}_1(0,0) = (1,0)$. To be consistent on the visible attributes, this forces us to redefine m_3 to $\widehat{m}_3$ where $\widehat{m}_3(1) = 0$; otherwise the row $(0,0,\underline{0},0,\underline{0},0)$ in R changes to $(0,0,\underline{1},0,\underline{1},1)$. This in turn forces us to define $\widehat{m}_1(1,0) = (0,0)$ and $\widehat{m}_3(0) = 1$. (This is because if we map $\widehat{m}_1(1,0)$ to any of $\{(1,0),(0,1),(1,1)\}$, either we have inconsistency on the visible attribute a_4, or $a_5 = 1$, and $\widehat{m}_3(1) = 0$, which gives a contradiction on the visible attribute $a_6 = 1$.)

Now consider the input (1,1) to m_1. For the sake of consistency on the visible attribute a_3, $\widehat{m}_1(1,1)$ can take value (1,1) or (0,1). But if $\widehat{m}_1(1,1) = (1,1)$ or (0,1), we have an inconsistency on the visible attribute a_6. For this input in the original relation R, $a_5 = a_6 = 1$. Due to the redefinition of $\widehat{m}_3(1) = 0$, we have inconsistency on a_6. But note that the downstream-safety condition has been satisfied so far by hiding a_3 and a_5. To have consistency on the visible attribute a_6 in the row $(1,1,\underline{1},1,\underline{1},1)$, we have to have $a_5 = 0$ (since $\widehat{m}_3(0) = 1$). The pre-image of $a_5 = 0$ is $a_3 = 0, a_4 = 0$, hence we have to redefine $\widehat{m}_1(1,1) = (\underline{0},0)$. But $(\underline{0},0)$ is not equivalent to original $m_1(1,1) = (\underline{1},1)$ w.r.t. the visible attribute a_4. So the only solution in this case for $\Gamma > 1$, assuming that we do not hide output a_6 of private module m_3, is to hide a_4, which makes the public module m_2 both upstream and downstream-safe. $\square$

This example also suggests that upstream-safety is needed only when a private module gets input from a module in the public-closure; the proof of Lemma 7.12 shows that this is indeed the case.

Proof of Lemma 7.12. The proof of Lemma 7.12 comprises two steps:

(Step-1) Consider the connected subgraph $C(\overline{V_i})$ as a single *composite* public module M, or equivalently assume that $C(\overline{V_i})$ contains a single public module. By the properties of single-predecessor workflows, M gets all its inputs from m_i, but can send its outputs to one or more private module, or to final output. Let I (resp. O) be the input (resp. output) attribute sets of M. In Figure 7.1, the box is M, $I = \{a_2, a_3\}$ and $O = \{a_{10}, a_{11}, a_{12}, a_{13}\}$. We argue that if M is UD-safe w.r.t. visible attributes $(I \cup O) \setminus H_i$, and the other conditions of Lemma 7.12 are satisfied, then m_i is workflow-private w.r.t. V.

(Step-2) We show that if every public module in the composite module $M = C(\overline{V_i})$ is UD-safe, then M is UD-safe. To continue with our example, in Figure 7.1, assuming that m_3, m_4, m_6, m_7 are UD-safe w.r.t. hidden attributes, we have to show that M is UD-safe.

The proof of Step-1 uses the following lemma (proved in Appendix A.5.2) which relates the actual image $\mathbf{z} = m_i(\mathbf{x})$ of an input $\mathbf{x}$ to m_i, with a candidate output $\mathbf{y} \in \texttt{OUT}_{\mathbf{x},m_i}$ of $\mathbf{x}$.

Lemma 7.14. *Let m_i be a standalone private module with relation R_i, let $\mathbf{x}$ be an input to m_i, and let V_i be a subset of visible attributes such that $\overline{V_i} \subseteq O_i$ (only output attributes are hidden). If $\mathbf{y} \in \texttt{OUT}_{\mathbf{x},m_i}$ then $\mathbf{y} \equiv_{V_i} \mathbf{z}$ where $\mathbf{z} = m_i(\mathbf{x})$.*

Proof of Step-1. The proof of Lemma 7.12 is involved even for the restricted scenario in Step-1, in which $C(\overline{V_i})$ contains a single public module. The proof is given in Appendix A.5.3, and we illustrate here the key ideas using a simple example of a chain workflow.

Example 7.15. Consider a chain workflow, for instance, the one given in Figure 7.3a with the relation in Table 7.1. Let $m_i = m_1$. Hiding its output a_3 gives Γ-standalone-privacy for $\Gamma = 2$. For input $\mathbf{x} = (0,0)$, with a_3 hidden, $\mathbf{y} = (\underline{1},0)$ is a possible output, whereas the original output for $\mathbf{x}$ is $\mathbf{z} = (\underline{0},0)$ (hidden attributes are underlined). Note that, as Lemma 7.14 says, $\mathbf{y}$ and $\mathbf{z}$ are equivalent on the visible attributes.

First consider the simpler case when m_3 does not exist, i.e. W contains only two modules and the column for a_6 does not exist in Table 7.1. As we mentioned before, when the composite public module does not have any private successor, we only need downstream-safety property of the modules in $C(\overline{V_i})$, in this case which comprises a single public module m_2. Then we define a possible world R' of R by redefining module m_1 to $\widehat{m}_1$ as follows. $\widehat{m}_1$ simply maps all pre-images of $\mathbf{y}$ to $\mathbf{z}$, and all pre-images of $\mathbf{z}$ to $\mathbf{y}$. In this case, both $\mathbf{y}, \mathbf{z}$ have single pre-image. So $\mathbf{x} = (0,0)$ gets mapped to $(\underline{1},0)$ and input $(1,0)$ gets mapped to $(0,0)$. Since m_2 has to be downstream-private, we also need to hide output a_5 of m_2. Finally R' is formed by the join of relations for $\widehat{m}_1$ and m_2. Note that the projection of R, R' will be the same on visible attributes a_1, a_2, a_4 (in R', the first row will be $(0,0,\underline{1},0,\underline{0})$ and the third row will be $(1,0,\underline{0},0,\underline{0})$). The proof of this simpler case borrows the idea presented in [63].

Next consider the more complicated case, when the modules in $C(\overline{V_i})$ have a private successor, in this case when the private module m_3 is present. We already argued in the proof of Proposition 7.13 that we also need to hide the input a_4 to ensure workflow privacy for $\Gamma > 1$. Let us now describe the proof strategy when a_4 is hidden. Here we consider the outputs of m_3 on input $\mathbf{y}, \mathbf{z}$; let $\mathbf{w_y} = m_3(\mathbf{y})$ and $\mathbf{w_z} = m_3(\mathbf{z})$. We redefine m_1 to $\widehat{m}_1$ as follows. For all input $\mathbf{u}$ to m_1 such that $\mathbf{u} \in m_1^{-1} m_2^{-1}(\mathbf{w_z})$ (resp. $\mathbf{u} \in m_1^{-1} m_2^{-1}(\mathbf{w_y})$), we define $\widehat{m}_1(\mathbf{u}) = \mathbf{y}$ (resp. $\widehat{m}_1(\mathbf{u}) = \mathbf{z}$). For $\widehat{m}_3$, we define, $\widehat{m}_3(\mathbf{w_y}) = m_3(\mathbf{w_z})$ and $\widehat{m}_3(\mathbf{w_z}) = m_3(\mathbf{w_y})$. By Lemma 7.14, $\mathbf{y} \equiv_V \mathbf{z}$, since m_2 is downstream-safe $\mathbf{w_y} \equiv_V \mathbf{w_z}$, since m_2 is also upstream-safe, for all input $\mathbf{u}$ to m_1 that are being redefined by $\widehat{m}_1$, their images under m_1 are equivalent w.r.t. V. In our example, $\mathbf{w_y} = m_3(1,0) = 1$, and $\mathbf{w_z} = m_3(0,0) = 0$. $m_1^{-1} m_2^{-1}(\mathbf{w_z}) = \{(0,0)\}$ and $m_1^{-1} m_2^{-1}(\mathbf{w_y}) = \{(0,1),(1,0),(1,1)\}$. So $\widehat{m}_1$ maps $(0,0)$ to $(1,0)$ and all of $\{(0,1),(1,0),(1,1)\}$ to $(0,0)$; $\widehat{m}_3$ maps 0 to 1 and 1 to 0. With these observations, it can be shown that the resulting relation R' formed by th join of $\widehat{m}_1, m_2, \widehat{m}_3$ will be a possible world of R. In this example, when a_3, a_4, a_5 are hidden, R' has the form in Table 7.2, and these observations can be verified.

Note that this relation R' has the same projection on visible attributes $\{a_1, a_2, a_6\}$ as R in Table 7.1 and the public module m_2 is unchanged. So R' is a possible world of R that maps $\mathbf{x} = (0,0)$ to $\mathbf{y} = (1,0)$ as desired, i.e. $\mathbf{y} \in \mathrm{OUT}_{\mathbf{x},W}$. □

The argument for more general single-predecessor workflows, like the one given in Figure 7.1 is more complex. Here a private module (like m_{11}) can get inputs from m_i

a_1	a_2	a_3	a_4	a_5	a_6
0	0	1	0	1	0
0	1	0	0	0	1
1	0	0	0	0	1
1	1	0	0	0	1

Table 7.2: Relation R', a possible world of the relation R for the workflow in Figure 7.3a w.r.t. $V = \{a_1, a_2, a_6\}$.

(in Figure 7.1, m_2), its public-closure $C(\overline{V_i})$ (in the figure, m_8), and also from the private successors of the modules in $C(\overline{V_i})$ (in the figure, m_{10}). In this case, the tuples $\mathbf{w_y}, \mathbf{w_z}$ are not well-defined, and redefining the private modules is more complex. In the full proof we give the formal argument using an *extended flipping function*, that selectively changes part of inputs and outputs of the private module based on their connection with the private module m_i considered in the lemma.

Proof of Step-2.

For Step-2 we prove the following lemma.

Lemma 7.16. *Let M be a composite module consisting of public modules. Let H be a subset of hidden attributes such that every public module m_j in M is UD-safe w.r.t. $A_j \setminus H$. Then M is UD-safe w.r.t. $(I \cup O) \setminus H$.*

Proof. (Sketch) The formal proof of this lemma is given in Appendix A.5.4. We sketch here the main ideas. To prove the lemma, we show that if every module in the public-closure is downstream-safe (resp. upstream-safe), then M is downstream-safe (resp. upstream-safe). For downstream-safety, we consider the modules in M in topological order, say $m_{i_1}, \cdots, m_{i_k}$ (in Figure 7.1, $k = 4$ and the modules in order may be m_3, m_6, m_4, m_7). Let M^j be the (partial) composite public module formed by the union of modules $m_{i_1}, \cdots, m_{i_j}$, and let I^j, O^j be its input and output (the attributes that are either from a module not in M^j to a module in M^j, or to a module not in M^j from a module in M^j. Clearly, $M^1 = \{m_{i_1}\}$ and $M^k = M$. Then by induction from $j = 1$ to k, we show that M^j is downstream-safe w.r.t. $(I^j \cup O^j) \setminus H$, if all of m_{i_ℓ}, $1 \le \ell \le j$ are downstream-safe w.r.t. $(I_{i_\ell} \cup O_{i_\ell}) \setminus H = A_{i_\ell} \setminus H$. For upstream-safety, we consider the modules in *reverse topological order*, $m_{i_k}, \cdots, m_{i_1}$, and give a similar argument by an induction on $j = k$ down to 1. $\square$

7.3.2 Necessity of Single-Predecessor Assumption

Here we prove Proposition 7.11, and show that the single-predecessor assumption in Theorem 7.10 is necessary. By Definition 7.8, a workflow W is *not* a single-predecessor workflow if one of the following holds: (i) if there is a public module m_j in W that belongs to a public-closure of a private module m_i but has no directed path from m_i, or, (ii) such a public module m_j has directed path from more than one private modules, or (iii) W has data sharing.

We now show an example for condition (i). Examples for the remaining conditions can be found in Appendix A.5.5.

Example 7.17. Consider the workflow W_a in Figure 7.3b. Here the public module m_2 belongs to the public-closure $C(\{a_3\})$ of m_1, but there is no directed public path from m_1 to m_2, thereby violating the condition of single-predecessor workflows (though there is no data sharing). Module functionality is as follows: (i) m_1 takes a_1 as input and produces $a_3 = m_1(a_1) = a_1$. (ii) m_2 takes a_2 as input and produces $a_4 = m_2(a_2) = a_2$. (iii) m_3 takes a_3, a_4 as input and produces $a_5 = m_3(a_3, a_4) = a3 \vee a_4$ (OR). (iv) m_4 takes a_5 as input and produces $a_6 = m_4(a_5) = a_5$. All attributes take values in $\{0,1\}$.

Clearly, hiding output $h_1 = \{a_3\}$ of m_1 gives 2-standalone privacy. For this hidden attribute, $H_1 \subseteq \{a_2, a_3, a_4, a_5\}$. We claim that hiding all of $\{a_2, a_3, a_4, a_5\}$ gives only trivial 1-workflow-privacy for m_1, although it satisfies the UD-safe condition of m_2, m_3. To see this, consider the relation R_a of all executions of W_a given in Table 7.3, where the hidden values are in Grey. The rows (tuples) here are numbered $r_1, \ldots, r_4$ for later reference.

	a_1	a_2	a_3	a_4	a_5	a_6
r_1	0	0	0	0	0	0
r_2	0	1	0	1	1	1
r_3	1	0	1	0	1	1
r_4	1	1	1	1	1	1

Table 7.3: Relation R_a for workflow W_a given in Figure 7.3b

When a_3 is hidden, a possible candidate output of input $a_1 = 0$ to m_1 is 1. So we need to have a possible world where m_1 is redefined as $\widehat{m}_1(0) = 1$. This would restrict a_3 to 1 whenever $a_1 = 0$. But note that whenever $a_3 = 1$, $a_5 = 1$, irrespective of the value of a_4 (m_3

is an OR function).

This affects the rows r_1 and r_2 in R. Both these rows must have $a_5 = 1$, however r_1 has $a_6 = 0$, and r_2 has $a_6 = 1$. But this is impossible since, whatever the new definition $\widehat{m}_4$ of private module m_4 is, it cannot map a_5 to both 0 and 1; $\widehat{m}_4$ must be a function and maintain the functional dependency $a_5 \rightarrow a_6$. Hence all possible worlds of R_a must map $\widehat{m}_1(0)$ to 0, and therefore $\Gamma = 1$. □ □

7.4 Optimization

Given a workflow W, represented by a relation R, and a privacy parameter Γ, we want to find a safe visible subset V with minimum cost s.t. all the modules in the workflow are Γ-workflow-private w.r.t. V. Recall that each attribute a in R has a cost c_a and ideally one would like to find a set V s.t. the cost of hidden attributes $c(\overline{V}) = \sum_{a \in \overline{V}} c_a$ is minimized. The difficulty however is that even for workflows that contain only private modules the problem is NP-hard (and in EXP-time) in the number of attributes in the relation R, which may be very large.

To avoid this exponential dependency on the overall number of attributes, we will use here Theorem 7.10 to assemble a cost efficient solution for the whole workflow out of solutions for the *individual modules*. As we show below, computing solutions for the individual modules may still take time exponential in the number of the module's attributes. But this number is much smaller than that of the whole workflow, and the computation may be performed as a pre-processing step with the cost being amortized over possibly many uses of the module in different workflows.

The optimization problem. Following Theorem 7.10, given a single-predecessor workflow W, our goal is thus to find a set V of minimal cost[41], such that $V = A \setminus (\bigcup_{i \in M^-} H_i)$ for some sets H_i that satisfy the requirements in the Theorem. Namely, such that there exist some safe subsets V_i for the private modules m_i in W where only output attributes of m_i are hidden, $\overline{V_i} \subseteq H_i \subseteq O_i \cup \bigcup_{k \in C(\overline{V_i})} A_k$, and for every public module $m_j \in C(\overline{V_i})$, m_j is UD-safe w.r.t. $A_j \setminus H_i$. We call this problem the **optimum-view** problem.

In Section 7.4.1 we discuss how the optimum-view problem can be solved in four

[41] i.e. where $c(\overline{V}) = \sum_{a \in \overline{V}} c_a$ is minimized

steps, which also allows us to study the complexity of the problem that arises from the respective steps. In Section 7.4.2 and Section 7.4.3, we discuss two of these four steps in more detail.

7.4.1 Optimization in Four Steps

For single-predecessor workflows, the optimum view problem is solved in four steps: (i) we find the safe solutions for standalone-privacy for individual private modules; (ii) we find the UD-safe solutions for individual public modules; (iii) we find the optimal hidden subset H_i for the public-closure of every private module m_i using the outputs of the first two steps; finally, (iv) we combine H_i-s to find the final optimal solution V. We next consider each of these steps. We use below M^- (resp. M^+) to denote the set of private (resp. public) modules m_i in W (or their indices i by overloading the notation).

First step: safe (standalone-private) solutions for individual private modules. First, we find the set of safe subsets for all private modules in M^-. For a module $m_i \in M^-$, we compute the set of subsets $\mathbf{S}_i = \{S_{i1}, \cdots, S_{ip_i}\}$, where $\overline{S_{i\ell}} = A_i \setminus S_{i\ell} \subseteq O_i$, and m_i is Γ-standalone-private w.r.t. each of the $S_{i\ell}$. Here p_i is the number of such safe subsets.

Recall from Theorem 7.10 that the choice of safe subset for m_i determines its public-closure (and consequently the possible H_i sets and the cost of the overall solution). It is thus not sufficient to consider only the safe subsets that have the minimum cost; we need to keep *all* safe subsets for m_i, to be examined by subsequent steps.

The complexity of finding safe subsets for individual private modules has been thoroughly studied in [63] under the name *standalone* `Secure-View` *problem*. It was shown that deciding whether a given visible subset V is safe for a private module is NP-hard in the number of attributes of the module; an exponential (again in the number of attributes of the module) lower bound on the communication complexity under different settings for this problem were also given. It was further shown that the set of *all* safe subsets for the module can be computed in time exponential in the number of attributes assuming constant domain size, which almost matches the lower bounds.

Although the lower and upper bounds are somewhat disappointing, as argued in [63], the number of attributes of an individual module is fairly small. The assumption of constant domain is reasonable for practical purposes, assuming that the integers and reals are represented in a fixed number of bits. In these cases the individual relations can be big, however this computation can be done only once as a pre-processing step and the cost can be amortized over possibly many uses of the module in different workflows. Expert knowledge (from the module designer) to decide the safe subsets can also be useful to further save time.

Second Step: UD-safe solutions for individual public modules. This step focuses on finding the set of all UD-safe solutions for the individual public modules. We denote the UD-safe solutions for a public module $m_j \in M^+$ by $UD-safe_j = \{U_{j1}, \cdots, U_{jp_j}\}$, where each UD-safe subset $U_{j\ell} \subseteq A_j$, and p_j denotes the number of UD-safe solutions for the public module m_j. We analyze the complexity of this step in detail in Section 7.4.2. We will see that even deciding whether a given subset is UD-safe for a module is coNP-hard in the number of attributes (and that the set of all such subsets can be computed in exponential time). However, similar to the argument given for the first step, this computation is a one-time procedure that can be done as a pre-processing step with its cost amortized over possibly many workflows where the module is used. In addition, we only need the UD-safe subsets for those public modules that belong to a public-closure w.r.t. some output attribute(s) of a private module, so computing the UD-safe subsets for only these modules suffices.

Third Step: optimal H_i for each private module. The third step aims to find a set H_i of hidden attributes, of minimum cost, for every private module $m_i \in M^-$. As per the theorem statement, this set H_i should satisfy the conditions: (a) $H_i \supseteq \overline{S_{i\ell}}$, for some $S_{i\ell} \in \mathbf{S}_i$, and (b) for this safe subset $S_{i\ell}$, every public module m_j in the closure $C(\overline{S_{i\ell}})$, there exists a UD-safe subset $U_{jq} \in UD-safe_j$ such that $U_{jq} = A_j \setminus H_i$. We will discuss solving this problem in Section 7.4.3, the motivation will be clear from the next step.

Fourth step: optimal V for the workflow. According to Theorem 7.10, $V = A \setminus \bigcup_{i \in M^-} H_i$ is a Γ-private solution for the workflow. Observe that finding the optimal (minimum cost) such solution V for single-predecessor workflows is straightforward, once the minimum cost H_i-s are found in the previous step: The proof of Lemma 7.12 shows that for any two private modules m_i, m_k, $H_i \cap H_k = \emptyset$. This implies that the visible subset V with the minimum cost $\overline{V} = A \setminus V = \bigcup_{i \in M^-} H_i$ can be obtained using the optimal hidden subsets H_i for individual private modules from Step 3.

As we mentioned above, the first step has been discussed in [63] and the fourth step is straightforward for single-predecessor workflows. We now focus on describing solutions to the second step (Section 7.4.2) and the third step (Section 7.4.3).

7.4.2 Solutions for Individual Modules

We show that verifying whether a standalone module m_j is upstream-downstream-safe (UD-safe) w.r.t. a given subset V is coNP-hard.

Theorem 7.18. *Given public module m_j with total number of attributes k, and a subset of attributes V, deciding whether m_j is UD-safe w.r.t V is coNP-hard in k.*

Proof. (Sketch) The reduction is from the UNSAT problem, where we are given n variables $x_1, \cdots, x_n$, and a 3NF formula $f(x_1, \cdots, x_n)$; the goal is to check whether f is *not* satisfiable. In our construction, m_i has $n+1$ inputs $x_1, \cdots, x_n$ and y, and the output is $z = m_i(x_1, \cdots, x_n, y) = f(x_1, \cdots, x_n) \vee y$ (OR). The set of visible attributes is $\{y, z\}$; so all of $x_1, \cdots, x_n$ are hidden. We claim that f is not satisfiable if and only if m_i is UD-safe w.r.t. V. The complete argument is given in Appendix A.5.6. $\square$

The same construction, with attributes y and z assigned cost zero and all other attributes assigned some higher constant cost, can be used to show that testing whether a safe subset with cost smaller than a given threshold exists is also coNP-hard. In Appendix A.5.7 we also show that the communication complexity to solve these problems is exponential in the number of attributes.

Regarding the upper bound, the trivial algorithm of going over all 2^k subsets $V \subseteq A_j$, and checking if V is UD-safe for m_j, can be done in EXP-time in k when the domain size is constant (see Appendix A.5.8 for details). Since the UD-safe property is *not monotone* w.r.t. further deletion of attributes, if V is UD-safe, its subsets may not be UD-safe.

Recall however that deleting the entire subset A_j (i.e. the empty subset $\emptyset \subseteq A_j$) is always UD-safe for m_j. So for practical purposes, when the public-closure (to be studied in the third step) for a private module involves a small number of attributes of the public modules in the closure, or if the attributes of those public modules have small cost c_a, the trivial solution ($V = \emptyset$ or $\overline{V} = A_j$) can be used.

7.4.3 Optimization: Single Private Module

Given a private module m_i, the set of safe subsets $\mathbf{S}_i$ of m_i, and the set of UD-safe subsets $UD - safe_j$ for all public modules m_j, the goal of this step is to find the hidden subset H_i with minimum cost $c(H_i)$such that $H_i \supseteq \overline{S_{i\ell}}$ for some $S_{i\ell} \in \mathbf{S}_i$ and for all $m_j \in C(\overline{S_{i\ell}})$, there is a $U_{jq} \in UD - safe_j$ such that $U_{jq} = A_j \setminus H_j$. We call this problem the **single-module** problem.

We show that, fortunately, for the important class of chain and tree workflows, this optimization problem is solvable in time polynomial in the number of modules n, number of attributes $|A|$, and the maximum number of sets in $\mathbf{S}_i$ and $UD - safe_j$, denoted by

$$L = \max_{m_i \in M^- \cup M^+} p_i$$

In contrast, the problem becomes NP-hard in n when the public-closure forms arbitrary subgraph, even when L is a constant and the number of attributes of the individual modules is bounded by a (small) constant.

Chain Workflows. Chain workflows are the simplest class of tree-shaped workflow, hence clearly any algorithm for trees will also work for chains. However, for simplicity we first explain how chains are processed, then extend to general trees. We start by proving the following.

Theorem 7.19. *The single-subset problem can be solved in PTIME (in n, $|A|$ and L) for chain workflows.*

Proof. Note that to obtain an algorithm of time polynomial in L, for a given module m_i, we can go over all choices of safe subsets $S_{i\ell} \in \mathbf{S}_i$ of m_i, compute the public-closure $C(\overline{S_{i\ell}})$, and choose a minimal cost subset $H_i = H_i(S_{i\ell})$ that satisfies the UD-safe properties of all modules in the public-closure. Then, output, among them, a subset having the minimum cost. Consequently, it suffices to explain how, given a safe subset $S_{i\ell} \in \mathbf{S}_i$, one can solve, in PTIME, the problem of finding a minimum cost hidden subset H_i that satisfies the UD-safe property of all modules in a subgraph formed by a given $C(\overline{S_{i\ell}})$.

To simplify notation, the given safe subset $S_{i\ell}$ will be denoted below by S_{i*}, the closure $C(\overline{S_{i\ell}})$ will be denoted by C_W, and the output hidden subset will be denoted by H.

Our PTIME algorithm employs dynamic programming to find the optimal H. First note that since C_W is the public-closure of output attributes for a chain workflow, C_W should be a chain itself. Let the modules in C_W be renumbered as $m_1, \cdots, m_k$ in order. Now we solve the problem by dynamic programming as follows. Let Q be an $k \times L$ two-dimensional array, where $Q[j,\ell]$ denotes the cost of minimum cost hidden subset $H^{j\ell}$ that satisfies the UD-safe condition for all public modules m_1 to m_j and $A_j \setminus H^{j\ell} = U_{j\ell}$ (here $j \leq k$, $\ell \leq p_j \leq L$, and A_j is the attribute set of m_j); the actual solution can be stored easily by standard argument.

The initialization step is , for $1 \leq \ell \leq p_1$,

$$\begin{aligned} Q[1,\ell] &= c(\overline{U_{1,\ell}}) \quad \text{if } \overline{U_{1,\ell}} \supseteq \overline{S_{i*}} \\ &= \infty \quad \text{otherwise} \end{aligned}$$

Recall that for a chain, $O_{j-1} = I_j$, for $j = 2$ to k. Then for $j = 2$ to k, $\ell = 1$ to p_j,

$$\begin{aligned} Q[j,\ell] &= \infty \quad \text{if there is no } 1 \leq q \leq p_{j-1} \\ &\qquad \text{such that } \overline{U_{j-1,q}} \cap O_{j-1} = \overline{U_{j,\ell}} \cap I_j \\ &= c(O_j \cap \overline{U_{j\ell}}) + \min_q Q[j-1,q] \\ &\qquad \text{where the minimum is taken over all such } q \end{aligned}$$

It is interesting to note that such a q always exists at least for one $\ell \leq p_j$: while defining UD-safe subsets in Definition 7.6, we discussed that any public module m_j is UD-safe when its entire attribute set A_j is hidden. Hence $\emptyset \in UD-safe_{j-1}$ and $\emptyset \in UD-safe_j$, and the respective complement subsets are A_{j-1} and A_j, which will make the equality check true (for a chain $O_{j-1} = I_j$).

The following lemma (whose proof is given in Appendix A.5.9) shows that $Q[j,\ell]$ correctly stores the desired value. Then the optimal solution H has cost $\min_{1 \leq \ell \leq p_k} Q[k,\ell]$; the corresponding solution H can be found by standard procedure.

Lemma 7.20. *For $1 \leq j \leq k$, the entry $Q[j,\ell]$, $1 \leq \ell \leq p_j$, stores the minimum cost of the hidden attributes $\cup_{x=1}^{j} A_x \supseteq H^{j\ell} \supseteq \overline{S_{i*}}$ such that $A_j \setminus H^{j\ell} = U_{j\ell}$, and every module $m_x, 1 \leq x \leq j$ in the chain is UD-safe w.r.t. $A_x \setminus H^{j\ell}$.*

This conclues our proof. □

Observe that, more generally, the algorithm may also be used for arbitrary non-chain workflows, if the public-closures of the safe subsets for private modules have chain shape. This observation also applies to the following discussion on tree workflows.

Tree Workflows. Now consider tree-shaped workflows, where every module in the workflow has at most one immediate predecessor (for all $m_i \in W$, if $I_i \cap O_j \neq \emptyset$ and $I_i \cap O_k \neq \emptyset$, then $j = k$), but a module can have one or more immediate successors.

The treatment of tree-shaped workflows is similar to what we have seen above for chains. Observe that, here again, since C_W is the public-closure of output attributes for a tree-shaped workflow, C_W will be a collection of trees all rooted at m_i. As for the case of chains, the processing of the public closure is based on dynamic-programming. The key difference is that the modules in the tree are processed bottom up (rather than top down as in what we have seen above) to handle branching. The details of the algorithm are given in Appendix A.5.10, proving the following theorem.

Theorem 7.21. *The single-subset problem can be solved in PTIME (in n, $|A|$ and L) for tree-shaped workflows.*

Public-closure of arbitrary shape. Finally, for public-closure with arbitrary shape we can show the following.

Theorem 7.22. *The problem of testing whether the single-subset problem has a solution with cost smaller than a given bound is NP-complete when the public-closure forms an arbitrary subgraph. This is the case even when both the number of attributes and the number of safe and UD-safe subsets of the individual modules is bounded by a (small) constant.*

The hardness proof works by a reduction from 3SAT and is given in Appendix A.5.11. The NP algorithm simply guesses a set of attributes and checks whether it forms a legal solution and has cost lower than the given bound. A corresponding EXPtime algorithm that iterates over all subsets can be used to find the optimal solution.

The NP-completeness here is in n, the number of modules in the public closure. We note however that in practice the number of public modules that process the output on an individual private module is typically not that high. So the obtained solution to the

optimum-view problem is still better than the naive one, which is exponential in the size of the *full* workflow.

7.5 General Workflows

The previous sections focused on single-predecessor workflows. In particular we presented a privacy theorem for such workflows and studied optimization w.r.t. this theorem. The following two observations highlight how this privacy theorem can be extended to general workflows. For the sake of brevity, the discussion is informal; full details are given in Appendix A.5.12.

Need for propagation through private modules All examples in the previous sections that show the necessity of the single-predecessor assumption had another private module m_k as a successor of the private module m_i being considered. For instance, in Example 7.17, $m_i = m_1$ and $m_k = m_4$. If we had continued hiding output attributes of m_4 in Example 7.17, we could obtain the required possible worlds leading to a non-trivial privacy guarantee $\Gamma > 1$. This implies that for general workflows, the propagation of attribute hiding should continue outside the public closure and through the descendant private modules.

D-safe suffices (instead of UD-safe) The proof of Lemma 7.12 shows that the UD-safe property of modules in the public-closure is needed only when some public-module in the public-closure has a private successor whose output attributes are visible. If all modules in the public closure have no such private successor, then a downstream-safety property (called the **D-safe property**) is sufficient. More generally, if attribute hiding is propagated through private modules (as discussed above), then it suffices to require the hidden attributes to satisfy the D-safe property rather than the stronger UD-safe property.

The intuition from the above two observations is formalized in a *privacy theorem for general workflows*, analogous to Theorem 7.10. First, instead of public-closure, it uses *downward-closure*: for a private module m_i, and a set of hidden attributes h_i, the downward-closure $D(h_i)$ consists of all modules (public or private) m_j, that are reachable from m_i by a directed path. Second, instead of requiring the sets H_i of hidden attributes to ensure UD-safe, it requires them to only ensure D-safe.

The proof of the revised theorem follows lines similar to that of Theorem 7.10, with an added complication due to the fact that, unlike in the previous case, here the H_i subsets are no longer guaranteed to be disjoint. This is resolved by proving that D-safe subsets are closed under union, allowing for the (possibly overlapping) H_i subsets computed for the individual private modules to be assembled together.

The hardness results from the previous section transfer to the case of general workflow. Since H_i-s in this case may be overlapping, the union of optimal solutions H_i for individual modules m_i may not give the optimal solution for the workflow, and whether a non-trivial approximation exists is an interesting open problem.

To conclude the discussion, note that for single-predecessor workflows, we now have two options to ensure workflow-privacy: (i) by considering public-closures and ensuring UD-safe properties for their modules (following the privacy theorem for single-predecessor workflows); or (ii) by considering downward-closures and ensuring D-safe properties for their modules (following the privacy theorem for general workflows). Observe that these two options are incomparable: Satisfying UD-safe properties may require hiding more attributes compared to what is needed for satisfying D-safe properties. On the other hand, the downward-closure includes more modules than the public-closure (for instance the reachable private modules), and additional attributes need to be hidden to satisfy their D-safe properties. One could therefore run both algorithms, and choose the lower cost solution.

7.6 Conclusion

In this chapter, we addressed the problem of preserving module privacy in public/private workflows (called workflow-privacy), by providing a view of provenance information in which the input to output mapping of private modules remains hidden. As several examples in this chapter show, the workflow-privacy of a module critically depends on the structure (connection patterns) of the workflow, the behavior/functionality of other modules in the workflow, and the selection of hidden attributes. We show that for an important class of workflows called single-predecessor workflows, workflow-privacy can be achieved via *propagation* through public modules only, provided we maintain an invariant on the propagating modules called the UD-safe property. On the other hand, for general workflows, we show that even though propagation through both public and private modules is necessary, a weaker invariant (called the D-safe property) on the propagating modules suffices. We also study related optimization problems.

Several interesting future research directions related to the application of differential privacy were discussed in Section 6.6. Another interesting problem is to develop PTIME approximation algorithms for module privacy (that can handle non-monotonicity of UD-safe and D-safe subsets) in single-predecessor and general workflows.

Chapter 8

Conclusions

In this dissertation we have studied connections between privacy and uncertainty in two main directions: how a succinct representation of provenance can help propagate uncertainty from source to output and vice versa (Chapters 3 to 5), and (ii) how uncertainty can help enable provenance to be revealed while hiding associated private information (Chapters 6 and 7).

Chapters 3 and 4 focus on computing uncertainty in the output given uncertain source data. In particular, we considered query evaluation in probabilistic databases: given a query q and a probabilistic database I, compute the probabilities of the answers in $q(I)$. Here our two main goals were (i) to compute the (exact or approximate) probability distribution of the answers efficiently in poly-time, and (ii) to identify the classes of queries q (or, query-instance pairs $\langle q, I\rangle$) for which such efficient computation is possible. This problem reduces to the computation of probabilities of boolean expressions given the probabilities of its constituent variables. Therefore, in both these chapters, we investigated boolean provenances resulting from query evaluation.

In Chapter 3, we proposed the instance-by-instance approach that considers both the query q and the given database instance I, in contrast to the widely-used approach of only considering the given query. We proposed a novel characterization and efficient algorithms to decide whether the boolean provenances of the answers in $q(I)$ are read-once, which allows us to efficiently compute the probabilities even for the "unsafe" queries for which computing the exact probability is #P-hard in general. However, the read-once property of boolean provenances is not a necessary condition for poly-time probability computation. In fact, it has been shown that other well-known knowledge-compilation techniques like *BDD*, *d-DNNF*, etc. can explain poly-time computation of answer probabilities [101, 137]. Since the computation of provenance does not have much overhead while the query is evaluated, an exact characterization of boolean provenances that are amenable to poly-time computation is of both theoretical and practical interest.

In Chapter 4, we expanded the class of poly-time computable queries by including difference operations. Difference operations are common in practice, but to the best of our knowledge, they have not been considered in this context. We showed that the computation of exact probability is #P-hard even in very restricted scenarios for queries with difference; moreover, unlike positive queries, even approximating these probabilities is computationally hard. On the positive side, we showed that for a class of queries (and a class of boolean provenances in the instance-by-instance approach), it is indeed possible to approximate the probabilities in poly-time. Our work is a first step toward understanding the complexity of queries with difference operations, and a deeper understanding of these queries will be an important direction to pursue in the future. As more general research directions, one can explore other classes of queries (*e.g.*, recursive datalog queries) and uncertain inputs (*e.g.*, databases allowing correlations in source tuples, semistructured or unstructured data) that have interesting practical applications.

In Chapter 5 we studied tracing errors in the output to find possible erroneous inputs, and thereby to refine the input to improve the quality of the output. We studied this problem in the context of dictionary refinement in information extraction. Many of the rules in a rule-based information extraction system can be abstracted as operations in relational queries, and therefore, the outputs of the system can be associated with boolean provenances in terms of the dictionary entries used in the system. We proposed solutions to address two main challenges in this problem: (i) handling incomplete and sparse labeled data, and (ii) selection of dictionary entries that remove as many false positives as possible without removing too many true positives. We also supported our theoretical results by an extensive empirical evaluation using real-world information extraction system and extractors.

There are numerous interesting future directions in this area. For example, an important problem is to develop techniques for adaptively labeling such that a high quality system can be built with only a small labeled dataset. This is more important for informal domains such as Twitter, Facebook, Youtube and Flickr that are increasingly receiving attention from millions of web users today (as opposed to formal domains like news articles where several syntactic features like proper punctuation and capitalization in the text are available). In addition, the provenance-based framework, models and algorithms in our work on information extraction can be useful in the context of recommendation systems or automated question-answering systems. Given a set of outputs from these systems, where some outputs are correct while others are erroneous, a natural goal is to find the primary causes or sources of these errors; clearly, this is closely related to the objective in our work.

In Chapters 6 and 7, we studied publishing privacy-aware provenance information by introducing uncertainty in the provenance. In Chapter 6, we proposed a formal model for module privacy in the context of workflow provenance by hiding partial provenance information (selected data values over all executions). We showed that private solutions for individual modules can be composed to guarantee their privacy when they are part of a workflow where each module interacts with many other modules. Since hiding provenance information also has a cost in terms of the loss of utility to the user, we thoroughly studied the complexity of finding the minimum amount of provenance that needs to be hidden to guarantee a certain privacy level.

Then in Chapter 7 we took a closer look at module privacy in the presence of public modules with known functionality. The composability property in the previous chapter does not hold any more and "hiding" the names of the modules by privatization may not work in practice. We proposed a propagation model, where the requirement of data hiding is propagated through public modules. We showed that another composability property holds under certain restrictions in this case. We also studied the corresponding optimization problem.

There are other privacy concerns that we propose as interesting future directions: Revealing the sequence of executed modules, even if the entire experimental data is withheld, can pose *structural privacy* threats for workflow provenance. For example, the results of a set of diagnostic tests conducted on a patient may be concealed, but even knowing that the tests were performed reveals information about the patient's medical condition. We can try to find suitable provenance queries and formalize structural privacy under such queries. Even for well-studied data privacy, we need to formalize the notion of privacy and utility with respect to workflow provenance where most of the data values are correlated.

Appendix A

Additional Details and Proofs

A.1 Proofs from Chapter 3

A.1.1 Example of Non-Read-Once Boolean Provenances Allowing Poly-Time Computation

The following example shows that there exists a query Q and a probabilistic database instance D such that the expression E for evaluation of query Q on database D is not read-once but still the probability of E being true can be computed in poly-time.

Example A.1. The database D has three tables $R(A), S(A,B), T(B)$. Table S has n tuples. The tuple in the $2i-1$-th row has tuple (a_i, b_i) and the tuple in the $2i$-th row has tuple (a_{i+1}, b_i). We suppose that S is a *deterministic* relation, i.e. all the tuples is S belong to S with probability one and are annotated with true. If $n = 2k$ then table R has $k+1$ tuples, if $n = 2k-1$ then R has k tuple. The tuple in row j of R is a_j and is annotated by x_{2j-1}. Table T has k tuples, where $n = 2k-1$ or $n = 2k$: the tuple in row j is b_j and is annotated by x_{2j}. It can be verified that the expression E_n annotating the answer to the query $Q() = R(A), S(A,B), T(B)$ is

$$E_n = x_1x_2 + x_2x_3 + \ldots x_{n-1}x_n + x_nx_{n+1}.$$

which is non-read-once for all $n \geq 3$. An example with $n = 3$ is given in Figure A.1.

Next we show that $P_n = P(E_n)$ can be computed in poly-time in n by dynamic programming. Note that P_1 can be computed in $O(1)$ time. Suppose for all $\ell < n$, P_ℓ is

$$S = \begin{array}{|c||c|}\hline a_1 & x_1 \\ \hline a_2 & x_3 \\ \hline\end{array} \qquad R = \begin{array}{|cc||c|}\hline a_1 & b_1 & z_1 = 1_{\mathbb{B}} \\ \hline a_2 & b_1 & z_2 = 1_{\mathbb{B}} \\ \hline a_2 & b_2 & z_3 = 1_{\mathbb{B}} \\ \hline\end{array} \qquad T = \begin{array}{|c||c|}\hline b_1 & x_2 \\ \hline b_2 & x_4 \\ \hline\end{array}$$

Figure A.1: Illustration with $n = 3$, $E_3 = x_1x_2 + x_2x_3 + x_3x_4$.

computed and stored in an array. Then

$$\begin{aligned} P_n &= P(x_1x_2 + \ldots + x_{n-2}x_{n-1} + x_{n-1}x_n + x_nx_{n+1}) \\ &= P(x_1x_2 + \ldots + x_{n-2}x_{n-1} + x_{n-1}x_n) + P(x_nx_{n+1}) \\ &\quad - P(x_nx_{n+1}[x_1x_2 + \ldots + x_{n-2}x_{n-1} + x_{n-1}x_n]) \\ &= P_{n-1} + P(x_nx_{n+1}) - P(x_nx_{n+1}[x_1x_2 + \ldots + x_{n-2}x_{n-1} + x_{n-1}x_n]) \qquad \text{(A.1)} \end{aligned}$$

Observe that: $P(x_nx_{n+1}[x_1x_2 + \ldots + x_{n-2}x_{n-1} + x_{n-1}x_n]) = P(x_{n+1})P(x_n[x_1x_2 + \ldots + x_{n-2}x_{n-1} + x_{n-1}x_n]) = P(x_{n+1})P(x_1x_2x_n + \ldots + x_{n-2}x_{n-1}x_n + x_{n-1}x_n)$ (by idempotency) $= P(x_{n+1})P(x_1x_2x_n + \ldots + x_{n-3}x_{n-2}x_n + x_{n-1}x_n)$ (by absorption) $= P(x_{n+1})P(x_n)P(x_1x_2 + \ldots + x_{n-3}x_{n-2} + x_{n-1}) = P(x_nx_{n+1})[P(x_1x_2 + \ldots + x_{n-3}x_{n-2}) + P(x_{n-1})]$. Therefore,

$$P(x_nx_{n+1}[x_1x_2 + \ldots + x_{n-2}x_{n-1} + x_{n-1}x_n]) = P(x_nx_{n+1})[P_{n-3} + P(x_{n-1})] \qquad \text{(A.2)}$$

From (A.1) and (A.2), $P_n = P_{n-1} + P(x_nx_{n+1}) - P(x_nx_{n+1})[P_{n-3} + P(x_{n-1})] = P_{n-1} + P(x_nx_{n+1})[1 - P_{n-3} - P(x_{n-1})]$ Since the variables x_i-s are independent, $P(x_nx_{n+1}) = P(x_n)P(x_{n+1})$, and while computing P_n, P_{n-1} and P_{n-3} are already available. Hence P_n can be computed in linear time.

$\square$

A.1.2 Proofs from Section 3.3

Proof of Lemma 3.12.

LEMMA 3.12. *Algorithm* COMPCOTABLE *adds an edge (x, y) to G_C if and only if x, y together appear in some implicant in f_{IDNF} and the tables containing x, y are adjacent in G_T.*

Proof. Suppose two variables x,y belong to the same implicant in f_{IDNF}, and their tables are adjacent in G_T. Then by Lemma 3.10, there is a $\cdot$-node $u \in \texttt{lca}(x,y)$, and $x \in \texttt{Var}(v_1), y \in \texttt{Var}(v_2)$ for two distinct successors v_1,v_2 of u. When the algorithm processes the node u, if an edge between x,y is not added in a previous step, the edge will be added. This shows the completeness of algorithm CompCoTable.

Now we show the soundness of the algorithm. Consider two variables x,y such that either the tables containing them are not adjacent in G_T or they do not belong together in any of the implicants in f_{IDNF}. If the tables containing x,y are not adjacent in G_T, clearly, the algorithm never adds an edge between them – so let us consider the case when x,y do not belong to the same implicant in f_{IDNF}. Then by Lemma 3.10, there is no $\cdot$-node $u \in \texttt{lca}(x,y)$.

Consider any iteration of the algorithm and consider that a node u is processed by the algorithm in this iteration. If u is a $+$-node or if either $x \notin \texttt{Var}(u)$ or $y \notin \texttt{Var}(u)$, again no edge is added between x,y. So assume that, u is a $\cdot$-node and $x,y \in \texttt{Var}(u)$. Then u is a common ancestor of x and y. But since $u \notin \texttt{lca}(x,y)$, by definition of least common ancestor set, there is a successor v of u such that v is an ancestor of both x,y and therefore, $x,y \in \texttt{Var}(v)$. However, by Corollary 3.11, since x or y cannot belong to two distinct successors of node u, node v must be the unique successor of u such that $x,y \in \texttt{Var}(v)$. Since CompCoTable only joins variables from two distinct children, no edge will be added between x and y in G_C. □

A.1.3 Time Complexity of CompCoTable

First we prove the following two lemmas bounding the number of times any given pair of variables x,y are considered by the algorithm. The first lemma shows that the variables x,y are considered by algorithm CompCoTable to add an edge between them in G_{co} only when they together appear in an implicant in f_{IDNF}, i.e. only if the edge actually should exist in G_{co}.

Lemma A.2. *Consider any two variables x,y and a $\cdot$-node u. If x,y do not appear together in an implicant in f_{IDNF}, x,y do not belong to the variable sets $\texttt{Var}(v_1), \texttt{Var}(v_2)$ for two distinct successors v_1,v_2 of u.*

Proof. This easily follows from Lemma 3.10 which says that if x,y do not appear together in an implicant in f_{IDNF}, then there is no $\cdot$-node in $\texttt{lca}(x,y)$. So for every $\cdot$-node u, either (i) one of x and $y \notin \texttt{Var}(u)$, or, (ii) there is a unique successor v of u which is a common ancestor of x,y, i.e. both $x,y \in \texttt{Var}(v)$ (uniqueness follows from Corollary 3.11). $\square$

The second lemma bounds the number of times a pair x,y is considered by the algorithm to add an edge between them.

Lemma A.3. *Suppose $x,y \in \texttt{Var}(f)$ be such that they together appear in an implicant f_{IDNF}. Then algorithm* CompCoTable *considers x,y in Step 10 to add an edge between them maximum β_H times, where β_H is the width of the provenance DAG H.*

Proof. Note that the check in Step 10 is performed only when the current node u is a $\cdot$-node. Consider any $\cdot$-node u. (i) if either x or y is not in $\texttt{Var}(u)$, clearly, x,y are not checked in this step, otherwise, (ii) if both $x,y \in \texttt{Var}(u)$, and $x,y \in \texttt{Var}(v)$ for a unique child v of u, then also x,y are not checked at this step, otherwise, (iii) if u joins x,y, i.e., $x \in \texttt{Var}(v_1)$, $y \in \texttt{Var}(v_2)$ for two distinct children v_1,v_2 of u, then only x,y are considered by the algorithm in Step 10. (and after this node u is processed, both x,y appear in $\texttt{Var}(u)$).

However, since the query does not have any self-joins, the only time two variables x,y appear in two distinct successors of a $\cdot$-node u when the query plan joins a subset of tables containing the table for x with a subset of tables containing the table for y. So the pair x,y is multiplied at a unique layer of H, and the total number of times they are multiplied cannot exceed the total number of nodes in the layer which is at most the width β_H of the DAG H. $\square$

Now we complete the running time analysis of algorithm CompCoTable.

Lemma A.4. *Given the table-adjacency graph G_T and input query plan H, algorithm* CompCoTable *can be implemented in time $O(\beta_H m_{co} + n m_H)$ time, where m_{co} is the number of edges in the co-occurrence graph, m_H is the number of edges in the DAG H, β_H is the width of the DAG H and $n = |\texttt{Var}(f)|$.*

Proof. Initialization step can be done in $O(n)$ time. The topological sort can be done in $O(m_H + |V(H)|)$ time by any standard algorithm.

At every node $u \in V(H)$, to compute set $\texttt{Var}(u)$, the algorithm scans $O(d_u)$ successors of u, where $d_u =$ the outdegree of node u in H. Although by Corollary 3.11, for every two distinct children v_1, v_2 of a $\cdot$-node u, $\texttt{Var}(v_1) \cap \texttt{Var}(v_2) = \phi$, they may have some overlap when u is a $+$-node, and here the algorithm incurs an $O(nm_H)$ cost total as follows: (i) create an n-length Boolean array for u initialized to all zero, (ii) scan $\texttt{Var}(v)$ list of very successor v of u, for a variable $x \in \texttt{Var}(v)$, if the entry for x in the Boolean array is false mark it as true, (iii) finally scan the Boolean array again to collect the variables marked as true for variables in $\texttt{Var}(u)$. At every node $u \in V(H)$, the algorithm spends $O(nd_u)$ time, where $d_u =$ the outdegree of node u in H. Hence the total time across all nodes = $\sum_{u \in V(H)} O(nd_u)$ = $O(nm_H)$.

Every check in Step 10, i.e., whether an edge (x, y) has already been added and whether the tables containing x, y are adjacent in G_T can be done in $O(1)$ time using $O(n^2 + k^2) = O(n^2)$ space. Further, by Lemma A.2 and A.3, the number of such checks performed is $O(\beta_H m_{co})$. Since $\texttt{Var}(f) \subseteq V(H)$, and H is connected, $n \leq |V(H)| \leq |E(H)|$. Hence the total time complexity is $O(nm_H + \beta_H m_{co})$. $\square$

We can now finish to prove Theorem 3.8. As shown in Section 3.3.2, computation of the table-adjacency graph G_T takes $O(k^2 \alpha \log \alpha)$ time and this proves the second part of Theorem 3.8. The time complexity analysis in Lemma A.4 also holds when we modify CompCoTable to compute the co-occurrence graph G_{co} instead of the co-table graph G_C: the only change is that we do not check whether the tables containing x, y are adjacent in G_T. Further, we do not need to precompute the graph G_T. This proves the first part and completes the proof of Theorem 3.8.

A.1.4 Proofs from Section 3.4

Modified query in Algorithm 4 evaluates the same expression:

Lemma A.5. *Suppose $I = R_{i_1}[T'_{i_1}], \cdots, R_{i_p}[T'_{i_p}]$ be the set of input tables to the table decomposition procedure* TD *and let $Q() : -R_{i_1}(\mathbf{x_{i_1}}), \cdots, R_{i_p}(\mathbf{x'_{i_p}})$ be the input query. Then the expression g generated by evaluating query Q on I is exactly the same as evaluating $\widehat{Q}$ on I, where $\widehat{Q'} =$*

$\widehat{Q_1}, \cdots, \widehat{Q_\ell}$ is the conjunction of modified queries $\widehat{Q_j}$ returned by the procedure TD *for groups $j = 1$ to ℓ.*

Proof. We prove that a set of p tuple variables taken from p tables satisfy the original input query Q' if and only if they satisfy the modified query $\widehat{Q}$. Since the new query subgoals make some of the original variables *free*, by replacing them with new variables, clearly, if a set of tuples satisfy the original query they also satisfy the modified query. So we prove that the modified query does not introduce any erroneous collection of tuples in the final answer.

Consider a set of tuple variables s.t. the corresponding tuples satisfy the modified query. Let us partition these variables according to the ℓ groups of tables as computed by procedure TD. Consider component j of the partition and any table R_i in component j. Recall that C_i is the set of all variables on the "$+$"-edges having one end at the table i in component j. A "$+$" edge between table R_i and $R_{i'}$ implies that the edges between every tuple in R_i and every tuple in R_j exist, which in turn implies that, all tuples in R_i and $R_{i'}$ must have the *same* values of the attributes corresponding to $C_e = \mathbf{x_i} \cap \mathbf{x_j}$. Then any set of p tuples taken from p tables must have the same value of attributes corresponding to variables in C_e. In other words, every variable $z \in C_i$ can be replaced by a new free variable z^i in every table R_i in component j (note that $C_i \subseteq \mathbf{x_i}$) without changing the final solution. □

Proof of Lemma 3.14.

Lemma 3.14. *At any step of the recursion, if row decomposition is successful then table decomposition is unsuccessful and vice versa.*

Proof. Consider any step of the recursive procedure, where the input tables are $R_{i_1}[T'_{i_1}]$, $\cdots$, $R_{i_q}[T'_{i_q}]$ $(\forall j, T'_{i_j} \subseteq T_{i_j})$, input query is $Q'() : -R_{i_1}(\mathbf{x_{i_1}}), \cdots, R_{i_q}(\mathbf{x_{i_q}})$, and the induced subgraphs of G_C and G_T on current sets of tuples and tables are G'_C and G'_T respectively.

Suppose row decomposition is successful, i.e., it is possible to decompose the tuples in G'_C into $\ell \geq 2$ connected components. Consider any two tables R_i, R_j such that the edge (R_i, R_j) exists in G'_T, and consider their sub-tables $R_i[T_i^1]$ and $R_j[T_j^2]$ taken from two different connected components in G'_C. Consider two arbitrary tuples $x \in T_i^1$ and $x' \in T_j^2$.

Since x and x' belong to two different connected components in G'_C, then there is no edge (x, x') in G'_C. Hence by Step 3 of the table decomposition procedure, this edge (R_i, R_j) will be marked by "$-$". Since (R_i, R_j) was an arbitrary edge in G'_T, all edges in G'_T will be marked by "$-$" and there will be a unique component in G'_T using "$-$" edges. Therefore, the table decomposition procedure will be unsuccessful.

Now suppose table decomposition is successful, i.e., G'_T can be decomposed into $\ell \geq 2$ components using "$-$" edges. Note that wlog. we can assume that the initial table-adjacency graph G_T is connected. Otherwise, we can run the algorithm on different components of G_T and multiplied the final expressions from different components at the end. Since the procedure TD returns induced subgraphs for every connected components, the input subgraph G'_T is always a connected graph at every step of the recursion. Now consider any two tables R_i and R_j from two different groups components such that (R_i, R_j) edge exists in G'_T (such a pair must exist since the graph G'_T is connected). Since this edge is between two components of a successful table decomposition procedure, it must be marked with "$+$". This implies that for any tuple $x \in R_i$ and any tuple $x' \in R_j$, the edge (x, x') exists in G'_C (which follows from Step 3 of this procedure). This in turn implies that row decomposition must fail at this step since the tables R_i, R_j cannot be decomposed into two disjoint components and the graph G'_C will be connected through these tuples. □

Proof of Lemma 3.15.

LEMMA 3.15. ***(Soundness)*** *If the algorithm returns with success, then the expression f^* returned by the algorithm* COMPRO *is equivalent to the expression $Q(I)$ generated by evaluation of query Q on instance I. Further, the output expression f^* is in read-once form.*

Proof. We prove the lemma by induction on n, where $n = \bigcup_{i=1}^{k} |T_i|$. The base case follows when $n = 1$. In this case there is only one tuple x, hence k must be 1 as well, and therefore, the algorithm returns x in Step 2. Here the algorithm trivially returns with success and outputs a read-once form. The output is also correct, since computation of co-table graph ensures that there is no unused tuple in the tables, and the unique tuple x is the answer to query Q on database I.

Suppose the induction hypothesis holds for all databases with number of tuples $\leq$

$n - 1$ and consider a database with n tuples. If $k = 1$, then irrespective of the query, all tuples in table R_1 satisfies the query $Q() : -R_1(\mathbf{x_1})$ (again, there are no unused tuples), and therefore the algorithm correctly returns $\sum_{x \in T_1} x$ as the answer which is also in read-once form. So let us consider the case when $k \geq 2$.

(1) Suppose the current recursive call successfully performs row decomposition, and $\ell \geq 2$ components $\langle R_1[T_1^1], \cdots, R_k[T_k^1] \rangle, \cdots, \langle R_1[T_1^\ell], \cdots, R_k[T_k^\ell] \rangle$ are returned. By the row decomposition algorithm , it follows that for $x \in T_i^j$ and $x' \in T_{i'}^{j'}$, x and x' do not appear together in any monomial in the DNF equivalent for $Q(I)$. So the tuples which row decomposition puts in different components do not join with each other and then the final answer of the query is the union of the answers of the queries on the different components. Then the final expression is the sum of the final expressions corresponding to the different components. Since all components have $< n$ tuples and the algorithm did not return with error in any of the recursive calls, by the inductive hypothesis all the expressions returned by the recursive calls are the correct expressions and are in read-once form. Moreover these expressions clearly do not share variables – they correspond to tuples from different tables since the query does not have a self-join. We conclude that the final expression computed by the algorithm is the correct one and is in read-once form.

(2) Otherwise, suppose the current step successfully performs table decomposition. Let $\ell \geq 2$ groups $\mathbf{R_1}, \cdots, \mathbf{R}_\ell$ are returned. Correctness of table decomposition procedure, i.e., correctness of the expression $f^* = f_1 \cdot \cdots \cdot f_\ell$, when all the recursive calls return successfully follows from Lemma A.5 using the induction hypothesis (the algorithm multiplies the expressions returned by different groups which themselves are correct by the inductive hypothesis). Further, since all components have $< n$ tuples, and the algorithm did not return with error in any of the recursive calls, all expressions returned by the recursive calls are in read-once form. Since they do not share any common variable, the final output expression is also in read-once form. $\square$

Proof of Lemma 3.16.

LEMMA 3.16. ***(Completeness)*** *If the expression $Q(I)$ is read-once, then the algorithm* COMPRO *returns the unique read-once form f^* of the expression.*

Proof. Suppose the expression is read-once and consider the tree representation T^* of the unique read-once form f^* of the expression (T^* is in *canonical form* and has alternate levels of $+$ and $\cdot$ nodes, which implies that every node in T^* must have at least two children.). We prove the lemma by induction on the height h of tree T^*.

First consider the base case. If $h = 1$, then the tree must have a single node for a single tuple variable x. Then k must be 1 and the algorithm returns the correct answer. So consider $h \geq 2$.

(1) Consider the case when root of the tree is a $+$ node. If $h = 2$, since we do not allow union operation, k must be 1 and all the tuples must belong to the same table R_1. This is taken care of by Step 2 of CompRO. If $h > 2$, then k must be ≥ 2 and the answer to the join operation must be non-empty. Every child of the root node corresponds to a set of monomials which will be generated by the equivalent DNF expression f_{DNF} for the subtree rooted at that child. Note that no two variables in two different children of the root node can belong to any monomial together since the tree T^* is in read-once form. In other words, they do not share an edge in G_C. Hence the component formed by the set of variables at a child will not have any edge to the set of variables at another child of the root node. This shows that all variables at different children of the root node will belong to different components by the row decomposition procedure.

Now we show that variables at different children of the root node are put to different components by the row decomposition procedure, which shows that the row decomposition algorithm will divide the tuples *exactly* the same was as the root of T^* divides tuples among its children. Since T^* is in canonical read-once form and has alternate levels of $+$ and $\cdot$ nodes, then row decomposition cannot be done within the same subtree of a $+$ node. So all variables in a subtree must form a connected component. Since the root has ≥ 2 children, in this case we will have a successful row decomposition operation. By inductive hypothesis, since the subtrees rooted at the children of the root are all in read-once form, the recursive calls of the algorithm on the corresponding subtrees are successful. Hence the overall algorithm at the top-most level will be successful.

(2) Now consider the case when root of the tree is a $\cdot$ node. Note that the $\cdot$ operator can only appear as a result of join operation. If the root has $\ell' \geq 2$ children $c_1, \cdots, c_{\ell'}$, then

every tuple x in the subtree at c_j joins with every tuple y in the subtree at $c_{j'}$ for every pair $1 \leq j \neq j' \leq \ell'$. Moreover since the query does not have a self join, x, y must belong to two different tables, which implies that there is an edge (x, y) in G_C between every pair of tuples x, y from subtrees at $c_j, c_{j'}$ respectively. Again, since we do not allow self-join, and T^* is in read-once form, the tuples in the subtrees at $c_j, c_{j'}$ must belong to different tables if $j \neq j'$. In other words, the tables $R_1, \cdots, R_k$ are partitioned into ℓ' disjoint groups $\mathbf{R'_1}, \cdots, \mathbf{R'_{\ell'}}$.

Next we argue that $\ell = \ell'$ and the partition returned by the table decomposition procedure $\mathbf{R_1}, \cdots, \mathbf{R_\ell}$ is identical to $\mathbf{R'_1}, \cdots, \mathbf{R'_{\ell'}}$ upto a permutation of indices. Consider any pair $\mathbf{R'_j}$ and $\mathbf{R'_{j'}}$. Since the tuple variables in these two groups are connected by a $\cdot$ operator, all tuples in all tables in $\mathbf{R'_j}$ join with all tuples in all tables in $\mathbf{R'_{j'}}$. In other words, for any pair of tuples x, x' from $R_{i_1} \in \mathbf{R'_j}$ and $R_{i_2} \in \mathbf{R'_{j'}}$, there is an edge (x, x') in co-occurrence graph G_{co}.Hence if there is a common subset of join attributes between R_{i_1} and R_{i_2}, i.e. the edge (R_{i_1}, R_{i_2}) exists in G_T, it will be marked by a "+" (all possible edges between tuples will exist in the co-table graph G_T). So the table decomposition procedure will put $\mathbf{R'_j}$ and $\mathbf{R'_{j'}}$ in two different components. This shows that $\ell \geq \ell'$. However, since T^* is in read-once form and has alternate levels of $+$ and $\cdot$ nodes, no $\mathbf{R'_j}$ can be decomposed further using join operation (i.e. using "+" marked edges by the table decomposition procedure); therefore, $\ell' = \ell$. Hence our table decomposition operations exactly outputs the groups $\mathbf{R'_1}, \cdots, \mathbf{R'_{\ell'}}$. By the inductive hypothesis the algorithm returns with success in all recursive calls, and since $\ell' = \ell \geq 2$, the table decomposition returns with success. So the algorithm returns with success. $\square$

A.1.5 Time Complexity of CompRO

Here we discuss the time complexity of algorithm CompRO in detail and show that algorithm CompRO runs in time $O(m_T \alpha \log \alpha + (m_C + n) \min(k, \sqrt{n}))$. We divide the time complexity computation in two parts: (i) total time required to compute the modified queries across *all* table decomposition steps performed by the algorithm (this will give $O(m_T \alpha \log \alpha)$ time) and (ii) total time required for all other steps: here we will *ignore* the time complexity for the modified query computation step and will get a bound of

$(m_C + n)\min(k, \sqrt{n}))$. First we bound the time complexity of individual row decomposition and table decomposition steps.

Lemma A.6. *The row decomposition procedure as given in Algorithm 3 runs in time $O(m'_C + n')$, where $n' = \sum_{j=1}^{q} |T_{i_j}|$ = the total number of input tuples to the procedure, and m'_C = the number of edges in the induced subgraph of G_C on these n' tuples.*

Proof. The row decomposition procedure only runs a connectivity algorithm like BFS/DFS to compute the connected components. Then it collects and returns the tuples and computes the induced subgraphs in these components. All these can be done in linear time in the size of the input graph which is $O(m'_C + n')$. □

Next we show that the table decomposition can be executed in time $O(m'_C + n')$ as well.

Lemma A.7. *The table decomposition procedure as given in Algorithm 4 runs in time $O(m'_C + n')$, ignoring the time required to compute the modified queries $\widehat{Q_j}$ where $n' = \sum_{j=1}^{q} |T'_{i_j}|$ = the total number of input tuples to the procedure, and m'_C = the number of edges in the induced subgraph of G_C on these n' tuples.*

Proof. Step 3 in the table decomposition procedure marks edges in G'_T using G'_C. Let us assume that G'_C has been represented in a standard adjacency list. Consider a table $R_j[T'_j]$, where $T'_j \subseteq T_j$ and let d be the degree of R_j in G'_T. Now a linear scan over the edges in G'_C can partition the edges e from a tuple $x \in T'_j$ in table R_j into $E_1, \cdots, E_d$, where E_q ($q \in [1, d]$) contains all edges from x to tuples x', belonging to the q-th neighbor of R_j. A second linear scan on these grouped adjacency lists computed in the previous step is sufficient to mark every edge in G'_T with a "+" or a "−": for every neighbor q of R_j, say $R_{j'}$, for every tuple x in T'_j, scan the q-th group in adjacency list to check if x has edges with all tuples in $R_{j'}[T'_{j'}]$. If yes, then all tuples in $R_{j'}$ also have edges to all tuples in R_j, and the edge $(R_j, R_{j'})$ is marked with a "+". Otherwise, the edge is marked with a "−". Hence the above two steps take $O(m'_C + n' + m'_T + k')$ time, where k' and m'_t are the number of vertices (number of input tables) and edges in the subgraph G'_T.

Finally returning the induced subgraphs of G'_T for the connected components and decomposition of the tuples takes $O(m'_C + n' + m'_T + k')$ time. Since $n' \geq k'$ and $m'_C \geq m'_T$, not considering the time needed to recompute the queries, step, the total time complexity is bounded by $O(m'_C + n')$. $\square$

The next lemma bounds the total time required to compute the modified queries over all calls to the recursive algorithm.

Lemma A.8. *The modified queries $\widehat{Q_j}$ over all steps can be computed in time $O(m_T \alpha \log \alpha)$, where α is the maximum size of a subgoal.*

Proof. We will use a simple charging argument to prove this lemma. For an edge $e = (R_i, R_j)$ in G_T, the common variable set $C_e = \mathbf{x_i} \cap \mathbf{x_j}$[42] can be computed by (i) first sorting the variables in $\mathbf{x_i}, \mathbf{x_j}$ in some fixed order, and then (ii) doing a linear scan on these sorted lists to compute the common variables. Here we to compute the set C_e. Hence this step takes $O(\alpha \log \alpha)$ time. Alternatively, we can use a hash table to store the variables in $\mathbf{x_i}$, and then by a single scan of variables in $\mathbf{x_j}$ and using this hash table we can compute the common attribute set C_e in $O(\alpha)$ expected time. When C_e has been computed in a fixed sorted order for every edge e incident on R_i to a different component, the lists C_e-s can be repeatedly merged to compute the variables set $C_i = \bigcup_e C_e$ in $O(d_i \alpha)$ time (note that even after merging any number of C_e sets, the individual lists length are bounded by the subgoal size of R_i which is bounded by α). However, instead of considering the total time $O(d_i \alpha)$ for the node R_i in G_T, we will *charge* every such edge $e = (R_i, R_j)$ in G_T for this merging procedure an amount of $O(\alpha)$. So every edge e from R_i to an R_j in different component gets a charge of $O(\alpha \log \alpha)$.

Suppose we charge the outgoing edges (R_i, R_j) from R_i to different components by a fixed cost of P, $P = O(\alpha)$ in the above process. From the table decomposition procedure it follows that, the common join attributes are computed, and the query is updated, only when the edge (R_i, R_j) belongs to the *cut* between two connected components formed by the "$-$" edges. These edges then get *deleted* by the table decomposition procedure: all the following recursive calls consider the edges *inside* these connected components and the edges *between* two connected components are never considered later. So each edge in the graph G_T can be *charged* at most once for computation of common join attributes and this gives $O(m_C)\alpha \log \alpha$ as the total time required for this process.

[42]We abuse the notation and consider the *sets* corresponding to *vectors* $\mathbf{x_i}, \mathbf{x_j}$ to compute C_e

Finally, the variables in $\mathbf{x_i}$ can also be replaced by new variables using the sorted list for C_i in $O(\alpha)$ time, so the total time needed is $O(m_C \alpha \log \alpha + n\alpha) = O(m_C \alpha \log \alpha)$ (since we assumed G_T for the query Q is connected without loss of generality). $\square$

Now we show that the depth of the recursion tree is $O(\min(k, \sqrt{n}))$ and in every level of the tree, the total time required is at most $O(m_C + n)$. Let us consider the recursion tree of the algorithm CompRO and wlog. assume that the top-most level performs a row decomposition. Since the size of the table-adjacency subgraph G'_T is always dominated by the co-table subgraph G'_C at any recursive call of the algorithm, we express the time complexity of the algorithm with k tables, and, n tuples and m edges in the subgraph G'_C as $T_1(n,m,k)$, where the top-most operation is a row decomposition. Further, every component after row decomposition has exactly k tables and therefore must have at least k tuples, because, we assumed wlog. that initial table adjacency graph G_T is connected and there is no unused tuples in the tables. Similarly, $T_2(n,m,k)$ denotes the time complexity when the top-most operation is a table decomposition operation. Note that at every step, for row decomposition, every tuple and every edge in G'_C goes to exactly one of the recursive calls of the algorithm; however, the number of tables k remains unchanged. On the other hand, for table decomposition operation, every tuple goes to exactly one recursive call, every edge goes to at most one such calls (edges between connected components are discarded), and every table goes to exactly one call. Recall that the row and table decomposition alternates at every step, and the time required for both steps is $O(m+n)$ (not considering computation of modified queries at every table decomposition steps) so we have the following recursive formula for $T_1(n,m,k)$ and $T_2(n,m,k)$.

$$\begin{aligned} T_1(n,m,k) &= O(m+n) + \sum_{j=1}^{\ell} T_2(n_j,m_j,k) && \text{where } \sum_{j=1}^{\ell} n_j = n, \sum_{j=1}^{\ell} m_j = m, n_j \geq k \forall j \\ T_2(n,m,k) &= O(m+n) + \sum_{j=1}^{\ell} T_1(n_j,m_j,k_j) && \text{where } \sum_{j=1}^{\ell} n_j = n, \sum_{j=1}^{\ell} m_j \leq m, \sum_{j=1}^{\ell} k_j = k \end{aligned}$$

where n_j, m_j and k_j are the total number of tuples and edges in G'_C, and the number of tables for the j-th recursive call (for row decomposition, $k_j = k$). For the base case, we have $T_2(n_j, m_j, 1) = O(n_j)$ – for $k = 1$, to compute the the read once form, $O(n_j)$ time is needed; also in this case $m_j = 0$ (a row decomposition cannot be a leaf in the recursion tree for a successful completion of the algorithm). Moreover, it is important to note that for a successful row or table decomposition, $\ell \geq 2$.

If we draw the recursion tree for $T_1(n, m_C, k)$ (assuming the top-most operation is a row-decomposition operation), at every level of the tree we pay cost at most $O(m_C + n)$. This is because the tuples and edges go to at most one of the recursive calls and k does not play a role at any node of the recursion tree (and is absorbed by the term $O(m_C + n)$).

Now we give a bound on the height of the recursion tree.

Lemma A.9. *The height of the recursion tree is upper bounded by $O(\min(k, \sqrt{n}))$.*

Proof. Every internal node has at least two children and there are at most k leaves (we return from a path in the recursion tree when k becomes 1). Therefore, there are $O(k)$ nodes in the tree and the height of the tree is bounded by $O(k)$ (note that both the number of nodes and the height may be $\Theta(k)$ when the tree is not balanced).

Next we show that the height of the recursion tree is also bounded by $4\sqrt{n}$. The recursion tree has alternate layers of table and row decomposition. We focus on only the table decomposition layers, the height of the tree will be at most twice the number of these layers. Now consider any arbitrary path P in the recursion tree from the root to a leaf where the number of table decompositions on P is h. Suppose that in the calls $T_2(n, m, k)$, the values of n and k along this path (for the table decomposition layers) are $(n_0, k_0), (n_1, k_1), \ldots, (n_h, k_h)$, where $k_0 = k$ and $n_0 \leq n$ (if the top-most level has a table-decomposition operation, then $n_0 = n$). We show that $h \leq 2\sqrt{n}$.

Let's assume the contradiction that $h > 2\sqrt{n}$ and let's look at the first $p = 2\sqrt{n}$ levels along the path P. If at any j-th layer, $j \in [1, p]$, $k_j \leq 2\sqrt{n} - j$, then the number of table decomposition steps along P is at most $2\sqrt{n}$: every node in the recursion tree has at least two children, so the value of k decreases by at least 1. The number of table-decomposition layers after the j-th node is at most k_j, and the number of table-decomposition layers before the j-th node is exactly j. Therefore, the total number of table-decomposition layers is $\leq 2\sqrt{n}$).

Otherwise, for all $j \in [1, p]$, $k_j > 2\sqrt{n} - j$. Note that $n_j \leq n_{j-1} - k_{j-1}$: there is a row decomposition step between two table decompositions, and every component in the j-th row decomposition step will have at least k_j nodes. If this is the case, we show that $n_p < 0$.

However,

$$\begin{aligned} n_p &\leq n_{p-1} - k_{p-1} \leq n_{p-2} - k_{p-2} - k_{p-1} \leq \cdots \leq n_0 - \sum_{j=0}^{p-1} k_j \leq n - \sum_{j=0}^{p-1} k_j \\ &\leq n - \sum_{j=0}^{2\sqrt{n}-1} (2\sqrt{n} - j) = n - \sum_{j=1}^{2\sqrt{n}} j = n - \frac{2\sqrt{n}(2\sqrt{n}+1)}{2} = n - 2n - \sqrt{n} < 0 \end{aligned}$$

which is a contradiction since n_p is the number of nodes at a recursive call and cannot be negative. This shows that along any path from root to leaves, the number of table decomposition layers is bounded by $2\sqrt{n}$ which in turn shows that the height of the tree is bounded by $4\sqrt{n}$. □

Since total time needed at every step of the recursion tree is $O(m_C + n)$, we have the following corollary,

Corollary A.10. *Not considering the time complexity to compute the modified queries by the table decomposition procedure, the algorithm* CompRO *runs in time $O((m_C + n)\min(k, \sqrt{n}))$.*

The above corollary together with Lemma A.8 (which says that to compute the modified queries $O(m_T \alpha \log \alpha)$ time suffices) shows that CompRO runs in time $O(m_T \alpha \log \alpha + (m_C + n)\min(k, \sqrt{n}))$.

A.2 Proofs from Chapter 4

A.2.1 Proof of Proposition 4.4

Proposition 4.4. *For any relational algebra query q such that $\delta(q) \leq 1$, and for any probabilistic database I where $|I| = n$, the boolean provenance of any tuple $t \in q(I)$ can be computed in poly-time in n in the form*

$$\alpha_0 + \sum_{i=1}^{r} \alpha_i \overline{\beta_i}$$

where each of $\alpha_0, \cdots, \alpha_r$ and $\beta_1, \cdots, \beta_r$ is a monotone DNF in poly-size in n while r is also polynomial in n; moreover, if $\delta(q) = 0$, then $r = 0$ (we have a single DNF α_0).

We combine the proofs of Proposition 4.1 and Proposition 4.3 together since they use similar induction argument.

Proof. Let us denote by $\mathcal{N}(q, I)$, the number of tuples in the answer output by query q on instance I. We prove by induction on the size of the $\mathcal{RA}$ expression q that (1) $\mathcal{N}(q, I)$ is bounded by $n^{|q|}$, (b) the provenance expression ϕ_t of all result tuples t can be computed in total time $O(n^{c|q|})$, for some constant $c \geq 3$, such that (a) each expression ϕ_t has the form $\phi_t = \alpha_0 + \sum_{i=1}^{r} \alpha_i \overline{\beta_i}$, (b) for every such ϕ_t, for all α_i and β_j, $|\alpha_i|$ and β_j are bounded by $n^{|q|+1}$, (c) $\sum_{t \in q[I]} r_t \leq n^{|q|}$, and, (d) for each t, $r_t = 0$ if $\delta(q) = 0$ (it may be the case that $r_t = 0$ even when $\delta(q) = 1$). Since $|q|$ is considered a constant, this will imply the statements of both Proposition 4.1 and Proposition 4.3.

The base case follows for queries q such that $|q| = 1$, which must be a single base relation R; hence $\delta(q) = 0$. Here $\mathcal{N}(q, I) \leq n$, the provenance expressions are the tuple variable themselves, so they can be computed in $O(n) = O(n^c)$ time. The tuple variables are trivially IDNF expression α_0 with $r_t = 0$ (hence $\sum_{t \in q[I]} r_t = 0 \leq n^{|q|}$) and $|\alpha_0| = 1 \leq n^{|q|}$.

Suppose the induction hypothesis holds for all $\mathcal{RA}$ expressions q such that $|q| < k$, where $\delta(q) \leq 1$. Now consider a query q such that $|q| = k$, $k > 1$. The hypothesis is proved for all possible operations in q.

1. (Project) If q is of the form Πq_1, then $|q_1| = |q| - 1 < k$. Hence $\mathcal{N}(q, I) \leq \mathcal{N}(q_1, I) \leq n^{|q_1|}$ (by IH), $\leq n^{|q|}$. The provenance expression for every tuple t will equal to $\phi_{t_1} + \cdots + \phi_{t_\ell}$, for some tuples $t_1, \cdots, t_\ell$ in $q_1[I]$. Since $\ell \leq n^{|q_1|}$, this can be computed in time $O(n^{3|q_1|}) + O(n^{c|q_1|}) = O(n^{c|q_1|}) = O(n^{c|q|})$ (the total time complexity to compute the project operation even by brute-force method is $O(\mathcal{N}(q_1, I) \times \mathcal{N}(q_1, I) \times \max_{t_1 \in q_1[I]})|\phi_{t_1}| = O(n^{3|q_1|})$, and the time complexity to compute $q_1[I]$ is $O(n^{c|q_1|})$) . Also, $\sum_{t \in q[I]} r_t$ is bounded by $\sum_{t \in q_1[I]} r_{t_1} \leq n^{|q_1|}$ (by IH), $\leq n^{|q|}$. This is because every tuple in $q_1[I]$ contributes to exactly one tuple in $q[I]$.

   $\phi_t = \phi_{t_1} + \cdots + \phi_{t_\ell}$ will have the form $\alpha_0 + \sum_{i=1}^{r} \alpha_i \overline{\beta_i}$ by IH. Size of every DNF α_i-s and β_j-s remains the same, hence the sizes are bounded by $n^{|q_1|} \leq n^{|q|}$. Further, if $\delta(q) = 0$, $\delta(q_1) = 0$ as well. So the value of $r = r(t_j)$ is 0 for all $j \in [1, \ell]$. Hence r_t is also 0.

2. (Select) Here $q = \sigma q_1$, hence $|q| = |q_1| + 1$. In this case $\phi_t = \phi_{t_1}$, for some t_1 in $q_1[I]$. Hence $\mathcal{N}(q, I) \leq n^{|q_1|}$ (by IH), $\leq n^{|q|}$, the tuple can be output in time $O(n^{|q_1|}) +$

$O(n^{c|q_1|}) = O(n^{c|q|})$. The expression $\phi_t = \phi_{t_1}$ has the required form, and α_i, β_j in ϕ_t has the required size bound by IH, both when $\delta(q) = 0$ and $\delta(q) = 1$.

3. (Join) Let $q = q_1 \bowtie q_2$, i.e. $|q| = |q_1| + |q_2| + 1$. In the worst case, every tuple in q_1 can join with every tuple in q_2. Therefore, $\mathcal{N}(q, I) \leq \mathcal{N}(q_1, I) \times \mathcal{N}(q_2, I)$, $\leq n^{|q_1|} \times n^{|q_2|}$ (by IH), $\leq n^{|q|}$. If $\delta(q) = 0$, both $\delta(q_1), \delta(q_2)$ are 0, therefore we have the required form of ϕ_t for a tuple t, where $r_t = 0$. Hence $\sum_{t \in q[I]} r_t = 0$ as well.

   If $\delta(q) = 1$, exactly one of $\delta(q_1), \delta(q_2)$ is 1; wlog. let $\delta(q_1) = 1, \delta(q_2) = 0$. Then also for every tuple ϕ_t in $q[I]$, $\phi_t = \phi_{t_1} \times \phi_{t_2}$. Let $\phi_{t_1} = \alpha_0 + \sum_{i=1}^{r} \alpha_i \overline{\beta_i}$, $\phi_{t_2} = \alpha_0'$. Then $\phi_t = \left(\alpha_0 + \sum_{i=1}^{r} \alpha_i \overline{\beta_i}\right) \times \alpha_0' = = \alpha_0'' + \sum_{i=1}^{r} \alpha_i'' \overline{\beta_i}$, where α_i'', $i \in [0, r]$, are obtained by computing the IDNF of $\alpha_i \times \alpha_0'$. Hence $\sum_{t \in q[I]} r_t = (\sum_{t \in q[I]} r_t) \times \mathcal{N}(q_2, I)$ (since every tuple in $q_1[I]$ can join with at most $\mathcal{N}(q_2, I)$ tuples in $q_2[I]$, and every tuple t_2 in $q_2[I]$ have $r_{t_2} = 0$) $\leq n^{|q_1|} \times n^{|q_2|}$ (by IH) $\leq n^{|q_1|+|q_2|} \leq n^{|q|}$. Size of $|\beta_j|$ remains the same, and $|\alpha_i''| \leq n^{|q_1|} \times n^{|q_2|} \leq n^{|q|}$. The time complexity of the brute-force method can be shown to be $O(n^{c|q|})$.

4. (Union) Let $q = q_1 \cup q_2$, i.e. $|q| = |q_1| + |q_2| + 1$. Therefore, $\mathcal{N}(q, I) = \mathcal{N}(q_1, I) + \mathcal{N}(q_2, I)$, $\leq n^{|q_1|} + n^{|q_2|}$ (by IH), $\leq n^{|q|}$.

   For union, either (i) $\phi_t = \phi_{t_1}$, for some t_1 in $q_1[I]$ or $q_2[I]$, or, (ii) $\phi_t = \phi_{t_1} + \phi_{t_2}$, for some t_1 in $q_1[I]$ and t_2 in $q_2[I]$. If $\delta(q) = 0$, both $\delta(q_1), \delta(q_2)$ are 0, therefore we have the required form of ϕ_t for a tuple t, where $r_t = 0$ (for both cases (i) and (ii)). If $\delta(q) = 1$, then also for both cases (i) and (ii) and for both (a) when exactly one of $\delta(q_1), \delta(q_2)$ is 1, and (b) when $\delta(q_1) = \delta(q_2) = 1$, ϕ_t has the required form. Further, $\sum_{t \in q[I]} r_t = \sum_{t_1 \in q_1[I]} r_{t_1} + \sum_{t_2 \in q_2[I]} r_{t_2}$ (since every tuple in $q_1[I]$ or $q_2[I]$ contributes to exactly one tuple in $q[I]$), $\leq n^{|q_1|} + n^{|q_2|} \leq n^{|q_1|+|q_2|+1} = n^{|q|}$. Size of every α_i, β_j in ϕ_t is bounded by $n^{|q_1|+1} + n^{|q_2|+1} \leq n^{|q|+1}$.

   To compute $\phi_t = \phi_{t_1} + \phi_{t_2}$, for every tuple t_1 in $q_1[I]$, we need to go over all tuples in $q_2[I]$, but unlike join, the expressions are not needed to be multiplied. Hence the union can be computed in total time $O(n^{|q|} \times n^{k_2} \times (n^{k_1}(n^{k_1} + 1) + n^{k_2} n^{k_2+1})$ $+ O(n^{ck_1}) + O(n^{ck_2})$ (by IH), $= O(n^{c(k_1+k_2+1)}) = O(n^{c|q|})$.

5. (Difference) Let $q = q_1 - q_2$, i.e. $|q| = |q_1| + |q_2| + 1$. Therefore, $\mathcal{N}(q, I) \leq \mathcal{N}(q_1, I)$,

$\leq n^{|q_1|}$ (by IH), $\leq n^{|q|}$.

For difference, $\delta(q)$ can not be 0, i.e., $\delta(q) = 1$, and it must be the case that $\delta(q_1) = \delta(q_2) = 0$. For a tuple t in $q[I]$, either (i) $\phi_t = \phi_{t_1}$, for some t_1 in $q_1[I]$ (when there is no tuple in $q_2[I]$ that has the same value as in t), or, (ii) $\phi_t = \phi_{t_1} \times \overline{\phi_{t_2}}$, for some t_1 in $q_1[I]$ and t_2 in $q_2[I]$. In case (i), ϕ_t obviously has the required form, $r_t = 0$, and $|\alpha_i|, |\beta_j|$ have the required size bound. In case (ii), since $\delta(q_1) = \delta(q_2) = 0$, we have by IH, $\phi_{t_1} = \alpha_0$ and $\phi_{t_2} = \alpha'_0$, where both α_0, α'_0 are positive IDNF. Hence $\phi_t = \phi_{t_1} \times \overline{\phi_{t_2}} = \alpha_0 \overline{\alpha'_0}$. Hence again ϕ_t has the required form, and again $|\alpha_0|$ and $|\alpha'_0|$ individually have the required size bound. For both t_1, t_2, $r_{t_1} = r_{t_2} = 0$, so $r_t = 1$. Hence $\sum_{t \in q[I]} r_t \leq \mathcal{N}(q, I) \leq n^{|q|}$. The difference can be computed in time $O(n^{|q_1|} \times n^{|q_2|}) + O(n^{c|q_1|}) + O(n^{c|q_2|})$ (by IH), $= O(n^{c|q|})$.

This completes the proof of the proposition. □

A.2.2 Proof of Theorem 4.5

The following lemma shows a nice property of product of two RO expressions and will be useful in proving Step3 and Step4.

Lemma A.11. *Let ϕ_1, ϕ_2 be two read-once Boolean expression on the same set of variables. Then either each of $\texttt{C}(\phi_1 \cdot \overline{\phi_2})$, $\texttt{C}(\phi_1 \cdot \phi_2)$, $\texttt{C}(\overline{\phi_1} \cdot \phi_2)$ and $\texttt{C}(\overline{\phi_1} \cdot \overline{\phi_2})$ is poly-time computable or none of them is.*

Proof. Let $n = |\texttt{Var}(\phi_1)| = |\texttt{Var}(\phi_2)|$. If ϕ is an RO expression, then $\texttt{C}(\phi)$ can be exactly computed in time linear in the size of ϕ. Therefore, $\texttt{C}(\overline{\phi_1}) = 2^n - \texttt{C}(\phi)$ can also be efficiently computed. Now we have, (i) $\texttt{C}(\phi_1) = \texttt{C}(\phi_1 \cdot \phi_2) + \texttt{C}(\phi_1 \cdot \overline{\phi_2})$, (ii) $\texttt{C}(\overline{\phi_1}) = \texttt{C}(\overline{\phi_1} \cdot \phi_2) + \texttt{C}(\overline{\phi_1} \cdot \overline{\phi_2})$, (iii) $\texttt{C}(\phi_2) = \texttt{C}(\phi_1 \cdot \phi_2) + \texttt{C}(\overline{\phi_1} \cdot \phi_2)$, and, (iv) $\texttt{C}(\overline{\phi_2}) = \texttt{C}(\phi_1 \cdot \overline{\phi_2}) + \texttt{C}(\overline{\phi_1} \cdot \overline{\phi_2})$. Using the above equations, if any one of $\texttt{C}(\phi_1 \cdot \phi_2)$, $\texttt{C}(\phi_1 \cdot \overline{\phi_2})$, $\texttt{C}(\overline{\phi_1} \cdot \phi_2)$, or $\texttt{C}(\overline{\phi_1} \cdot \overline{\phi_2})$ can be exactly computed in poly-time, all the others can be exactly computed in poly-time. □

Now we discuss the details of these four steps.

A.2.3 Step1: #P-hardness of **#4Partite-4BEC**

For a graph $G(V,E)$, a subset $U \subseteq E$ is an *edge cover* if every vertex of G has at least one incident edge in U. The following theorem shows the #P-hardness of the #4Partite-4BEC problem. The proof uses similar idea as given in [23] which reduces #independent set problem in a constant degree graph to the #edge-cover problem in a graph of minimum degree Δ, and shows that by constructing a number of #edge-cover instances, and using the *interpolation technique* [168], independent set in a constant degree graph can be exactly computed. However, the edge-cover instances constructed in [23] are non-bipartite and the vertices in those instances do not have constant degrees; hence we need to modify the construction.

We reduce the problem of #independent set in 3-regular bipartite graphs (#3-reg-BIS) to the #4Partite-4BECproblem. The #vertex-cover problem in 3-regular bipartite graphs is known to be #P-hard [176]. In a graph $G(V,E)$, $S \subseteq V$ is a vertex cover if and only if $V \setminus S$ is an independent set. Hence the set of vertex cover has a 1-1 correspondence with the set of independent sets in any graph which shows that #3-reg-BIS is #P-hard.

Theorem A.12. *#4Partite-4BECis #P-hard.*

Proof. Let $G(V_1,V_2,E)$ be a bipartite 3-regular graph, where $|V_1| = |V_2| = n'$. We will use $V = V_1 \cup V_2$ to denote the vertex set of G, and and $n = 2n'$ to denote the total number of vertices in G. Let $I_j(G)$ be the number of independent sets of size j, $j \in [0,n]$. Form G' by inserting a vertex u_e on every edge $e \in E$. Let $U = \{x_e : e \in E\}$ ($|U| = 3n/2$). Let $N_i(G')$ be the number of edge subsets in G' which leave exactly i vertices in V uncovered, but no vertex in U uncovered($N_0(G')$ is the number of edge covers in G'). The set of vertices in G' which are not covered by such a subset of edges forms an independent set in G: if a vertex $u \in V$ is uncovered in G', for every incident edge e on u, $e = (u,v) \in E$, x_e and hence v must be covered.

Let $W \subset V$ be an independent set of size j in G. Let us count the number of edge subsets in G' that (i) do not cover W, (ii) cover all members of U, and (iii) may or may not cover the vertices in $V \setminus W$. The $3j$ edges incident on W must not be chosen in such a subset, hence the $3j$ edges next to these edges must be chosen to cover all members in U. Hence we are left with $3n - 6j$ edges which can be paired up into $\frac{3n-6j}{2}$ groups according to the edges in E they came from. At least one edge in each such pair must be chosen to cover vertices in U, so we have 3 choices for each pair. Hence for a fixed independent set W of size j, the number of choices is $3^{(3n-6j)/2}$. If we multiply this quantity with the number of independent sets of size j in G, i.e. $I_j(G)$, we double count some of the edge subsets which cover more than one independent sets of size j. An edge subset which covers exactly i vertices from V and all vertices from U is counted $\binom{i}{j}$ times, once for every choice of j out of those i vertices the subset covers. Hence it follows that

$$3^{(3n-6j)/2} I_j(G) = \sum_{i=j}^{n} \binom{i}{j} N_i(G') \tag{A.3}$$

Hence if we can compute all $N_i(G')$, $i \in [0, n]$, we can compute the number of all independent sets $\sum_{j=0}^{n} I_j(G)$ in G.

So focus on computing all $N_i(G')$ in G'. Let C_k denote a chain with k nodes. We attach a copy C_s^v of C_s to each vertex $v \in V$ by identifying v with an endpoint of C_s. Let us call the resulting graph G'_s. Since we started with a regular bipartite graph G of degree 3, G'_s is a bipartite graph with maximum degree 4 (vertices in V have degree 4, vertices in U have degree 2, and the vertices in the chains C_s^v, $v \in V$ have degree at most 2).

Let M_k denote the number of edge covers in C_k. Now we count the number of edge covers in G'_s. Each edge cover of G'_s induces an edge subset of G'. To extend an edge subset of G' which leaves exactly i vertices in V uncovered, and no vertex in U uncovered, to an edge cover of G'_s, we must select an edge cover for i copies of C_s, but either an edge cover or a cover missing the first vertex (identified with the corresponding x_e) for the remaining $n-i$ copies of C_s. Thus the total number of edge covers $N_0(G'_s)$ is given by:

$$\begin{aligned} N_0(G'_s) &= \sum_{i=0}^{n} M_s^i (M_s + M_{s-1})^{n-i} N_i(G') \\ &= M_s^n \sum_{i=0}^{n} (1 + \frac{M_{s-1}}{M_s})^{n-i} N_i(G') \end{aligned} \tag{A.4}$$

In Lemma A.13 we show that the edge set can indeed be partitioned into four disjoint matchings, which shows that each G'_s, for any value of s, is an instance of the `#4Partite-4BEC` problem.

Suppose we are given an oracle for the #4Partite-4BEC problem. That oracle can be used to compute $N_0(G'_s)$ for any s value. From Lemma A.17 the values of $\frac{M_{s-1}}{M_s}$ are distinct for all distinct s, and therefore from Fact A.16, all the coefficients $N_i(G')$ can be computed by calling the edge cover oracle on G'_s graphs for $n+1$ distinct values of s (in (A.4). From these $N_i(G')$ values the number of independent sets in a 3-regular graph can be exactly computed using (A.3). This completes the reduction. □

Lemma A.13. *The set of edges in each G'_s in the proof of Theorem A.12 can be partitioned into set of four disjoint matchings.*

Proof. Recall that we started with a 3-regular bipartite graph $G(V, E)$, inserted the vertex set $U = \{x_e : e \in E\}$ into the edges in E to form the graph G', and attached chains with s vertices, C_s with every vertex in V to form the graph G'_s.

It is well-known that a 3-regular graph can be partitioned into 3 disjoint perfect matchings (see, for instance [22]). Let a set of 3 disjoint matchings for G be E_1, E_2, E_3, where $E = E_1 \cup E_2 \cup E_3$. Let V_1, V_2 be the bipartition of vertices in G ($V = V_1 \cup V_2$), and let E_i^1, E_i^2, $i \in [1,3]$, denote the set of edges in G' which are incident on V_1 and V_2 respectively. Note that $\bigcup_{i=1}^{3}(E_i^1 \cup E_i^2)$ is exactly the edge set in G'.

Further, any chain C_s can be partitioned into two disjoint matchings (comprising the alternating edges). For a copy of the chain C_s^v, $v \in V$, let $M_1(C_s^v)$ the matching which includes the edge incident on v, and $M_2(C_s^v)$ be the other matching.

Now we construct four disjoint group of matchings P_1, P_2, P_3, P_4 as follows (there may be several other ways): (i) $P_1 = E_1^1 \cup E_2^2 \cup \bigcup_{s \in V}\{M_2(C_s^v)\}$, (ii) $P_2 = E_2^1 \cup E_3^2$, (iii) $P_3 = E_3^1 \cup E_1^2$, (iv) $P_4 = \bigcup_{s \in V}\{M_1(C_s^v)\}$.

Clearly, $P_1 \cup P_2 \cup P_3 \cup P_4$ is exactly the edge set of G'_s. The fact that P_4 is a matching follows easily, since all the copies of the chains C_s^v are disjoint. Each E_j^i, $i \in [1,2], j \in [1,3]$ forms a matching themselves, since they are taken from the matchings E_1, E_2, E_3 in G. In P_1, P_2 or P_3, the edges taken from G' are of the form $E_j^i \cup E_{j'}^{i'}$, where $i \neq i'$ and $j \neq j'$. Since $i \neq i'$, if the edges E_j^i are incident on V_1, $E_{j'}^{i'}$ are incident on V_2 (or vice versa). Since $j \neq j'$, no two edges incident on some $x_e \in U$ are ever included together. Moreover, P_1 includes the edges from the chains $\bigcup_{s \in V}\{M_2(C_s^v)\}$ which themselves are matchings, and are not incident on any vertex in $V_1 \cup V_2$ Therefore, the P_i-s, $i \in [1,4]$ partition the edge set in G'_s

in four disjoint matchings. $\square$

The following lemma counts the number of edge covers in a chain (similar to vertex cover counting in a chain [168]).

Lemma A.14. *Let M_k denote the number of edge covers in a chain graph with k nodes C_k. Then M_k can be computed by the recurrence $M_k = M_{k-1} + M_{k-2}$, for $k \geq 3$, and $M_2 = M_3 = 1$ (there is no edge cover for a singleton node).*

Proof. In C_2, the unique edge must be chosen, so $M_2 = 1$. Again in C_3, both the edges must be chosen to cover two endpoints, hence $M_3 = 1$.

Let $v_0, \cdots, v_k$ be the vertices in the chain C_k, $k \geq 3$. The first edge (v_0, v_1) in the chain must be included in any edge cover which also covers v_1. Hence v_1 may or may not be covered by the edge (v_1, v_2). The number of edge subsets in the chain $v_1, \cdot, v_k$ which covers v_1 (and the other vertices $v_2, \cdots, v_k$), i.e. includes (v_1, v_2), is M_{k-1}, whereas the number of edge subsets which do not cover v_1 (does not include (v_1, v_2)), but covers $v_2, \cdots, v_k$ is M_{k-2}. Hence $M_k = M_{k-1} + M_{k-2}$. $\square$

Corollary A.15. *The value of M_k is the same as the $(k-1)$-th number in the Fibonacci series.*

The following Fact A.16 and Lemma A.17 used in the proof of Theorem A.12 have been proved in [168] (Fact 4.1 and Lemma A.1 respectively).

Fact A.16. [168] Let $f(x) = \sum_{i=0}^{d} a_i x^i$ be a polynomial with rational coefficients. If (α_0, β_0), $\cdots$, (α_d, β_d) are such that $f(\alpha_j) = \beta_j$, $j = 1$ to d, and all α_j-s are *distinct*, then all the coefficients a_i-s of f can be recovered in time polynomial in d and the maximum number of bits needed to represent α_j-s and β_j-s.

Lemma A.17. *[168] Let F_N denote the N-th Fibonacci number and let $r_N = \frac{F_{N+1}}{F_N}$. Then $r_i \neq r_j$ for any $i \neq j$.*

A.2.4 Step2: #P-hardness of #`RO`$\times$`RO-4Partite-4CNF`

Here we prove the following theorem.

Theorem A.18. *#`RO`$\times$`RO-4Partite-4CNF` is #P-hard.*

Given an instance of #4Partite-4BEC, we construct an instance of the problem #RO$\times$RO-4Partite-4CNF as follows (as done in [31]): Let the #4Partite-4BEC instance be $G(V_1, V_2, E)$. There is a variable x_e for every edge $e \in E$. The two DNF expressions ψ_1, ψ_2 correspond to bipartite vertex sets V_1, V_2 respectively. There is a clause $\sum_{e \ni v} x_e$ in ψ_1 (resp. ψ_2) for every vertex $v \in V_1$ (resp $v \in V_2$). Note that $E' \subseteq E$ is an edge-cover of G if and only if by assigning 1 to the variables x_e, $e \in E'$ and assigning 0 to the variables x_e, $e \notin E'$ we get a satisfying assignments of $\psi_1 \cdot \psi_2$. Hence given an oracle for computing $\mathtt{C}(\psi_1 \cdot \psi_2)$, the number of edge covers in G can be exactly computed.

By construction, both ψ_1, ψ_2 are positive and RO (since G is bipartite). Moreover, since degree of every vertex in V_1, V_2 is bounded by 4, the number of variables in every clause is also bounded by 4. The set of edges E can be partitioned into four disjoint matchings P_i ($i \in [1,4]$). However, two variables $x_e, x_{e'}$ co-occur in a clause in ψ_1 or ψ_2 if and only if e and e' have a common endpoint. Therefore, the variable set $\{x_e : e \in E\}$ can be partitioned into four groups X_i, where $X_i = \{x_e : e \in P_i\}$ ($i \in [1,4]$), such that no two variables from the same X_i will co-occur in a clause in ψ_1 or ψ_2. Finally, each edge has an endpoint in both V_1 and V_2, hence $\mathtt{Var}(V_1) = \mathtt{Var}(V_2)$. Hence $\psi_1 \cdot \psi_2$ is an instance of the #RO$\times$RO-4Partite-4CNF problem. Since the problem #4Partite-4BEC is #P-hard, #RO$\times$RO-4Partite-4CNF is also #P-hard.

A.2.5 Step3: #P-hardness of #RO$\times$RO-4Partite-4DNF

Here we show that the #RO$\times$RO-4Partite-4DNF problem is #P-hard by a reduction from #RO$\times$RO-4Partite-4CNF that uses the properties of RO expressions given by Lemma A.11.

Theorem A.19. *The #RO$\times$RO-4Partite-4DNF problem is #P-hard.*

Proof. Consider an instance of #RO$\times$RO-4Partite-4CNF, where we are given two positive RO CNF expressions ψ_1, ψ_2 that are defined on the variable set, such that every clause in ψ_1, ψ_2 has at most four variables. The goal is to compute $\mathtt{C}(\psi_1 \cdot \psi_2)$. Then using Lemma A.11, $\mathtt{C}(\overline{\psi_1} \cdot \overline{\psi_2})$ is #P-hard (otherwise given an oracle to compute $\mathtt{C}(\overline{\psi_1} \cdot \overline{\psi_2})$, $\mathtt{C}(\psi_1 \cdot \psi_2)$ can be exactly computed). Since ψ_1, ψ_2 are positive CNF RO expression where each clause has at most four positive variables, both $\overline{\psi_1}, \overline{\psi_2}$ are DNF RO expressions, where each term has at most four variables (all are negated literals).

From $\overline{\psi_1}, \overline{\psi_2}$, we construct two positive DNF RO expressions η_1, η_2, by making each negated variable in each term of $\overline{\psi_1}, \overline{\psi_2}$ positive in η_1, η_2 respectively. Hence η_1, η_2 are positive DNF RO expressions, where each term has at most four positive variables, and therefore $\eta_1 \cdot \eta_2$ is an instance of the `#RO×RO-4Partite-4DNF` problem. Let $N = |\texttt{Var}(\psi_1)| = |\texttt{Var}(\psi_2)| = |\texttt{Var}(\eta_1)| = |\texttt{Var}(\eta_2)|$. It is easy to verify that, $\mathbf{x} = \langle x_1, \cdots, x_N \rangle$ is a satisfying assignment of $\overline{\psi_1} \cdot \overline{\psi_2}$, if and only if $\mathbf{y} = \langle \overline{x_1}, \cdots, \overline{x_N} \rangle$ is a satisfying assignment of $\eta_1 \cdot \eta_2$. Hence there is a one-one correspondence between the satisfying assignments of $\overline{\psi_1} \cdot \overline{\psi_2}$ and $\eta_1 \cdot \eta_2$, and therefore $\texttt{C}(\overline{\psi_1} \cdot \overline{\psi_2}) = \texttt{C}(\eta_1 \cdot \eta_2)$. Combining the above two steps, if there is an oracle to solve `#RO×RO-4Partite-4DNF`, we can also solve `#RO×RO-4Partite-4CNF`. Since the problem `#RO×RO-4Partite-4CNF` is #*P*-hard, `#RO×RO-4Partite-4DNF` is also #*P*-hard. □

A.2.6 Step4: #*P*-hardness of $\Pr[q_1 - q_2]$

Finally we prove Step4 by reducing `#RO×RO-4Partite-4DNF` to probability computation of a query with difference.

Proof. Let $\eta_1 \cdot \eta_2$ be an instance of `#RO×RO-4Partite-4DNF`. To prove the #*P*-hardness of exact computation of $\Pr[\phi_1 \cdot \overline{\phi_2}]$ (equivalently $\texttt{C}(\phi_1 \cdot \overline{\phi_2})$ since all the uncertain variables x will have $\Pr[x] = \frac{1}{2}$) it suffices to construct two queries q_1, q_2 along with an instance of probabilistic database such that $\phi_1 = q_1(I) = \eta_1$ and $\phi_2 = q_2(I) = \eta_2$. This again follows from Lemma A.11, since given an oracle for $\texttt{C}(\phi_1 \cdot \phi_2)$, $\texttt{C}(\phi_1 \cdot \overline{\phi_2})$ can be exactly computed.

(1) First we extend the minterms (any term in an RO DNF expression is a minterm) with one, two or three variables to have exactly four variables by inserting dummy variables which always assume value 1 (in probabilistic database they correspond to deterministic tuples). Let the corresponding RO DNF expressions be η_1' and η_2'. Always fresh variables are inserted in every clause of η_1 and η_2 and therefore $\texttt{Var}(\eta_1') \neq \texttt{Var}(\eta_2')$. However, $\texttt{C}(\eta_1 \cdot \eta_2) = \texttt{C}(\eta_1' \cdot \eta_2')$ (the satisfying assignments have a one-one correspondence).

(2) Now recall that the variables $V = \texttt{Var}(\eta_1) = \texttt{Var}(\eta_2)$ can be partitioned into four disjoint groups $V_1, \cdots, V_4$ such that no two variables x, y from the same group V_i, $i \in [1,4]$

do not co-occur in a minterm in η_1 or η_2. Since we always inserted fresh variables in the minterms of η_1, η_2, this property also holds for η_1' and η_2' by arbitrarily assigning new variables to different groups. Hence, every minterm will have a variable from each of the groups $V_1, \cdots, V_4$.

(3) The probabilistic database instance I has four relations $R_1(A, B)$, $R_2(A, B)$, $R_3(A, B)$, $R_4(A, B)$, all with two attributes A and B, that contain tuples annotated with the variables in $X_1, \cdots, X_4$ respectively. Let the number of minterms in η_1' (resp. η_2') be m_1 (resp. m_2). We fill out the attribute values s of the four tables in the following way:

(a) Let the i-th minterm in η_1' be $\langle x_{i_1} y_{i_2} z_{i_3}, w_{i_4} \rangle$, where $x_{i_1} \in R_1$, $y_{i_2} \in R_2$, $z_{i_3} \in R_3$, and $w_{i_4} \in R_4$. Assign a_i as the values of attribute A in the i_1-th row of R_1, i_2-th row of R_2, i_3-th row of R_3 and i_4-th row of R_4, for all $i \in [1, m_1]$.

(b) Similarly, let the j-th minterm in η_2' be $\langle x_{j_1} y_{j_2} z_{j_3}, w_{j_4} \rangle$, where $x_{j_1} \in R_1$, $y_{j_2} \in R_2$, $z_{j_3} \in R_3$, and $w_{j_4} \in R_4$. Assign b_j as the values of B in the j_1-th row of R_1, j_2-th row of R_2, j_3-th row of R_3 and j_4-th row of R_4, for all $j \in [1, m_2]$.

(c) Now, due to the introduction of dummy variable, and possible unequal values of m_1, m_2, some of the attribute values in the tables $R_1, \cdots, R_4$ may be unassigned after the above two steps. Every such unassigned positions are filled with a fresh attribute value not used before.

(4) Let $N_i = |X_i|$ ($i \in [1, 4]$). Let the value of the tuple in the j-th row of $R_1, \cdots, R_4$ be a_j, b_j, c_j and d_j respectively, $j \in [1, N_i]$. The original variables in $V = \texttt{Var}(\eta_1) = \texttt{Var}(\eta_2)$ appear with probability $\frac{1}{2}$, whereas the new variables inserted in η_1' and η_2' appear with probability 1.

(5) Finally, $q_1() := R_1(x, y_1)\ R_2(x, y_2)\ R_3(x, y_3)\ R_4(x, y_4)$ and $q_2() := R_1(x_1, y)\ R_2(x_2, y)\ R_3(x_3, y)\ R_4(x_4, y)$.

Since exactly the tuples appearing in the same minterms of η_1', η_2' join in q_1, q_2 respectively, it easily follows that $q_1(I) = \eta_1'$ and $q_2(I) = \eta_2'$. Clearly, q_1 and q_2 are queries in CQ$^-$. Further, both Boolean queries q_1, q_2 are hierarchical (i.e. for every two variables x, y, the sets of subgoals that contain x, y are either disjoint or one is contained in the other) and therefore are safe [56]. $\square$

This completes the proof of Theorem 4.5.

A.2.7 Proof of Theorem 4.7

Here we show that in general no non-trivial approximation exists even for general SPJUD queries with difference rank = 1.

Proof. The reduction is from counting the number of independent sets in a graph. Given a graph $G(V,E)$, consider the relational schema (V,E,S) where $V(A), S(A)$ are unary relations while $E(A_1, A_2)$ is binary. V and E capture the vertex and edges in G, whereas S captures a subset of vertices, in particular, the set of possible worlds of I which correspond to an independent set in G. The tuple variables in V, E are deterministic, and appear with probability 1, whereas, every tuple variable in S appears with probability $\frac{1}{2}$. As we discussed in Section 4.2.3, the independence sets in a graph can be captured by the SPJUD query $q_{ind-set} = \mathit{True} - [E \bowtie \rho_{A_1/A} S \bowtie \rho_{A_2/A} S]$. Clearly, $q_{ind-set} = 1$.

Let ϕ be the boolean provenance of the unique tuple in $q_{ind-set}(I)$. Clearly, $\Pr[\phi] = \frac{N_{IS}}{2^n}$, where $n = |V|$ and N_{IS} is the number of independent sets in G. It is known that counting independent sets in an arbitrary graph does not have any non-trivial approximation unless $P = \mathcal{NP}$ [72]. This shows the inapproximability of tuple probabilities generated by SPJUD queries even if the query has difference rank 1 and proves Theorem 4.7. □

Remark: The above query $q_{ind-set}$ uses self-join. The hardness can be extended to queries without self-join under the weaker assumption that counting independent sets in bipartite graphs do not have any approximation [72].

A.3 Proofs from Chapter 5

A.3.1 NP-hardness for Recall Budget Constraint in Single Dictionary

We prove the following theorem:

Theorem A.20. *Maximization of the residual F-score for single dictionary refinement under recall budget constraint is NP-hard.*

We prove the NP-hardness via a reduction from the *subset-sum problem* which is known to be NP-hard [80]. In the subset-sum problem the input is a sequence of positive integers

$I = \langle x_1, \cdots, x_n \rangle$, and an integer C, and the goal is to decide if there is a subset $S \subseteq I$ such that $\sum_{x_i \in S} x_i = C$.

Construction. Given an instance of subset sum problem we reduce is to the following instance the dictionary refinement problem. In the dictionary refinement instance we create, $A = \{w_1, \cdots, w_n\}$. There are n dictionary entries $w_1, \cdots, w_n$ corresponding to the integers $x_1, \cdots, x_n$ such that each w_i has $f_{w_i} = cx_i$ and $p_{w_i} = \frac{1}{c}$ (hence $p_{w_i} f_{w_i} = x_i$). we choose $c = 3 + K$. In addition, there is a special entry w^* such that $f_{w^*} = 1$ and $p_{w^*} = 1$. The budget on the residual recall $\rho = \frac{1+B}{1+K}$. The goal is to check if there is a subset S' of the entries such that the residual F-score $F_{\overline{S'}} \geq \frac{2(1+B)}{(1+K)+(1+cB)}$ and the residual recall $R_{\overline{S'}} \geq \rho$. Next we argue that such a subset S' exists if and only if the instance of the subset-sum problem has a solution.

(if) Let the subset-sum instance have a solution S such that $\sum_{x_i \in S} x_i = C$, i.e. $\sum_{x_i \in I \setminus S} x_i = K - C = B$. We show that the subset of entries $S' = \{w_i : x_i \in S\}$ has the desired properties. First, $R_{\overline{S'}} = \frac{\sum_{w \notin S} p_w f_w}{\sum_{w \in A} p_w f_w} = \frac{1+\sum_{x_i \in I \setminus S} x_i}{1+\sum_{x_i \in I} x_i} = \frac{1+B}{1+K} = \rho$. Also $F_{\overline{S'}} = \frac{2\sum_{w \notin S} p_w f_w}{\sum_{w \in A} p_w f_w + \sum_{w \notin S} f_w} = 2\frac{1+\sum_{x_i \in I \setminus S} x_i}{(1+\sum_{x_i \in I} x_i)+(1+\sum_{x_i \in I \setminus S} cx_i)} = \frac{2(1+B)}{(1+K)+(1+cB)}$.

(only if) Suppose there is a subset S' of entries such that $F_{\overline{S'}} \geq \frac{2(1+B)}{(1+K)+(1+cB)}$ and $R_{\overline{S'}} \geq \rho = \frac{1+B}{1+K}$. Wlog., we can assume that $w^* \notin S'$, since otherwise we can exclude w^* from S' without decreasing the values of residual precision or residual recall (hence also the residual F-score). Hence all entries in S' corresponds to integers in the subset-sum problem, and Let the corresponding set of integers in the subset-sum problem be S. Now $R_{\overline{S'}} = \frac{\sum_{w \notin S} p_w f_w}{\sum_{w \in A} p_w f_w} = \frac{1+\sum_{x_i \in I \setminus S} x_i}{1+K} \geq \frac{1+B}{1+K}$, or,

$$\sum_{x_i \in I \setminus S} x_i \geq B \tag{A.5}$$

Again, $F_{\overline{S'}} = 2\frac{\sum_{w \notin S'} p_w f_w}{\sum_{w \in A} p_w f_w + \sum_{w \notin S'} f_w} = \frac{2(1+\sum_{x_i \in I \setminus S} x_i)}{(1+K)+(1+c\sum_{x_i \in I \setminus S} x_i)} \geq \frac{2(1+B)}{(1+K)+(1+cB)}$. Rearranging and simplifying both sides, $(c-2-K)\sum_{x_i \in I \setminus S} x_i \leq (c-2-K)B$. Since $c = 3 + K$,

$$\sum_{x_i \in I \setminus S} x_i \leq B \tag{A.6}$$

From (A.5) and (A.6), $\sum_{x_i \in I \setminus S} x_i = B$, or, $\sum_{x_i \in S} x_i = K - B = C$. Therefore $S \subseteq I$ is a solution for the subset-sum problem.

A.3.2 Update Rules for EM

Here we derive the update rules for the EM-based algorithm described in Section 5.3.3. Let us denote the parameter vector at iteration t to be $\vec{\theta}^t$. Suppose $c_{w_i,\tau_j,t} = \mathbb{E}[y_j^\ell | \tau_j, \vec{\theta}^t]$, where $\tau_j \in \texttt{Succ}(w_i)$ and $\texttt{Prov}(y_j^\ell) = w_i$. We show that the update rules for parameters p_i has a nice closed form: $p_i = \frac{C_1}{C_1+C_2}$, where $C_1 = \sum c_{w_i,\tau_j,t}$ and $C_2 = \sum(1 - c_{w_i,\tau_j,t})$, where the sum is over $1 \leq j \leq N$ such that $\tau_j \in \texttt{Succ}(w_i)$. These parameter values are considered to be $\vec{\theta^{t+1}}$, estimation of the parameters in the $t+1$-th round.

Derivation of the update rules. The log-likelihood of the observed data

$$q(\vec{x};\vec{\theta}) = \log P(\vec{x}|\vec{\theta}) = \sum_{j=1}^{N} P(\tau_i|\vec{\theta})$$

The complete data version of the problem will have the observed data $\vec{x} = \langle \tau_1, \cdots, \tau_N \rangle$ as well as the hidden data $\vec{\vec{y}} = \langle y_j^\ell \rangle_{j \in [1,N], b \in [1,\ell]}$. The expected log-likelihood of the complete data given the observed data $\vec{x}$ and current parameter vector $\vec{\theta}^t$ will be given by

$$\begin{aligned}
& \mathbb{E}[q(\vec{x},\vec{\vec{y}};\vec{\theta})|\vec{x},\vec{\theta}^t] \\
= & \sum_{j=1}^{N} \sum_{\vec{y}_j} \Pr[\vec{y}_j|\tau_j,\vec{\theta}^t] \log \Pr[\tau_j,\vec{y}_j|\vec{\theta}] \\
= & \sum_{j=1}^{N} \sum_{\substack{\vec{y}_j: \\ \phi_j(\vec{y}_j)=\tau_j}} \Pr[\vec{y}_j|\tau_j,\vec{\theta}^t] \log \Pr[\tau_j,\vec{y}_j|\vec{\theta}] \\
= & \sum_{j=1}^{N} \sum_{\substack{\vec{y}_j: \\ \phi_j(\vec{y}_j)=\tau_j}} \Pr[\vec{y}_j|\tau_j,\vec{\theta}^t] \log[\Pr[\tau_j|\vec{y}_j,\vec{\theta}] \Pr[\vec{y}_j|\vec{\theta}]] \\
= & \sum_{j=1}^{N} \sum_{\substack{\vec{y}_j: \\ \phi_j(\vec{y}_j)=\tau_j}} \Pr[\vec{y}_j|\tau_j,\vec{\theta}^t] \log \Pr[\vec{y}_j|\vec{\theta}] \\
= & K(say)
\end{aligned}$$

In the third step of the above derivation, for $\vec{y}_j$ such that $\phi_j(\vec{y}_j) \neq \tau_j, \Pr[\vec{y}_j|\tau_j,\vec{\theta}^t] = 0$, and in the fifth step, for $\vec{y}_j$ such that $\phi_j(\vec{y}_j) = \tau_j, \Pr[\tau_j|\vec{y}_j,\vec{\theta}] = 1$. Note that, given the current guess of parameters $\vec{\theta}^t = \langle p_1^t, \cdots, p_n^t \rangle$, $\Pr[\vec{y}_j|\tau_j,\vec{\theta}^t]$ can be easily computed and is a constant. For $\vec{y}_j$ such that $\phi_j(\vec{y}_j) = \tau_j$, $\Pr[\vec{y}_j|\tau_j,\vec{\theta}^t] = \frac{\Pr[\vec{y}_j|\vec{\theta}^t]}{\Pr[\tau_j|\vec{\theta}^t]} = \frac{\prod_{y_j^\ell=1} p^t_{\texttt{Prov}(y_j^\ell)} \prod_{y_j^\ell=0}(1-p^t_{\texttt{Prov}(z_j^\ell)})}{\sum_{\phi_j(\vec{z}_j)=\tau_j}\left[\prod_{z_j^\ell=1} p^t_{\texttt{Prov}(z_j^\ell)} \prod_{z_j^\ell=0}(1-p^t_{\texttt{Prov}(z_j^\ell)})\right]}$ (we slightly abuse the notation here: $p^t_{\texttt{Prov}(y^\ell_j)} = p^t_i$, where $\texttt{Prov}(y^\ell_j) = w_i$).

Next we rewrite K by expanding and collecting coefficients of $\log p_i$ and $\log(1 - p_i)$ for every word w_i, $i \in [1,n]$. Then the value of K

$$
\begin{aligned}
&= \sum_{j=1}^{N} \sum_{\substack{\vec{y}_j:\\ \phi_j(\vec{y}_j)=\tau_j}} \Pr[\vec{y}_j|\tau_j,\vec{\theta}^t] \log \Pr[\vec{y}_j|\theta] \\
&= \sum_{j=1}^{N} \sum_{\substack{\vec{y}_j:\\ \phi_j(\vec{y}_j)=\tau_j}} \Pr[\vec{y}_j|\tau_j,\vec{\theta}^t] \log[\prod_{\ell=1}^{\ell} \Pr[y^\ell_j|\theta]] \\
&= \sum_{j=1}^{N} \sum_{\substack{\vec{y}_j:\\ \phi_j(\vec{y}_j)=\tau_j}} \Pr[\vec{y}_j|\tau_j,\vec{\theta}^t] \log[\prod_{y^\ell_j=1} p_{\texttt{Prov}(y^\ell_j)} \prod_{y^\ell_j=0} (1 - p_{\texttt{Prov}(y^\ell_j)})] \\
&= \sum_{j=1}^{N} \sum_{\substack{\vec{y}_j:\\ \phi_j(\vec{y}_j)=\tau_j}} \Pr[\vec{y}_j|\tau_j,\vec{\theta}^t] [\sum_{y^\ell_j=1} \log[p_{\texttt{Prov}(y^\ell_j)}] \sum_{y^\ell_j=0} \log(1 - p_{\texttt{Prov}(y^\ell_j)})] \\
&= \sum_{i=1}^{n} \sum_{\substack{j=1,\tau_j\in\\ \texttt{Succ}(w_i)}}^{N} \sum_{\substack{\vec{y}_j:\\ \phi_j(\vec{y}_j)=\tau_j,\\ \texttt{Prov}(y^\ell_j)=w_i}} \Pr[\vec{y}_j|\tau_j,\vec{\theta}^t] \left[y^\ell_j \log p_i + (1 - y^\ell_j) \log(1 - p_i) \right] \\
&= \sum_{i=1}^{n} \sum_{\substack{j=1\\ \tau_j\in\ \texttt{Succ}(w_i)\\ \texttt{Prov}(y^\ell_j)=w_i}}^{N} \mathbb{E}[y^\ell_j|\tau_j,\vec{\theta}^t] \log p_i + (1 - \mathbb{E}[y^\ell_j|\tau_j,\vec{\theta}^t]) \log(1 - p_i) \\
&= \sum_{i=1}^{n} \sum_{\substack{j=1\\ \tau_j\in\ \texttt{Succ}(w_i)\\ \texttt{Prov}(y^\ell_j)=w_i}}^{N} c_{w_i,\tau_j,t} \log p_i + (1 - c_{w_i,\tau_j,t}) \log(1 - p_i)
\end{aligned}
$$

In the above equations, $c_{w_i,\tau_j,t} = \mathbb{E}[y^\ell_j|\tau_j,\vec{\theta}^t]$. In the E-step, for every word w_i, and for every occurrence $\tau_j \in \texttt{Succ}(w_i)$, we compute the expectation of y^ℓ_j (the ℓ-th bit of $\vec{y}_j$) given the current parameter vector $\vec{\theta}^t$, where $\texttt{Prov}(y^\ell_j) = w_i$. This can be computed from the probabilities $\Pr[\vec{y}_j|\tau_j,\vec{\theta}^t]$. So for every occurrence τ_j, if ϕ_j takes b inputs, we have a vector of real numbers of size b after the E-step.

In the M-step, we maximize the expression K w.r.t. parameter vector $\vec{\theta}$ to get the next guess of the parameters θ^{t+1}.

For every $i \in [1,n]$,

$$\begin{aligned}
&\frac{\delta K}{\delta p_i} = 0 \\
\Rightarrow &\sum_{\substack{j=1 \\ \tau_j \in \texttt{Succ}(w_i) \\ \texttt{Prov}(y_j^{\ell}) = w_i}}^{N} \left[\frac{c_{w_i,\tau_j,t}}{p_i} - \frac{1 - c_{w_i,\tau_j,t}}{1 - p_i}\right] = 0 \\
\Rightarrow &\frac{C_1}{p_i} - \frac{C_2}{1-p_i} = 0 \qquad \text{[collecting the constants]} \\
\Rightarrow &p_i = \frac{C_1}{C_1 + C_2}
\end{aligned}$$

A.3.3 Proof of Theorem 5.7

Theorem 5.7. *Maximization of the residual F-score for multiple dictionary refinement under size constraint is NP-hard even for the simple firstname-lastname rule* (i.e., *the rule R_4 in Figure 2.4).*

We give a reduction from the *k'-densest subgraph problem in bipartite graphs* which has been proved to be NP-hard in [48]. Here the input is a bipartite graph $H(U,V,E)$ with n' vertices and m' edges, and, an integer $k' < n'$. The goal is to select a subset of vertices $W \subseteq U \cup V$ such that $|W| = k'$ and the subgraph induced on W has the maximum number of edges. We will denote the set of edges in the *induced subgraph* on W (every edge in the subgraph has both its endpoints in W) by $E(W)$.

For simplicity, first we prove a weaker claim: removing a subset S such that the the size of S is *exactly k* (as opposed to *at most k*) is NP-hard. Intuitively, the vertices correspond to entries and the edges correspond to occurrences. We show that if the induced subgraph on a subset of vertices of size at most k' has a large number of edges, then removing entries in the *complement* of this subset results in this induced subgraph that gives a large residual F-score.

Construction for the weaker claim. Given an instance of the k'-densest subgraph problem, we create an instance of the dictionary refinement problem with two dictionaries

D_1, D_2 as follows. The vertices in U (resp. V) correspond to the entries in dictionary D_1 (resp. D_2). The dictionaries D_1, D_2 contain the entries in the firstname and lastname dictionaries in the firstname-lastname rule, *i.e.*, the rule R_4 in Figure 2.4. Every edge $(u,v) \in E$ corresponds to a unique provenance expression $\phi_{u,v} = uv$ for all occurrences produced by the join of u from D_1 and v from D_2. For each $(u,v) \in E$, there is a `Good` result and a `Bad` result. The parameter k in the dictionary refinement problem is $k = n' - k'$. We prove the following lemma:

Lemma A.21. *There is a subset $W \subseteq U \cup V$, such that $|W| = k'$ and $E(W) \geq q$ if and only if there is a subset S for the dictionary refinement problem such that $|S| = k$ and $F_{\overline{S}} \geq \frac{2}{\frac{m'}{q}+2}$.*

Proof. (if) Let W be a solution of k'-densest subgraph problem such that $|E(W)| \geq q$, $|W| = k'$. Choose S to be the dictionary entries corresponding to vertices in $(U \cup V) \setminus W$. Then $|S| = n' - k' = k$ as desired. Further for any edge $(u,v) \in H$, any result τ such that $\texttt{Prov}(\tau) = uv \in \texttt{surv}(S)$ if and only if both $u,v \in W$, i.e. $(u,v) \in E(W)$. Also recall that for any $(u,v) \in E$, there is a `Good` tuple τ_g with $\phi(\tau_g) = 1$ and a `Bad` tuple τ_b with $\phi(\tau_b) = 0$. Then $P_{\overline{S}} = \frac{\sum_{\tau \in \texttt{surv}(S)} \phi(\tau)}{|\texttt{surv}(S)|} = \frac{1}{2}$, and $R_{\overline{S}} = \frac{\sum_{\tau \in \texttt{surv}(S)} \phi(\tau)}{\sum_{\tau} \phi(\tau)} = \frac{|E(W)|}{m'} \geq \frac{q}{m'}$. Hence $F_{\overline{S}} = \frac{2}{1/P_{\overline{S}} + 1/R_{\overline{S}}} \geq \frac{2}{\frac{m'}{q}+2}$.

(only if) Let S be a solution of the dictionary refinement problem, such that $|S| = k$ and $F_{\overline{S}} \geq \frac{2}{\frac{m'}{q}+2}$. Choose $W = (U \cup V) \setminus W_S$, where W_S is the subset of vertices corresponding to S. Again, $|W| = n' - k = k'$. For subset S, $P_{\overline{S}} = \frac{1}{2}$ and $F_{\overline{S}} = \frac{2}{2+1/R_{\overline{S}}} \geq \frac{2}{\frac{m'}{q}+2}$. Therefore $R_{\overline{S}} = \frac{|E(W)|}{m'} \geq \frac{q}{m}$, i.e. $|E(W)| \geq q$ as desired. □

Proof of Theorem 5.7. Now we strengthen the above reduction to work for the relaxed constraint $|S| \leq k$.

Construction. Given an instance of the k'-densest subgraph problem, we create an instance of the dictionary refinement problem with two dictionaries D_1, D_2 as follows. (i) The vertices in U (resp. V) form a subset of the entries in dictionary D_1 (resp. D_2). For all results τ produced by the join of u from D_1 and v from D_2, where $(u,v) \in E$, $\texttt{Prov}(\tau) = uv$. For each $(u,v) \in E$, as before, there is a single `Good` tuple τ_g with $\phi(\tau_g) = 1$ and a `Bad` tuple τ_b with $\phi(\tau_b) = 0$.

(ii) In addition, we add s Good result tuples with provenance ab unrelated to anything where $s = m'n'$. These tuples will make the differences in the recall between solutions tiny (while preserving monotonicity in the size of $E(W)$). (iii) For every vertex $u \in U \cup V$ in the graph H, we add s Bad tuples with provenance uu^i. (iv) The parameter k in the dictionary refinement problem is $k = n' - k'$. The normalizing constant is $C = 2m' + n's + s$.

Lemma A.22. *There is a subset $W \subseteq U \cup V$, such that $|W| = k'$ and $E(W) \geq q$ if and only if there is a subset S for the dictionary refinement problem such that $|S| \leq k$ and $F_{\overline{S}} \geq \frac{2}{\frac{m'+(n'-k)s}{s+q}+2}$.*

Proof. (only if) Suppose there is such a solution W of the k'-densest subgraph problem. Select S to be the entries corresponding to $U \cup V \setminus W$ in the dictionary refinement instance created. Hence $|S| = n' - k' = k$. Also after removal of S, the tuples in induced subgraph on W, the auxiliary bad tuples incident on W, and the special good tuples with provenance ab will survive. Hence, residual recall = $R_{\overline{S}} = \frac{s+|E(W)|}{s+m'}$ and residual precision $P_{\overline{S}} = \frac{s+q}{s+2|E(W)|+(n'-k)s}$. Hence $F_{\overline{S}} \geq \frac{2(s+|E(W)|)}{(s+m')+(s+2|E(W)|+(n'-k)s)} = \frac{2(s+|E(W)|)}{m'+2(s+|E(W)|)+(n'-k)s} = \frac{2}{\frac{m'+(n'-k)s}{s+|E(W)|}+2}$. Since $|E(W)| \geq q$, $F_{\overline{S}} \geq \frac{2}{\frac{m'+(n'-k)s}{s+q}+2}$.

(if) We prove this direction in a few steps. Let S be the set of entries deleted, such that $|S| \leq k$ and $F_{\overline{S}} \geq \frac{2}{\frac{m'+(n'-k)s}{s+q}+2}$.

Claim 1. We can assume that wlog. that S contains only corresponding to vertices in $U \cup V$.

Note that the entries a or b occurring in the special Good tuples with provenance ab will never be deleted, since otherwise we can remove this entry without increasing the size of set S or decreasing the value of F-score. Also we can assume that if an entry $u \in U \cup V$ is in S, then none of the entries from its auxiliary bad tuples u^i are included in S. Let $u^i \in S$ for some $u \in U \cup V$. We show that the solution $S' = S - \{u^i\} \cup \{u\}$ has F-score at least as large as $F_{\overline{S}}$. Hence we can replace u^i by u and continue until S contains only vertices from H.

Let M be the number of Good tuples and N be the number of Bad tuples in $T = S \setminus \{u^i\}$.

Then

$$P_{\overline{S}} = \frac{s+M}{s+M+(N-1)}, \quad R_{\overline{S}} = \frac{s+M}{s+m'}, \quad F_{\overline{S}} = 2\frac{s+M}{(s+m')+(s+M+N-1)} \tag{A.7}$$

Suppose $m_u \leq n'$ be the number of edges incident on u. All entries of its auxiliary `Bad` tuples were not chosen before. Hence

$$P_{\overline{S'}} \geq \frac{s+M-m_u}{s+M+(N-s)}, \qquad R_{\overline{S'}} = \frac{s+M-m_u}{s+m'}$$

$$F_{\overline{S'}} = 2\frac{s+M-m_u}{(s+m')+(s+M+N-s)} \geq 2\frac{s+M-n'}{(s+m')+(s+M+N-s)} \tag{A.8}$$

We claim that $F_{\overline{S'}} \geq F_{\overline{S}}$. From (A.7) and (A.8) it suffices to show that

$$\begin{aligned} (s+M-n')(s+m'+(s+M+N-1)) &\geq (s+M)(s+m'+(s+M+N-s)) \\ or, \quad -s-M-n'(s+m'+s+M+N-1) &\geq -s(s+M) \\ or, \quad s(s+M-1-2n') &\geq M+(M+N)n'-n' \end{aligned}$$

and the last inequality holds since $s = m'n'$, whereas $M, N \leq 2m'$. Hence any such entry u^i from auxiliary `Bad` tuple can be replaced by u itself without decreasing the F-score value. So from now on we can assume that S only contains entries from the vertex set of H.

Claim 2. We can assume wlog. that the size of the set S is exactly k.

Let S be the current solution with $|S| = \ell < k$. Here we show that adding one more entry from the vertex set of H improves the F-score. Let u be a vertex not included in S, and let $S' = S \cup \{u\}$. Let M_1 be the number of edges in the induced subgraph on $U \cup V \setminus S$, and let M_2 be the number of edges in the induced subgraph on $U \cup V \setminus S'$. Then

$$P_{\overline{S}} = \frac{s+M_1}{s+2M_1+(n'-\ell)s}, \qquad R_{\overline{S}} = \frac{s+M_1}{s+m'}$$

$$F_{\overline{S}} = 2\frac{s+M_1}{(s+m')+(s+2M_1+(n'-\ell)s)} = \frac{2}{2+\frac{m'+(n'-\ell)s}{s+M_1}} \tag{A.9}$$

And,

$$P_{\overline{S'}} \geq \frac{s+M_2}{s+2M_2+(n'-\ell-1)s}, \qquad R_{\overline{S'}} = \frac{s+M_2}{s+m'}$$

$$F_{\overline{S'}} = 2\frac{s+M_2}{(s+m')+(s+2M_2+(n'-\ell-1)s)} = \frac{2}{2+\frac{m'+(n'-\ell-1)s}{s+M_2}} \tag{A.10}$$

Again we claim that $F_{\overline{S'}} \geq F_{\overline{S}}$. From (A.9) and (A.10) it suffices to show that

$$\begin{aligned} (s+M_2)(m'+(n'-\ell)s) &\geq (s+M_1)(m'+(n'-\ell-1)s) \qquad \text{(A.11)} \\ \text{or,} \quad M_2(m'+(n'-\ell)s) &\geq -s^2+M_1(m'+(n'-\ell)s)-M_1 s \\ \text{or,} \quad s^2+sM_1 &\geq (M_1-M_2)(m'+(n'-\ell)s) \end{aligned}$$

However, $M_1 \geq M_2 \geq M_1 - n'$, since deleting one additional vertex can delete at most n' edges. On the other hand $s = m'n'$. Hence inequality (A.12) holds (actually the inequality will be strict) and therefore, by deleting an entry corresponding to one additional vertex the residual F-score increases. We repeat this procedure until $|S| = k$.

Lower bound on induced edge set. So far we have shown that wlog. we can assume that $|S| = k$ and corresponds to vertices in H. Let W be the vertex set for the entries not in S. Then

$$P_{\overline{S}} = \frac{s+|E(W)|}{s+2|E(W)|+(n'-k)s}, \qquad R_{\overline{S}} = \frac{s+|E(W)|}{s+m'}$$

$$F_{\overline{S}} = 2\frac{s+|E(W)|}{(s+m')+(s+2|E(W)|+(n'-k)s)} = \frac{2}{2+\frac{m'+(n'-k)s}{s+|E(W)|}} \qquad \text{(A.12)}$$

Now, from the solution of the dictionary refinement problem,

$$\begin{aligned} F_{\overline{S}} &\geq \frac{2}{2+\frac{m'+(n'-k)s}{s+q}} \\ \Rightarrow \frac{2}{2+\frac{m'+(n'-k)s}{s+|E(W)|}} &= \frac{2}{2+\frac{m'+(n'-k)s}{s+q}} \\ \Rightarrow |E(W)| &\geq q \end{aligned}$$

Also $|W| = n' - k = k'$. This completes the proof of the lemma. ☐

Theorem 5.7 directly follows from Lemma A.22.

A.4 Proofs from Chapter 6

A.4.1 Proof of Theorem 6.9

Proof. We prove the theorem by a communication complexity reduction from the set disjointness problem: Suppose Alice and Bob own two subsets A and B of a universe U, $|U| = N$. To decide whether they have a common element (i.e. $A \cap B \neq \phi$) takes $\Omega(N)$ communications [115].

We construct the following relation R with $N+1$ rows for the module m. m has three input attributes: a, b, id and one output attribute y. The attributes a, b and y are Boolean, whereas id is in the range $[1, N+1]$. The input attribute id denotes the identity of every row in R and takes value $i \in [1, N+1]$ for the i-th row. The module m computes the AND function of inputs a and b, i.e., $y = a \wedge b$.

Row i, $i \in [1, N]$, corresponds to element $i \in U$. In row i, value of a is 1 iff $i \in A$; similarly, value of b is 1 iff $i \in B$. The additional $N+1$-th row has $a_{N+1} = 1$ and $b_{N+1} = 0$. The standalone privacy requirement $\Gamma = 2$ and the goal is to check if visible attributes $V = \{id, y\}$ (with hidden attributes $\overline{V} = \{a, b\}$) is safe for this privacy requirement.

Note that if there is a common element $i \in A \cap B$, then there are two y values in the table: in the i-th row, the value of $y = a \wedge b$ will be 1, whereas, in the $N+1$-th row it is 0. Hence hiding $\overline{V} = \{a, b\}$ will ensure the privacy requirement of $\Gamma = 2$ (every input $\mathbf{x}$ to m can be mapped either to 0 or 1). If there is no such $i \in A \cap B$, the value of y in all rows $i \in [1, N+1]$ will be zero which does not meet the privacy requirement $\Gamma = 2$. Hence we need to look at $\Omega(N)$ rows to decide whether $V = \{id, y\}$ is safe. □

A.4.2 Proof of Theorem 6.10

In our reduction, $N = 2^{k-1}$. Hence, alternatively, if N is the number of tuples in the relation, there does not exists a $poly(\log N)$ algorithm, unless P = NP.

Proof. We prove the theorem by a reduction from UNSAT: Suppose we are given a Boolean CNF formula g on ℓ Boolean variables $x_1, \cdots, x_\ell$. The goal is to decide if *no* assignment of $x_1, \cdots, x_\ell$ can satisfy g. Given such an UNSAT instance, we build a relation R with input attributes $x_1, \cdots, x_\ell, y$ and output attribute z, all of Boolean domain (hence $k = \ell + 2$).

The function of m has a succinct description as follows: $m(x_1, \cdots, x_\ell, y) = \neg g(x_1, \cdots, x_\ell) \wedge \neg y$ (i.e., NOR of $g(x_1, \cdots, x_\ell)$ and y). Hence in the table R, *implicitly*, we have two tuples for each assignment to the variables, $x_1, \cdots, x_\ell$: if the assignment for $x_1, \cdots, x_\ell$ satisfies the formula g then, for both $y = 0$ and $y = 1$, we have $z = 0$. Otherwise if the assignment does not satisfy the formula, for $y = 0$, we have $z = 1$, and for $y = 1$, $z = 0$. The privacy requirement is $\Gamma = 2$ and the goal is to decide if visible attributes $V = \{x_1, \cdots, x_\ell\} \cup \{z\}$ is a safe subset where hidden attributes $\overline{V} = \{y\}$.

Note that if the formula g is *not* satisfiable, then it suffices to hide y to get 2-privacy, i.e. V is safe. This is because for *every* satisfying assignment, there are two ways to complete y value (one that is the correct one and one that is the opposite). On the other hand, if the function g has at least one satisfying assignment, for that assignment in R,regardless of the value of the hidden attribute y, the output z has to always be 0. In that case V is not a safe subset. □

A.4.3 Proof of Theorem 6.11

Proof. Assume, for the sake of contradiction, that an algorithm Algo exists which uses $2^{o(k)}$ oracle calls. We will build an adversary that controls the `Safe-View` oracle and outputs answers to the queries consistent with a fixed function m_1 and a dynamically changing function m_2 that depends on the set of queries asked. The minimum cost of a safe subset for m_1 will be 3/2 times that for (all definitions of) m_2, thereby proving the theorem.

Consider a function with ℓ Boolean input attributes in I, and one output attribute in O where ℓ is even (i.e. $k = \ell + 1$). The costs of all attributes in I is 1, the cost of attribute y in O is ℓ. We want to decide whether there exists a safe visible subset V such that the cost of the hidden subset $\overline{V}$ is at most $C = \frac{\ell}{2}$, or all hidden subsets have cost at least $\frac{3C}{2} = \frac{3\ell}{4}$. Hence, any such set $\overline{V}$ can never include the output attribute.

The oracle behaves as follows:

(P1) The oracle answers Yes for every set V of input attributes s.t. $|V| < \frac{\ell}{4}$ (i.e. $|\overline{V}| > \frac{3\ell}{4}$), and

(P2) The oracle answers No for every subset V of input attributes s.t. $|V| \geq \frac{\ell}{4}$ (i.e. $|\overline{V}| \leq \frac{3\ell}{4}$).

The functions m_1 and m_2 are defined as follows:

- m_1 returns 1 iff the total number of input attributes whose value is 1 is at least $\frac{\ell}{4}$ (and otherwise 0),

- m_2 has a special set A such that $|A| = \frac{\ell}{2}$. It returns 1 iff the total number of input attributes whose value is 1 is at least $\frac{\ell}{4}$ and there is at least one input attribute outside A whose value is 1 (and otherwise 0).

Note that while the cheapest safe subset for m_1 has cost greater than $\frac{3\ell}{4}$, m_2 has a safe subset A where the cost of $\overline{A}$ is $\frac{\ell}{2}$.

It remains to show that the behavior of the oracle (i.e. properties (P1) and (P2)) remains consistent with the definitions of m_1 and m_2. We consider m_1 first.

- (P1) holds for m_1: An all-0 $\overline{V}$ and an all-1 $\overline{V}$ respectively imply an answer of 0 and 1 independent of the assignment of V.

- (P2) holds for m_1: An all-1 V implies an answer of 1 independent of the assignment of $\overline{V}$.

Now, we consider m_2.

- (P1) holds for m_2: An all-0 $\overline{V}$ and an all-1 $\overline{V}$ respectively imply an answer of 0 and 1 independent of the assignment of V or the definition of A (in the first case, number of 1 is $< \frac{\ell}{4}$ and in the second case the number of 1 is $\geq \frac{3\ell}{4} > \frac{\ell}{4}$ and there is one 1 outside A since $\frac{3\ell}{4} > \frac{\ell}{2}$).

- (P2) holds for m_2 as long as V is not a subset of A, since an all-1 V will imply an answer of 1 independent of the assignment of $\overline{V}$. Therefore, such a query restricts the possible candidates of A, and discards at most $\binom{3\ell/4}{\ell/4}$ candidates of A.

Since there are $\binom{\ell}{\ell/2}$ possible definitions of A overall, the number of queries required to certify the absence of A (i.e. certify that the function is indeed m_1 and not m_2 with some definition of A) is at least

$$\frac{\binom{\ell}{\ell/2}}{\binom{3\ell/4}{\ell/4}} = \prod_{i=0}^{\ell/2-1} \frac{\ell - i}{3\ell/4 - i} \geq (4/3)^{\ell/2} = 2^{\Omega(k)}.$$

Therefore, for a $2^{o(k)}$-restricted algorithm Algo , there always remains at least one subset A defining a function m_2 that is consistent with all previous answers to queries.

Hence after $2^{o(k)}$ calls, if the algorithm decides that there is a safe subset with cost $< C$, we choose m to be m_1; on the other hand, if it says that there is no such subset, we set $m = m_2$ (with the remaining consistent subset of size $\frac{\ell}{2}$ as its special subset A). In both the cases the answer of the algorithm is wrong which shows that there cannot be such an algorithm distinguishing these two cases with $2^{o(k)}$ calls. $\square$

Remark. The above construction also shows that given a cost limit C, deciding whether there exists a safe subset V with cost at most C or all safe subsets have cost at least $\frac{3C}{2}$ requires $2^{\Omega(k)}$ oracle calls. By adjusting the parameters in this construction, the gap can be increased to a factor of $\Omega(k^{1/3})$ from a constant. More specifically, it can be shown that deciding whether there exists a safe subset with cost at most C, or whether for all safe subsets the cost is at least $\Omega(k^{1/3}C)$ requires $2^{\Omega(k^{1/6})}$ calls to the `Safe-View` oracle.

A.4.4 Proof of Proposition 6.13

We construct a simple workflow with two modules m_1, m_2 joined back to back as a chain. Both m_1, m_2 are one-one functions with k Boolean inputs and k Boolean outputs (for example, assume that m_1 is an identity function, whereas m_2 reverses the values of its k inputs). The module m_1 gets initial input attribute set I_1, produces $O_1 = I_2$ which is fed to the module m_2 as input, and m_2 produces final attribute set O_2. Let V_1 be an arbitrary subset of O_1 such that $|\overline{V_1}| = \log \Gamma$ (for simplicity, we assume that Γ is a power of 2). It can be easily verified that, m_1 as a standalone module is Γ-standalone-private w.r.t. visible attributes V_1 and both m_1, m_2 are Γ-workflow-private w.r.t. visible attributes V_1 (since m_1, m_2 are one-one modules).

Next we discuss how the one-one nature of m_1 and m_2 restricts the size of $\texttt{Worlds}(R, V_1)$ compared to that of $\texttt{Worlds}(R_1, V_1)$. Since both m_1 and m_2 are one-one functions, the workflow W also computes a one-one function. Hence any relation S in $\texttt{Worlds}(R, V_1)$ has to compute a one-one function as well. But when m_1 was standalone, any This in turn implies that the projection $\pi_{I_1 \cup O_1}(S)$ on $I_1 \cup O_1$ for any such relation S has to be one-one as well, otherwise S cannot compute a one-one function (S has to satisfy the functional dependencies $I_1 \to O_1$ and $I_2 \to O_2$). However, both I_1 and O_2 are visible and $S \in \texttt{Worlds}(R, V_1)$, i.e., $\pi_{V_1}(S) = \pi_{V_1}(R)$. Therefore fixing the attribute values in

$\pi_{I_1 \cup O_1}(S)$ also fixes the relation S. Hence the number of relations in $\texttt{Worlds}(R, V_1)$ is exactly the same as the number of relations S' over $I_1 \cup O_1$ such that (1) S' computes a one-one function from I_1 to O_1, and, (2) $\pi_{(I_1 \cup O_1) \cap V_1}(S') = \pi_{(I_1 \cup O_1) \cap V_1}(R_1)$. On the other hand, $\texttt{Worlds}(R_1, V_1)$ will be all possible relations S' on $I_i \cup O_i$ such that only (2) holds.

Let us first exactly compute $|\texttt{Worlds}(R_1, V_1)|$. Given an input to m_1, the visible output bits in V_1 are fixed; however, the hidden output bits in $\overline{V_1}$ can have arbitrary values. Since $|\overline{V_1}| = \log \Gamma$, any input to m_1 can be mapped to one of Γ different outputs. There are 2^k different inputs to m_1, and any relation $S' \in \texttt{Worlds}(R_1, V_1)$ is an arbitrary combination of the mappings for individual inputs. Hence $|\texttt{Worlds}(R_1, V_1)| = \Gamma^{2^k}$.

Next we compute $|\texttt{Worlds}(R, V_1)|$ which is the same as the number of one-one mappings for the module m_1 with the same values of the visible bits. Let us partition the set of 2^k different values of initial inputs to m_1 into $2^k/\Gamma$ groups, where all Γ initial inputs in a group produce the same values of visible intermediate attributes V_1. Any relation $S \in \texttt{Worlds}(R_1, V_1)$ has to map the input tuples in each such group to Γ *distinct* intermediate tuples. Hence S must permute the Γ intermediate tuples corresponding to a group of Γ input tuples.

Thus, the total number of relations in $\texttt{Worlds}(R, V_1)$ is $(\Gamma!)^{2^k/\Gamma} \simeq ((2\pi\Gamma)^{1/2\Gamma}(\Gamma/e))^{2^k}$ by Stirling's approximation (the input tuples in a group can map to one of $\Gamma!$ permutations, there are $2^k/\Gamma$ groups which can map independently). Hence the ratio of $|\texttt{Worlds}(R, V_1)|$ and $|\texttt{Worlds}(R_1, V_1)|$ is $\left(\frac{(2\pi\Gamma)^{1/2\Gamma}}{e}\right)^{2^k} < 1.4^{-2^k}$ for any $\Gamma \geq 2$ [43].

A.4.5 Proof of Lemma 6.15

Proof. If $\mathbf{y} \in \texttt{OUT}_{x,m_i}$ w.r.t. visible attributes V_i, then from Definition 6.3,

$$\exists R' \in \texttt{Worlds}(R, V_i),\ \ \exists t' \in R' \ \ s.t\ \ \mathbf{x} = \pi_{I_i}(\mathbf{t}') \wedge \mathbf{y} = \pi_{O_i}(\mathbf{t}') \tag{A.13}$$

Further, from Definition 6.1, $R' \in \texttt{Worlds}(R, V_i)$ only if $\pi_{V_i}(R_i) = \pi_{V_i}(R')$. Hence there must exist a tuple $\mathbf{t} \in R_i$ such that

$$\pi_{V_i}(\mathbf{t}) = \pi_{V_i}(\mathbf{t}') \tag{A.14}$$

[43]For $x > 1$, $x^{1/x}$ is a decreasing function and $\frac{e}{(\pi x)^{1/x}} \geq 1.4$

Let $\mathbf{x}' = \pi_{I_i}(\mathbf{t}')$ and $\mathbf{y}' = \pi_{O_i}(\mathbf{t}')$, i.e. $\mathbf{y}' = m_i(\mathbf{x}')$. Then by definition of $\mathbf{x}, \mathbf{y}, \mathbf{x}', \mathbf{y}'$ and from (A.14), $\pi_{V_i \cap I_i}(\mathbf{x}) = \pi_{V_i \cap I_i}(\mathbf{x}')$ and $\pi_{V \cap O_i}(\mathbf{y}) = \pi_{V \cap O_i}(\mathbf{y}')$. □

A.4.6 Proof of Lemma 6.14

First we introduce some additional notations. Recall that the relation R for workflow W is defined on the attribute set (or vector) A. For a tuple $\mathbf{x}$, we will use $\mathbf{x}(Q)$ to denote that the tuple $\mathbf{x}$ is defined on the attribute subset $Q \subseteq A$; For an attribute $a \in Q$, $x[a]$ will denote the value of the attribute a in $\mathbf{x}$, i.e. $x[a] = \pi_a(\mathbf{x})$.

Let $\mathbf{x}(P)$ be a tuple defined on an arbitrary attribute subset $P \subseteq A$ and $\mathbf{p}(Q), \mathbf{q}(Q)$ be two tuples defined on another arbitrary subset $Q \subseteq A$. Then, the tuple $\mathbf{y} = \text{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{x})$ defined on attribute subset P is defined as

$$y[a] = \begin{cases} q[a] & \text{if } a \in Q \text{ and } x[a] = p[a] \\ p[a] & \text{if } a \in Q \text{ and } x[a] = q[a] \\ x[a] & \text{otherwise.} \end{cases}$$

Intuitively, if the input tuple $\mathbf{x}$ shares the same value of some common attribute $a \in P \cap Q$ with that of $\mathbf{p}$ (i.e. $x[a] = p[a]$), the flip operation replaces the attribute value $x[a]$ by $q[a]$ in $\mathbf{x}$, whereas, if $x[a] = q[a]$, it replaces the value $x[a]$ by $p[a]$. If $a \in P \setminus Q$, or, if for some $a \in P \cap Q$, $x[a] \neq p[a]$ and $x[a] \neq q[a]$, $x[a]$ remains unchanged. If $x[a] = p[a] = q[a]$, then also the value of $x[a]$ remains unchanged.

It is easy to see that $\text{FLIP}_{\mathbf{p},\mathbf{q}}(\text{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{x})) = \mathbf{x}$. In the proof of the lemma, we will also use the notion of *function flipping*, which first flips the input on $\mathbf{p}, \mathbf{q}$, then applies the function on the flipped input, and then flips the output again on $\mathbf{p}, \mathbf{q}$. The formal definition is as follows.

Definition A.23. *Consider a module m mapping attributes in I to attributes in O. Let $\mathbf{p}(P), \mathbf{q}(P)$ be two tuples defined on attribute subset $P \subseteq A$. Then, $\forall \mathbf{x}(X)$ defined on X, $\text{FLIP}_{m,\mathbf{p},\mathbf{q}}(\mathbf{x}) = \text{FLIP}_{\mathbf{p},\mathbf{q}}(m(\text{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{x})))$.*

Now we complete the proof of Lemma 6.14.

Proof of Lemma 6.14.

Proof. If a module m_i is Γ-workflow-private w.r.t. visible attributes V_i, then from Proposition 6.7, m_i is also Γ-workflow-private w.r.t. any $V_i' \subseteq V_i$ (or equivalently, $\overline{V_i'} \supseteq \overline{V_i}$). Hence we will prove Lemma 6.14 for $V = V_i$.

Consider module m_i with relation R_i, input tuple $\mathbf{x}$ and visible attribute subset V_i as stated in Lemma 6.14. Let $\mathbf{y} \in \texttt{OUT}_{x,m_i}$. We will prove $\mathbf{y} \in \texttt{OUT}_{x,W}$, by showing the existence of a relation $R' \in \texttt{Worlds}(R,V)$ and a tuple $\mathbf{t'} \in R'$ such that $\mathbf{x} = \pi_{I_i}(\mathbf{t'}) \wedge \mathbf{y} = \pi_{O_i}(\mathbf{t'})$ (ref. Definition 6.6).

Since $\mathbf{y} \in \texttt{OUT}_{x,m_i}$, by Lemma 6.15, there are $\mathbf{x'} \in \pi_{V_i \cap I_i}(R_i)$, $\mathbf{y'} = m_i(\mathbf{x'})$ such that

$$\pi_{V_i \cap I_i}(\mathbf{x}) = \pi_{V_i \cap I_i}(\mathbf{x'}), \pi_{V_i \cap O_i}(\mathbf{y}) = \pi_{V_i \cap O_i}(\mathbf{y'}) \tag{A.15}$$

Let $\mathbf{p}(I_i \cup O_i), \mathbf{q}(I_i \cup O_i)$ be two tuples on $I_i \cup O_i$ where

$$p[\ell] = \begin{cases} x[\ell] & \text{if } \ell \in I_i \\ y[\ell] & \text{if } \ell \in O_i \end{cases} \text{ and } q[\ell] = \begin{cases} x'[\ell] & \text{if } \ell \in I_i \\ y'[\ell] & \text{if } \ell \in O_i. \end{cases}$$

(Recall that $I_i \cap O_i = \phi$). Hence $\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{x'}) = \mathbf{x}$ and $\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{y'}) = \mathbf{y}$. It should be notes that $\mathbf{x}, \mathbf{x'}$ and $\mathbf{y}, \mathbf{y'}$ have the same values on visible attribute subsets $I_i \cap V$ and $O_i \cap V$ respectively. So $\mathbf{p}$ and $\mathbf{q}$ only differ on the hidden attributes. Therefore, for any two tuples $\mathbf{w}, \mathbf{z}$, if $\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{w}) = \mathbf{z}$, then $\mathbf{w}$ and $\mathbf{z}$ will also only differ on the hidden attributes and their visible attribute values are the same.

For each $j \in [1,n]$, we define $g_j = \texttt{FLIP}_{m_j,\mathbf{p},\mathbf{q}}$. Then the desired relation $R' \in \texttt{Worlds}(R,V)$ is obtained by collecting executions of the workflow where every module m_i is replaced by module g_i, $i \in [1,n]$. So we need to show that (i) there is a tuple $\mathbf{t} \in S$, such that $\pi_{I_i}(\mathbf{t}) = \mathbf{x}$ and $\pi_{O_i}(\mathbf{t}) = \mathbf{y}$, and, (ii) $R' \in \texttt{Worlds}(R,V)$.

(i): To show the existence of such a tuple $\mathbf{t} \in R'$, it suffices to show that $g_i(\mathbf{x}) = \mathbf{y}$, since then for any tuple $\mathbf{t} \in R'$, if $\pi_{I_i}(\mathbf{t}) = \mathbf{x}$, then $\pi_{O_i}(\mathbf{t}) = \mathbf{y}$. We claim that g_i maps $\mathbf{x}$ to $\mathbf{y}$ as desired. This holds since $g_i(\mathbf{x}) = \texttt{FLIP}_{m_i,\mathbf{p},\mathbf{q}}(\mathbf{x}) = \texttt{FLIP}_{\mathbf{p},\mathbf{q}}(m_i(\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{x}))) = \texttt{FLIP}_{\mathbf{p},\mathbf{q}}(m_i(\mathbf{x'})) = \texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{y'}) = \mathbf{y}$.

(ii): Since every g_j is a function, R' satisfies all functional dependencies $I_i \to O_i$, $i \in [1,n]$. Hence to prove $R' \in \texttt{Worlds}(R,V)$, it suffices to show that, for the same *initial inputs* in R and R', the values of all the visible attributes in R and R' are the same. Let I_0 be the set of initial inputs to workflow W. We need to show that for any two tuples $\mathbf{t} \in R$

and $\mathbf{t}' \in R'$ on attribute set A, if $\pi_{I_0}(\mathbf{t}) = \pi_{I_0}(\mathbf{t}')$, then $\mathbf{t}, \mathbf{t}'$ also have the same values on the visible attributes V, i.e., $\pi_V(\mathbf{t}) = \pi_V(\mathbf{t}')$.

Let us fix any arbitrary tuple $\mathbf{p}$ on input attributes I_0. Let us assume, wlog., that the modules $m_1, \cdots, m_n$ (and corresponding $g_1, \cdots, g_n$) are ordered in a topological sorted order in the DAG W. Since I_0 is essentially the key attributes of relation R or R', there are two unique tuples $\mathbf{t} \in R$ and $\mathbf{t}' \in R'$ such that $\pi_{I_0}(\mathbf{t}) = \pi_{I_0}(\mathbf{t}') = \mathbf{p}$. Note that any intermediate or final attribute $a \in A \setminus I_0$ belongs to O_j, for a unique $j \in [1, n]$ (since for $j \neq \ell$, $O_j \cap O_\ell = \phi$). We prove by induction on j that the values of the visible attributes $O_j \cap V$ are the same for $\mathbf{t}$ and $\mathbf{t}'$ for every j. Together with the fact that the values of the attributes in I_0 are the same in $\mathbf{t}, \mathbf{t}'$, this shows that $\pi_V(\mathbf{t}) = \pi_V(\mathbf{t}')$.

Let $\mathbf{c_{j,f}}, \mathbf{c_{j,g}}$ be the values of input attributes I_j and $\mathbf{d_{j,f}}, \mathbf{d_{j,g}}$ be the values of output attributes O_j of module m_j in $\mathbf{t} \in R$ and $\mathbf{t}' \in R'$ respectively on initial input attributes $\mathbf{p}$ (i.e. $\mathbf{c_{j,f}} = \pi_{I_j}(\mathbf{t})$, $\mathbf{c_{j,g}} = \pi_{I_j}(\mathbf{t}')$, $\mathbf{d_{j,f}} = \pi_{O_j}(\mathbf{t})$ and $\mathbf{d_{j,g}} = \pi_{O_j}(\mathbf{t}')$). Then, we prove that $\mathbf{d_{j,g}} = \texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{d_{j,f}})$.

From (A.15), $\mathbf{x}, \mathbf{x}'$, and $\mathbf{y}, \mathbf{y}'$ have the same values of the visible attributes. Therefore the tuples $\mathbf{p}$ and $\mathbf{q}$ only differ in hidden attributes. Then if the above claim is true, for every j, $\mathbf{d_{j,g}}$ and $\mathbf{d_{j,f}}$ are the same on the visible attributes $O_i \cap V$. Equivalently, $\mathbf{t}$ and $\mathbf{t}'$ have the same values of visible attributes V as desired.

Note that if the inductive hypothesis holds for all $j' < j$, then $\mathbf{c_{j,g}} = \texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{c_{j,f}})$, since the modules are listed in a topological order. Thus,

$$\begin{aligned}
\mathbf{d_{j,g}} &= g_j(\mathbf{c_{j,g}}) = \texttt{FLIP}_{m_j,\mathbf{p},\mathbf{q}}(\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{c_{j,f}})) \\
&= \texttt{FLIP}_{\mathbf{p},\mathbf{q}}(m_j(\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{c_{j,f}})))) \\
&= \texttt{FLIP}_{\mathbf{p},\mathbf{q}}(m_j(\mathbf{c_{j,f}})) = \texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{d_{j,f}}).
\end{aligned}$$

Hence the hypothesis $\mathbf{d_{j,g}} = \texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{d_{j,f}})$ also holds for module m_j. This completes the proof of this lemma. □

A.4.7 Proof of Theorem 6.18

In `Secure-View` problem with cardinality constraint, as stated in Section 6.4.2, every module m_i, $i \in [1, n]$, has a requirement list of pair of numbers $L_i = \{(\alpha_i^j, \beta_i^j) : \alpha_i^j \leq |I_i|, \beta_i^j \leq$

$|O_i|, j \in [1, \ell_i]\}$. The goal is to select a safe subset of attributes V with minimum cost $c(\overline{V})$, such that for every $i \in [1, n]$, at least α_i^j input attributes and β_i^j output attributes of module m_i are hidden for some $j \in [1, \ell_i]$.

In this section, we prove Theorem 6.18. First we give an $O(\log n)$-approximation algorithm, and then show that this problem is $\Omega(\log n)$-hard under standard complexity-theoretic assumptions, even if the cost of hiding each data is identical, and the requirement list of every module in the workflow contains exactly one pair of numbers with values 0 or 1.

A.4.7.1 $O(\log n)$-Approximation Algorithm

Our algorithm is based on rounding the fractional relaxation (called the LP relaxation) of the integer linear program (IP) for this problem presented in Figure 6.2.

One can write a simpler IP for this problem, where the summations are removed from constraints (6.4) and (6.5), and constraints (6.6) and (6.7) are removed altogether. To see the necessity of these constraints, consider the LP relaxation of the IP, obtained by replacing constraint (6.8) with $x_b, r_{ij}, y_{bij}, z_{bij} \in [0, 1]$.

Suppose constraints (6.6) and (6.7) were missing from the IP, and therefore from the LP as well. For a particular $i \in [1, n]$, it is possible that a fractional solution to the LP has $r_{ij} = 1/2$ for two distinct values j_1 and j_2 of j, where $\alpha_{ij_1} > \alpha_{ij_2}$ and $\beta_{ij_1} < \beta_{ij_2}$. But constraint (6.2) (resp., constraint (6.3)) can now be satisfied by setting $y_{bij_1} = y_{bij_2} = 1$ (resp., $z_{bij_1} = z_{bij_2} = 1$) for $\alpha_{ij_1}/2$ input data (resp., $\beta_{ij_2}/2$ output data). However, $(\alpha_{ij_1}/2, \beta_{ij_2}/2)$ might not satisfy the privacy requirement for i, forcing an integral solution to hide some data b with $x_b = 0$. This will lead to an unbounded integrality gap.

Now, suppose constraints (6.2) and (6.3) did not have the summation. For a particular $i \in [1, n]$, it is possible that a fractional solution to the LP has $r_{ij} = 1/\ell_i$ for all $j \in [1, \ell_i]$. Constraint (6.2) (resp., constraint (6.3)) can then be satisfied by setting $y_{bij} = 1/\ell_i$ for all $1 \leq \ell_i$, for $\max_j\{\alpha_i^j\}$ distinct input data (resp., $\max_j\{\beta_i^j\}$ distinct output data). Correspondingly, $x_b = 1/\ell_i$ for those data b. If all the α_i^js and β_i^js for different $j \in [1, \ell_i]$ have similar values, it would mean that we are satisfying the privacy constraint for m_i paying an ℓ_i fraction of the cost of an integral solution. This can be formalized to yield an integrality gap of $\max_i\{\ell_i\}$, which could be n. Introducing the summation in the LP precludes this possibility.

Analysis. We can assume wlog that the requirement list L_i for each module m_i is non-redundant, i.e. for all $1 \leq j_1 \neq j_2 \leq \ell_i$, either $\alpha_{ij_1} > \alpha_{ij_2}$ and $\beta_{ij_2} < \beta_{ij_1}$, or $\alpha_{ij_1} < \alpha_{ij_2}$ and $\beta_{ij_2} > \beta_{ij_1}$. We can thus assume that for each module m_i, the list L_i is sorted in increasing order on the values of α_i^j and in decreasing order on the values of β_i^j. The following lemma shows that step 2 satisfies the privacy requirement of each module with high probability.

Lemma 6.19 *Let m_i be any module in workflow W. Then with probability at least $1 - 2/n^2$, there exists a $j \in [1, \ell_i]$ such that $|I_i^h| \geq \alpha_i^j$ and $|O_i^h| \geq \beta_i^j$.*

Proof. Given a fractional solution to the LP relaxation, let $p \in [1, \ell_i]$ be the index corresponding to the *median* (α_i^j, β_i^j), satisfying $\sum_{j=1}^{p-1} r_{ij} < 1/2$ and $\sum_{j=1}^{p} r_{ij} \geq 1/2$. We will show that after step 2, at least α_{ip} input data and β_{ip} output data is hidden with probability at least $1 - 2/n^2$ for module m_i.

We partition the set of data A into two sets: the set of data deterministically included in B, $B^{det} = \{b : x_b \geq 1/16 \log n\}$, and the set of data probabilistically rounded, $B^{prob} = A \setminus B^{det}$. Also, let $B^{round} = B \setminus B^{det}$ be the set of data that are actually hidden among B^{prob}. For each module m_i, let $I_i^{det} = B^{det} \cap I_i$ and $O_i^{det} = B^{det} \cap O_i$ be the set of hidden input and output data in B^{det} respectively. Let the size of these sets be $\alpha_i^{det} = |I_i^{det}|$ and $\beta_i^{det} = |O_i^{det}|$. Also, let $I_i^{prob} = B^{prob} \cap I_i$ and $O_i^{prob} = B^{prob} \cap O_i$. Finally, let $I_i^{round} = B^{round} \cap I_i$ and $O_i^{round} = B^{round} \cap O_i$. We show that for any module m_i, $|I_i^{round}| \geq \alpha_{ip} - \alpha_i^{det}$ and $|O_i^{round}| \geq \beta_{ip} - \beta_i^{det}$ with probability at least $1 - 1/n^2$.

First we show that $\sum_{b \in I_i^{prob}} x_b \geq (\alpha_{im} - \alpha_i^{det})/2$. Constraint (6.2) implies that $\sum_{b \in I_i} y_{bij} \geq r_{ij}\alpha_i^j$, while constraint (6.6) ensures that $\sum_{b \in I_i^{det}} y_{bij} \leq r_{ij}\alpha_i^{det}$. Combining these, we have

$$\sum_{b \in I_i^{prob}} y_{bij} \geq r_{ij}(\alpha_i^j - \alpha_i^{det}). \qquad \text{(A.16)}$$

From constraint (6.4), we have

$$\sum_{b \in I_i^{prob}} x_b \geq \sum_{b \in I_i^{prob}} \sum_{j=1}^{\ell_i} y_{bij} \geq \sum_{j=p}^{\ell_i} \sum_{b \in I_i^{prob}} y_{bij}.$$

Then, from Eqn. (A.16),

$$\sum_{b \in I_i^{prob}} x_b \geq \sum_{j=p}^{\ell_i} r_{ij}(\alpha_i^j - \alpha_i^{det}) \geq (\alpha_{ip} - \alpha_i^{det}) \sum_{j=p}^{\ell_i} r_{ij}.$$

Finally, using constraint (6.1), we conclude that

$$\sum_{b \in I_i^{prob}} x_b \geq \frac{\alpha_{ip} - \alpha_i^{det}}{2}. \tag{A.17}$$

Similarly, since $\sum_{j=1}^{p} r_{ij} \geq 1/2$ and the list L_i is sorted in decreasing order of β_i^j, it follows that

$$\sum_{b \in O_i^{prob}} x_b \geq \frac{\beta_{ip} - \beta_i^{det}}{2}. \tag{A.18}$$

Next we show that $|I_i^{round}| \geq \alpha_{ip} - \alpha_i^{det}$ with probability $\geq 1 - 1/n^2$. Each $b \in B^{prob}$ is independently included in B^{round} with probability $16 x_b \log n$. Hence, by Eqn. (A.17),

$$E[|I_i^{round}|] = \sum_{b \in I_i^{prob}} 16 x_b \log n \geq 8(\alpha_{ip} - \alpha_i^{det}) \log n.$$

Using Chernoff bound[44], $|D_{round} \cap I_i| \leq \alpha_{ip} - \alpha_i^{det}$ with probability at most $1/n^2$. Similarly, using Eqn. (A.18), $|O_i^{round}| \leq \beta_{ip} - \beta_i^{det}$ with probability at most $1/n^2$. The lemma follows by using union bound over the failure probabilities. $\square$

We get the following corollary from Lemma 6.19, which proves the approximation result in Theorem 6.18.

Corollary A.24. *Algorithm 9 gives a feasible safe subset V with expected cost $O(\log n)$ times the optimal.*

Proof. Using union bound over the set of n modules in the above lemma, we conclude that with probability at least $1 - 2/n$, the solution produced by the rounding algorithm after step 2 is feasible. By linearity of expectation, the cost of the rounded solution at this stage is at most $16 \log n$ times that of the LP solution, and therefore $O(\log n)$ times that of the optimal cost. If all modules are not satisfied after step 3, the cost of the data added

[44]If X is sum of independent Boolean random variables with $E[X] = \mu$, then $\Pr[X \leq \mu(1-\epsilon)] \leq e^{\frac{-\mu\epsilon^2}{2}}$ (see, for instance, [132]).

to B in step 3 (by greedily picking the best option B_i^{min} for individual module) is at most $O(n)$ times the optimal. However, this happens with probability at most $2/n$; thus the expected total cost of the final solution V produced by this algorithm remains $O(\log n)$ times the optimal cost. Further, the solution V returned by the algorithm is always a safe subset. $\square$

A.4.7.2 $\Omega(\log n)$-Hardness

The following theorem shows that Algorithm 9 produces an optimal answer upto a constant factor and proves the hardness result in Theorem 6.18.

We give a reduction from the minimum set cover problem to this version of the `Secure-View` problem where $\ell_{\max} = 1$ and each data has unit cost. Since set cover is hard to approximate within a factor of $o(\log n)$ unless NP $\subseteq$ DTIME$(n^{O(\log\log n)})$ [76, 119], the hardness result of the `Secure-View` problem with cardinality constraints follows under the same assumption.

An instance of the set cover problem consists of an input universe $U = \{u_1, u_2, \ldots, u_n\}$, and a set of its subsets $\mathbf{S} = \{S_1, S_2, \ldots, S_M\}$, i.e. each $S_i \subseteq U$. The goal is to find a set of subsets $\mathbf{T} \subseteq \mathbf{S}$ of minimum size (i.e. $|\mathbf{T}|$ is minimized) subject to the constraint that $\cup_{S_i \in \mathbf{T}} S_i = U$.

We create an instance of the `Secure-View` problem with workflow W, where W has a module m_i corresponding to each element $u_i \in U$, and an extra module z (in addition to the dummy source and sink modules s and t) We now express the connections between modules in the workflow W. There is a single incoming edge e_z from source module s to z (for initial input data), a set of edges $\{e_{ij} : S_i \ni u_j\}$ from z to each f_j (for intermediate data), and a single outgoing edge e_j from each f_j (for final output data). to the sink node t. The edge e_z uniquely carries data b_s, and each edge e_j uniquely carries data b_j for $j \in [1, n]$. All edges $\{e_{ij} : j \in S_i\}$ carry the same data a_i, $i \in [1, M]$.

The privacy requirement for z is any single data a_i carried by one of its outgoing edges, while that for each f_j is any single data a_i carried by one of its incoming edges (i.e. $S_i \ni u_j$). In other words, $L_z = \{(0,1)\}$, $L_j = \{(1,0)\}$. Hence only the intermediate data, $\{a_i : i \in [1, M]\}$, can be hidden; the cost of hiding each such data is 1. Note that the maximum list size $\ell_{\max}$ is 1 and the individual cardinality requirements are bounded by 1.

If the minimum set cover problem has a cover of size k, hiding the data corresponding to the subsets selected in the cover produces a solution of cost k for this instance of the `Secure-View` problem. Conversely, if a solution to the `Secure-View` problem hides a set of k data in $\{a_i : i \in [1,M]\}$, selecting the corresponding sets produces a cover of k sets. Hence the `Secure-View` problem with cardinality constraint is $\Omega(\log n)$-hard to approximate.

A.4.8 Proof of Theorem 6.20

We now consider the `Secure-View` problem with set constraints. Here the input requirement lists L_i-s are given as a list of pair of subsets of input and output attributes: $L_i = \{(\overline{I}_i^j, \overline{O}_i^j) : j \in [1,\ell_i], \overline{I}_i^j \subseteq I_i, \overline{O}_i^j \subseteq O_i\}$, for every $i \in [1,n]$ (see Section 6.4.2). The goal is find a safe subset V with minimum cost of hidden attributes $\mathrm{c}(\overline{V})$ such that for every $i \in [1,n]$, $\overline{V} \supseteq (\overline{I}_i^j \cup \overline{O}_i^j)$ for some $j \in [1,\ell_i]$.

Recall that $\ell_{\max}$ denotes the maximum size of the requirement list of a module. Now we prove Theorem 6.20. First we show that the `Secure-View` problem with set constraints is $\ell_{\max}^{\epsilon}$-hard to approximate, and then we give an $\ell_{\max}$-approximation algorithm for this problem.

A.4.8.1 $\ell_{\max}$-Approximation Algorithm

Here we give an $\ell_{\max}$-approximation algorithm for the `Secure-View`problem with set constraints as claimed in Theorem 6.20. The algorithm rounds the solution given by LP relaxation of the following integer program:

$$\text{Minimize } \textstyle\sum_{b \in A} c_b x_b \quad \text{subject to}$$

$$\sum_{j=1}^{\ell_i} r_{ij} \geq 1 \qquad \forall i \in [1,n] \tag{A.19}$$

$$x_b \geq r_{ij} \qquad \forall b \in \overline{I}_i^j \cup \overline{O}_i^j,\ \forall i \in [1,n] \tag{A.20}$$

$$x_b, r_{i,j} \in \{0,1\} \tag{A.21}$$

The LP relaxation is obtained by changing Eqn. (A.21) to

$$x_b, r_{ij} \in [0,1]. \tag{A.22}$$

The rounding algorithm includes all attributes $b \in A$ such that $x_b \geq 1/\ell_{\max}$ to the hidden attribute set $\overline{V}$. The corresponding visible attribute subset V is output as a solution.

Next we discuss the correctness and approximation ratio of the rounding algorithm. Since the maximum size of a requirement list is $\ell_{\max}$, for each i, there exists a $j = j(i)$ such that in the solution of the LP, $r_{ij} \geq 1/\ell_i \geq 1/\ell_{\max}$. Hence there exists at least one $j \in [1, \ell_i]$ such that $\overline{I}_i^j \subseteq \overline{V}, \overline{O}_i^j \subseteq \overline{V}$. Since $\mathtt{c}(\overline{V})$ is most $\ell_{\max}$ times the cost of LP solution, this algorithm gives an $\ell_{\max}$-approximation.

A.4.8.2 $\ell_{\max}^{\epsilon}$-Hardness

The hardness result in Theorem 6.20 is obtained by a reduction from the *minimum label cover* problem [11]. An instance of the minimum label cover problem consists of a bipartite graph $H = (U, U', E_H)$, a label set L, and a non-empty relation $R_{uw} \subseteq L \times L$ for each edge $(u,w) \in E_H$. A feasible solution is a label assignment to the vertices, $A : U \cup U' \to 2^L$, such that for each edge (u,w), there exist $\ell_1 \in A(u), \ell_2 \in A(w)$ such that $(\ell_1, \ell_2) \in R_{uw}$. The objective is to find a feasible solution that minimizes $\sum_{u \in U \cup U'} |A(u)|$.

Unless $\text{NP} \subseteq \text{DTIME}(n^{\text{polylog } n})$, the label cover problem is $|L|^{\epsilon}$-hard to approximate for some constant $\epsilon > 0$ [11, 143]. The instance of the `Secure-View` problem in the reduction will have $\ell_{max} = |L|^2$. Theorem 6.20 follows immediately.

Given an instance of the label cover problem as defined above, we create an instance of the `Secure-View` problem by constructing a workflow W (refer to Figure A.2). For each edge $(u,w) \in E_H$, there is a module x_{uw} in W. In addition, W has another module z.

As shown in Figure A.2, the input and output attributes of the modules are as follows: (i) z has a single incoming edge with the initial input data item b_z, (ii) z has $(|U| + |U'|) \times L$ output attributes $b_{u,\ell}$, for every $u \in U \cup U'$ and every $\ell \in L$. Every such attribute $b_{u,\ell}$ is sent to all x_{uw} where $(u,w) \in E_H$ (see Figure A.2). Hence every x_{uw} has $2L$ input attributes: $\{b_{u,\ell} : \ell \in L\} \bigcup \{b_{w,\ell'} : \ell' \in L\}$. (iii) there is a single outgoing edge from each x_{uw} carrying data item b_{uw} (final output data). The cost of hiding any data is 1. The requirement list of z contains singleton subsets of each intermediate data $b_{u,\ell}$, i.e., $L_z = \{(\phi, \{b_{u,\ell}\}) : u \in$

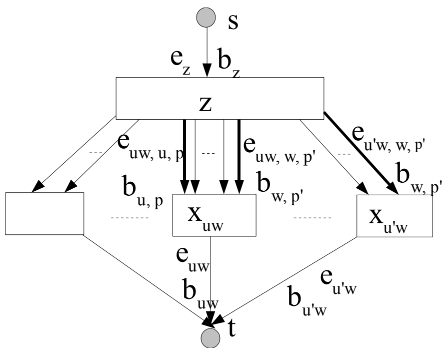


Figure A.2: Reduction from label cover: bold edges correspond to $(p,q) \in R_{uw}$, dotted edges are for data sharing.

$U \cup U', \ell \in L\}$. The list of x_{uw}, for each $(u,w) \in E$, contains pairs of data corresponding to the members of the relation R_{uw}, i.e $L_{uw} = \{(\phi, \{b_{u,\ell_1}, b_{w,\ell_2}\}) : (\ell_1, \ell_2) \in R_{uw}\}$.

The following lemma proves the correctness of the reduction.

Lemma A.25. *The label cover instance H has a solution of cost K iff the* `Secure-View` *instance W has a solution of cost K.*

Proof. Let $A : U \cup U' \to 2^L$ be a solution of the label cover instance H with total cost $K = \sum_{u \in U \cup U'} |A(u)|$. We create a solution $\overline{V}$ for the `Secure-View` instance W as follows: for each $u \in U \cup U'$, and $\ell \in L$, add $b_{u,\ell}$ to hidden attributes $\overline{V}$ iff $\ell \in A(u)$. We claim that $\overline{V}$ is a feasible solution for G. For each $u \in U \cup U'$, $A(u)$ is non-empty; hence, the requirement of z is trivially satisfied. Since A is a valid label cover solution, for each $(u,w) \in E_H$, there exists $\ell_1 \in A(u)$ and $\ell_2 \in A(w)$ such that $(\ell_1, \ell_2) \in R_{uw}$. Hence for the same pair (ℓ_1, ℓ_2), $(b_{u,\ell_1}, b_{w,\ell_2}) \in L_{x_{uw}}$, and both $b_{u,\ell_1}, b_{w,\ell_2} \in \overline{V}$. This satisfies the requirement for all modules x_{uw} in W.

Conversely, let $\overline{V}$ be a solution of the `Secure-View` instance W, where $|\overline{V}| = K$. Note that $\overline{V}$ can include only the intermediate data. For each $u \in U \cup U'$, we define $A(u) = \{\ell | b_{u,\ell} \in \overline{V}\}$. Clearly, $\sum_{u \in U \cup U'} |A(u)| = K$. For each $x_{uw} \in V$, the requirement of x_{uw} is satisfied by $\overline{V}$; hence there exist $\ell_1, \ell_2 \in L$ such that $b_{u,\ell_1}, b_{w,\ell_2} \in \overline{V}$. This implies that for each edge $(u,w) \in E_H$, there exist $\ell_1 \in A(u)$ and $\ell_2 \in A(w)$, where $(\ell_1, \ell_2) \in R_{uw}$, thereby proving feasibility. □

Remark. If $N = |U| + |U'|$, the label cover problem is also known to be $\Omega(2^{\log^{1-\gamma} N})$-hard to approximate for all constant $\gamma > 0$, unless NP $\subseteq$ DTIME$(n^{\text{polylog } n})$ [11, 143]. Thus, the `Secure-View` problem with set constraints is $\Omega(2^{\log^{1-\gamma} n})$-hard to approximate as well, for all constant $\gamma > 0$, under the same complexity assumption.

A.4.9 Proof of Theorem 6.21

The `Secure-View` problem becomes substantially easier to approximate if the workflow has *bounded data sharing*, i.e. when every data d produced by some module is either a final output data or is an input data to at most γ other modules. Though the problem remains NP-hard even with this restriction, Theorem 6.21 shows that it is possible to approximate it within a constant factor when γ is a constant.
First, we give a $(\gamma+1)$-approximation algorithm for the `Secure-View` problem with set constraints, where each data is shared by at most γ edges. This also implies an identical approximation factor for the cardinality version. Then, we show that the cardinality version of the problem is APX-hard, i.e. there exists a constant $c > 1$ such that it is NP-hard to obtain a c-approximate solution to the problem. The set version is therefore APX-hard as well.

A.4.9.1 $(\gamma+1)$-Approximation Algorithm

Recall that the input to the problem includes a requirement list $L_i = \{(\overline{I}_i^j, \overline{O}_i^j) : j \in [1, \ell_i], \overline{I}_i^j \subseteq I_i, \overline{O}_i^j \subseteq O_i\}$ for each module v_i. Let $(\overline{I}_i^{j*}, \overline{O}_i^{j*})$ be a minimum cost pair for module v_i, i.e. $\mathtt{c}(\overline{I}_i^{j*} \cup \overline{O}_i^{j*}) = \min_{j=1}^{\ell_i} \mathtt{c}(\overline{I}_i^j \cup \overline{O}_i^j)$. The algorithm greedily chooses $\overline{I}_i^{j*} \cup \overline{O}_i^{j*}$ for each module v_i, i.e. the set of hidden data $\overline{V} = \bigcup_{1 \le i \le n} (\overline{I}_i^{j*} \cup \overline{O}_i^{j*})$.

Note that each intermediate data is an input to at most γ modules. In any optimal solution, assume that each terminal module of every hidden edge carrying this data pays its cost. Then, the total cost paid by the modules is at most $\gamma + 1$ times the cost of the optimal solution. On the other hand, the total cost paid by any module is at least the cost of the edges incident on the module that are hidden by the algorithm. Thus, the solution of the algorithm has cost at most $\gamma + 1$ times the optimal cost.

A very similar greedy algorithm with the same approximation factor can be given to the Secure-View problem with cardinality constraints.

A.4.9.2 APX-Hardness

We reduce the *minimum vertex cover* problem in cubic graphs to the Secure-View problem with cardinality constraints. An instance of the vertex cover problem consists of an undirected graph $G'(V',E')$. The goal is to find a subset of vertices $S \subseteq V'$ of minimum size $|S|$ such that each edge $e \in E'$ has at least one endpoint in S.

Given an instance of the vertex cover problem, we create a Secure-View instance W (see Figure A.3). For each edge $(u,v) \in E'$, there is a module x_{uv} in W; also there is a module y_v for each vertex $v \in V'$. In addition to these, W contains a single module z.

Next we define the edges in the workflow W; since there is no data sharing, each edge corresponds to a unique data item and cost of hiding each edge is 1. For each x_{uv}, there is a single incoming edge (carrying initial input data) and two outgoing edges (x_{uv}, y_u) and (x_{uv}, y_v). There is an outgoing edge (y_v, z) from every y_v (carrying final output data). Finally, there is an outgoing edge from z for final output data item.

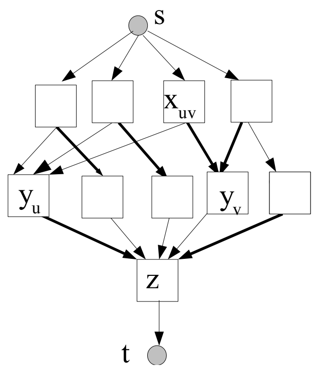


Figure A.3: Reduction from vertex cover, the dark edges show a solution with cost $|E'| + K$, $K =$ size of a vertex cover in G'

Now we define the requirement list for each module in W. For each x_{uv}, $L_{uv} = \{(0,1)\}$, i.e. the requirement for x_{uv} is any single outgoing edge. For each y_v, $L_v = \{(d_v,0),(0,1)\}$, where d_v is the degree of the v in G'. Hence the requirement of the vertex y_v is either all of its incoming edges, or a single outgoing edge. For vertex z, $L_z = \{(1,0)\}$, i.e. hiding any incoming edge suffices.

Lemma A.26. *The vertex cover instance G' has a solution of size K if and only if the* `Secure-View` *instance W has a solution of cost $m' + K$, where $m' = |E'|$ is the number of edges in G'.*

Proof. Let $S \subseteq V'$ be a vertex cover of G' of size K. We create a create a set of hidden edges $\overline{V}$ for the `Secure-View`problem as follows: for each $v \in S$, add (y_v, z) to $\overline{V}$. Further, for each x_{uv}, if $u \notin S$, add (x_{uv}, y_u) to $\overline{V}$, otherwise add (x_{uv}, y_v). For this choice of $\overline{V}$, we claim that V is safe set of attributes for W.

Clearly, the requirement is satisfied for each x_{uv}, since one outgoing edge is hidden; the same holds for all y_v such that $v \in S$. Assuming E' to be non-empty, any vertex cover is of size at least one. Hence at least one incoming edge to z is hidden. Finally, for every y_v such that $v \notin S$, all its incoming edges are hidden; if not, S is not a vertex cover. Hence $\overline{V}$ satisfies the requirement of all modules in W. Since we hide exactly one outgoing edge from all x_{uv}, and exactly one outgoing edge from all y_v where $v \in S$, the cost of the solution is $m' + K$.

Now assume that we have a solution $\overline{V} \subseteq A$ of the `Secure-View` instance with cost $|\overline{V}| = K'$. We can assume, wlog, that for each x_{uv} exactly one outgoing edge is included in $\overline{V}$; if both (x_{uv}, y_u) and (x_{uv}, y_v) are in $\overline{V}$, we arbitrarily select one of u or v, say u, and replace the edge (x_{uv}, y_u) in $\overline{V}$ with the edge (y_u, z) to get another feasible solution without increasing the cost. We claim that the set $S \subseteq V'$ of vertices v such that $(y_v, z) \in \overline{V}$ forms a vertex cover. For any edge $(u, v) \in E'$, if $(x_{uv}, y_u) \notin \overline{V}$, then $(y_u, z) \in \overline{V}$ to satisfy the requirement of y_u, and therefore $u \in S$; otherwise, $v \in S$ by the same argument. Hence S is a vertex cover. Since each vertex x_{uv} has exactly one outgoing edge in $\overline{V}$, $|S| = K' - m'$. $\square$

To complete the proof, note that if G' were a cubic graph, i.e. the degree of each vertex is at most 3, then the size of any vertex cover $K \geq m'/3$. It is known that vertex cover in cubic graphs is APX-hard [9]; hence so is the `Secure-View` problem with cardinality constraints and no data sharing. An exactly identical reduction shows that the `Secure-View` problem with set constraints and no data sharing is APX-hard as well.

A.4.10 Proof of Theorem 6.24

As sketched in Section 6.5.1, the proof of Theorem 6.24 directly follows from the following lemma. This lemma uses the same notations as in Lemma 6.14. Here again m_i is a fixed private module.

Lemma A.27. *If $m_j, j \in [K+1, n]$ is a public module such that $m_j \neq g_j$ in the proof of Lemma 6.14, then $(I_j \cup O_j) \cap \overline{V_i} \neq \phi$, and therefore m_j will be hidden.*

Proof. We will show that if $(I_j \cup O_j) \cap \overline{V_i} = \phi$, then $m_j = g_j$.

Recall that we defined two tuples $\mathbf{p}, \mathbf{q}$ over attributes $I_i \cup O_i$ in the proof of Lemma 6.14 and argued that $p[a] = q[a]$ for all the attributes $a \in (I_i \cup O_i) \cap V_i$. Hence if $p[a] \neq q[a]$, then $a \in \overline{V_i}$, i.e. a is hidden. From the definition of $\texttt{FLIP}$ it follows that, when $(I_j \cup O_j) \cap \overline{V_i} = \phi$, for an input $\mathbf{u}$ to m_j $\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{u}) = \mathbf{u}$. Similarly, $\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{v}) = \mathbf{v}$, where $\mathbf{v} = m_j(\mathbf{u})$. Hence for any input $\mathbf{u}$ to m_j, $g_j(\mathbf{u}) = \texttt{FLIP}_{m_j,\mathbf{p},\mathbf{q}}(\mathbf{u}) = \texttt{FLIP}_{\mathbf{p},\mathbf{q}}(m_j(\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{u})))$ $= \texttt{FLIP}_{\mathbf{p},\mathbf{q}}(m_j(\mathbf{u})) = \texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{v}) = \mathbf{v} = m_j(\mathbf{u})$.

Since this is true for all input $\mathbf{u}$ to m_j, $m_j = g_j$ holds. □

A.4.11 Bounded Data Sharing

In Section A.4.9 we showed that the `Secure-View` problem with cardinality or set constraints has a $\gamma + 1$-approximation where every data item in A can be fed as input to at most γ modules. This implies that without any data sharing (when $\gamma = 1$), `Secure-View` with cardinality or set constraints had a 2-approximation. In the following theorem we show that in arbitrary networks, this problem is $\Omega(\log n)$-hard to approximate.

Theorem A.28. ***Bounded Data sharing (general workflows):*** *The `Secure-View` problem with cardinality constraints without data sharing in general workflows is $\Omega(\log n)$-hard to approximate unless NP $\subseteq$ DTIME($n^{O(\log \log n)}$), even if the maximum size of the requirement lists is 1 and the individual requirements are bounded by 1.*

Proof. The reduction will again be from the set cover problem. An instance of the set cover problem consists of an input universe $\mathbf{U} = \{u_1, u_2, \ldots, u_{n'}\}$, and a set of its subsets $\mathbf{S} = \{S_1, S_2, \ldots, S_{m'}\}$, i.e. each $S_i \subseteq U$. The goal is to find a set of subsets $\mathbf{T} \subseteq \mathbf{S}$ of minimum size (i.e. $|\mathbf{T}|$ is minimized) subject to the constraint that $\cup_{S_i \in \mathbf{T}} S_i = \mathbf{U}$.

Given an instance of the set-cover problem, we construct a workflow W as follow: (i) we create a public module for every element in U, (ii) we create a private module for every set in $\mathbf{S}$, (iii) we add an edge (S_i, u_j) with data item b_{ij} if and only if $u_j \in S_i$. Every set S_i has an incoming edge with data item a_i (initial input data) and every element u_j has an outgoing edge with data item b_j (final output data). The cost of hiding every edge is 0 and the cost of privatizing every set node S_i is 1. The requirement list of every private module u_j is $L_j = \{(1,0)\}$, i.e., for every such module one of the incoming edges must be chosen.

It is easy to verify that the set cover has a solution of size K if and only if the Secure-View problem has a solution of cost K. Since set-cover is known to be $\Omega(\log n')$-hard to approximate, m' is polynomial in n' in the construction in [119], and the total number of nodes in W, $n = O(n' + m')$, this problem has the same hardness of approximation as set cover, i.e. the problem is $\Omega(\log n)$-hard to approximate. □

A.4.12 Cardinality Constraints

Here we show that the Secure-View problem with cardinality constraints is $\Omega(2^{\log^{1-\gamma} n})$-hard to approximate. This is in contrast with the $O(\log n)$-approximation obtained for this problem in all-private workflows (see Theorem 6.18).

Theorem A.29. ***Cardinality Constraints (general workflows):*** *The* Secure-View *problem with cardinality constraints in general workflows is $\Omega(2^{\log^{1-\gamma} n})$-hard to approximate unless $NP \subseteq DTIME(n^{polylog\, n})$, for all constant $\gamma > 0$ even if the maximum size of the requirement lists is 1 and the individual requirements are bounded by 1.*

The hardness result in Theorem A.29 is obtained by a reduction from the *minimum label cover* problem [11]. An instance of the minimum label cover problem consists of a bipartite graph $H = (U, U', E_H)$, a label set L, and a non-empty relation $R_{uw} \subseteq L \times L$ for each edge $(u,w) \in E_H$. A feasible solution is a label assignment to the vertices, $A : U \cup U' \to 2^L$, such that for each edge (u,w), there exist $\ell_1 \in A(u), \ell_2 \in A(w)$ such that $(\ell_1, \ell_2) \in R_{uw}$. The objective is to find a feasible solution that minimizes $\sum_{u \in U \cup U'} |A(u)|$. If $N = |U| + |U'|$, the label cover problem is known to be $|L|^{\epsilon}$-hard to approximate for some constant $\epsilon > 0$, as well as, $\Omega(2^{\log^{1-\gamma} N})$-hard to approximate for all constant $\gamma > 0$, unless $\text{NP} \subseteq \text{DTIME}(n^{\text{polylog}\, n})$ [11, 143].

Given an instance of the label cover problem as defined above, we create an instance of the Secure-View problem by constructing a workflow W (refer to Figure A.4). We will show that the label cover instance H has a solution of cost K if and only if the Secure-View instance W has a solution with the same cost K. Further, in our reduction, the number of modules n in the workflow W will be $O(N^2)$. Hence the Secure-View problem with cardinality constraints in general workflows will be $\Omega(2^{\log^{1-\gamma} n})$-hard to approximate for all constant $\gamma > 0$ under the same complexity assumption which proves Theorem A.29.

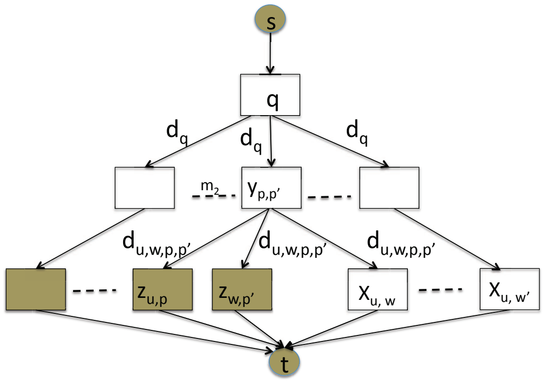


Figure A.4: Reduction from label cover: $(p,q) \in R_{uw}$, the public modules are darken, all public modules have unit cost, all data have zero cost, the names of the data never hidden are omitted for simplicity

Construction First we describe the modules in W. For each edge $(u,w) \in E_H$, there is a private module $x_{u,w}$ in W. For every pair of labels (ℓ_1,ℓ_2), there is a private module y_{ℓ_1,ℓ_2}. There are public module $z_{u,\ell}$ for every $u \in U \cup U', \ell \in L$. In addition, W has another private module v.

As shown in Figure A.4, there are the following types of edges and data items in W: (i) an incoming single edge to v carrying initial input data item d_s, $\mathrm{c}(d_s) = 0$, (ii) an edge from v to every node y_{ℓ_1,ℓ_2}, each such edge carries the same data d_v produced by v, $\mathrm{c}(d_v) = 0$, (iii) for every $(u,w) \in E_H$, for $(\ell_1,\ell_2) \in R_{u,w}$, there is an edge from y_{ℓ_1,ℓ_2} to $x_{u,w}$ carrying data d_{u,w,ℓ_1,ℓ_2}, and $\mathrm{c}(d_{u,w,\ell_1,\ell_2}) = 0$, (iv) further, every such data d_{u,w,ℓ_1,ℓ_2} produced by y_{ℓ_1,ℓ_2} is also fed to both z_{u,ℓ_1} and z_{v,ℓ_2}. (v) All y_{ℓ_1,ℓ_2} and all $z_{u,\ell}$ have an outgoing edge carrying data (final output data items) d_{ℓ_1,ℓ_2} and d_{u,ℓ_1} respectively; $\mathrm{c}(d_{\ell_1,\ell_2}) = 0$ and $\mathrm{c}(d_{u,\ell_1}) = 0$.

Privacy requirement of v is $\{(0,1)\}$, i.e., v has to always choose the output data d_v to satisfy its requirement. Privacy requirement of every y_{ℓ_1,ℓ_2} is $\{(1,0)\}$, i.e. choosing d_v satisfies the requirement for all such y_{ℓ_1,ℓ_2}-s. Requirements of every $x_{u,w}$ is $(1,0)$, so one of the data items d_{u,w,ℓ_1,ℓ_2}, $(\ell_1,\ell_2) \in R_{u,w}$ must be chosen.

All modules except $z_{u,\ell}$-s are private. Cost of privatizing $z_{u,\ell}$, $\mathtt{c}(z_{u,\ell}) = 1$. In the above reduction, all data items have cost 0 and the cost of a solution entirely comes from privatizing the public modules $z_{u,\ell}$-s.

In this reduction, maximum list size and maximum magnitude of any cardinality requirement are both bounded by 1. In the label cover instance it is known that number of labels $L \leq$ number of vertices N, therefore the total number of modules in $W = O(L^2 + LN + N^2) = O(N^2)$. The following lemma proves the correctness of the reduction.

Lemma A.30. *The label cover instance H has a solution of cost K iff the* `Secure-View` *instance W has a solution of cost K.*

Proof. Let $A : U \cup U' \to 2^L$ be a solution of the label cover instance H with total cost $K = \sum_{u \in U \cup U'} |A(u)|$. We create a solution $\overline{V}$ for the `Secure-View` instance W as follows: First, add d_v to the hidden attribute subset $\overline{V}$. This satisfies the requirement of v and all y_{ℓ_1,ℓ_2} without privatizing any public modules. So we need to satisfy the requirements of $x_{u,w}$.

Since A is a valid label cover solution, for every edge $(u,w) \in E$, there is a label pair $(\ell_1,\ell_2) \in R_{uw}$ such that $\ell_1 \in A(u)$ and $\ell_2 \in A(w)$. For every $x_{u,w}$, add such d_{u,w,ℓ_1,ℓ_2} to $\overline{V}$. For each $u \in U \cup U'$, and $\ell \in L$, add $z_{u,\ell}$ to the privatized public modules P. iff $\ell \in A(u)$. It is easy to check that (V,P) is safe for W. If d_{u,w,ℓ_1,ℓ_2} is hidden, both z_{u,ℓ_1} and z_{v,ℓ_2} are added to P. Since $\mathtt{c}(\overline{V}) = 0$, cost of the solution is $\mathtt{c}(\overline{P}) = K = \sum_{u \in U \cup U'} |A(u)|$.

Conversely, let (V,P) be a safe solution of the `Secure-View` instance W, where $K = \mathtt{c}(\overline{P})$. For each $u \in U \cup U'$, we define $A(u) = \{\ell | z_{u,\ell} \in P\}$. Clearly, $\sum_{u \in U \cup U'} |A(u)| = K$. For each $(u,w) \in E$, the requirement of $x_{u,w}$ is satisfied by $\overline{V}$; hence there exist $\ell_1, \ell_2 \in L$ such that $d_{u,w,\ell_1,\ell_2} \in \overline{V}$. Therefore both $z_{u,\ell_1}, z_{w,\ell_2} \in P$. This implies that for each edge $(u,w) \in E_H$, there exist $\ell_1 \in A(u)$ and $\ell_2 \in A(w)$, where $(\ell_1,\ell_2) \in R_{uw}$, thereby proving feasibility. □

A.4.13 Set-Constraints

We modify the LP given in Section A.4.8 and give an $\ell_{\max}$-approximation algorithm for the set-constraints version in general workflows. As before, for an attribute $b \in A$, $x_b = 1$ if and only if b is hidden (in the final solution $b \in \overline{V}$). In addition, for a public module $m_i, i \in [K+1, n]$, $w_i = 1$ if and only if m_i is hidden (in the final solution $m_i \in \overline{P}$). The algorithm rounds the solution given by LP relaxation of the following integer program. The new condition introduced is (A.25), which says that, if any input or output attribute of a public module m_i is included in $\overline{V}$, m_i must be hidden. Further, (A.23) is needed only for the private modules (m_i such that $i \in [1, K]$). For simplicity we denote $\mathrm{c}(b) = c_b$ and $\mathrm{c}(m_i) = \mathrm{c}_i$.

$$\text{Minimize } \textstyle\sum_{b \in A} \mathrm{c}_b x_b + \sum_{i \in [K+1,n]} \mathrm{c}_i w_i \quad \text{subject to}$$

$$\sum_{j=1}^{\ell_i} r_{ij} \geq 1 \qquad \forall i \in [1, K] \tag{A.23}$$

$$x_b \geq r_{ij} \qquad \forall b \in I_{ij} \cup O_{ij},\ \forall i \in [1, K] \tag{A.24}$$

$$w_i \geq x_b \qquad \forall b \in I_i \cup O_i,\ \forall i \in [K+1, n] \tag{A.25}$$

$$x_b, r_{i,j}, w_i \in \{0, 1\} \tag{A.26}$$

The LP relaxation is obtained by changing Constraint (A.26) to

$$x_b, r_{ij}, w_i \in [0, 1]. \tag{A.27}$$

The rounding algorithm outputs $\overline{V} = \{b : x_b \geq 1/\ell_{\max}\}$ and $\overline{P} = \{m_i : b \in \overline{V}$ for some $b \in I_i \cup O_i\}$.

Since the maximum size of a requirement list is $\ell_{\max} = \max_{i=1}^{n} \ell_i$, for each i, there exists a j such that in the solution of the LP, $r_{ij} \geq 1/\ell_i \geq 1/\ell_{\max}$ (from (A.23)). Hence there exists at least one $j \in [1, \ell_i]$ such that $I_{ij} \cup O_{ij} \subseteq \overline{V}$. Further, a public module m_i, $i \in [K+1, n]$ is hidden (i.e. included to $\overline{V}$) only if there is an attribute $b \in I_i \cup O_i$ which is included to $\overline{V}$. Therefore from (A.25), for all $m_i \in \overline{P}$, $w_i \geq \frac{1}{\ell_{\max}}$. Since both $\mathrm{c}(\overline{V})$ and $\mathrm{c}(\overline{P})$ are most $\ell_{\max}$ times the cost of the respective cost in the LP solution, this rounding algorithm gives an $\ell_{\max}$-approximation.

A.5 Proofs from Chapter 7

A.5.1 Limitation of Upward Propagation

We illustrate here why upward propagation requires upstream public modules to be onto (see Section 7.2.1). If particular, we show that if hiding a subset of input attributes for a standalone module m_2 gives Γ-standalone privacy, then hiding the same subset of input attributes in a simple chain workflow ($m_1 \longrightarrow m_2$) may not give Γ-workflow-privacy for m_2 unless m_1 is an onto function.

Let $m_1 : I_1 \to O_1$ be a public module and $m_2 : I_2 \to O_2$ be a private module that gets all inputs from m_1, i.e. $I_2 = O_1$. Let the ranges $R_1 \subseteq O_1$ and $R_2 \subseteq O_2$ of m_1, m_2 respectively be such that $|R_1| = K_1$ and $|R_2| = K_2$. Then hiding all inputs of m_2 gives K_2-standalone-privacy. But in the workflow, m_2 can have at most K_1-workflow privacy when all output attributes of m_2 are visible. This is because any input $\mathbf{x} \in I_2$ has to map to $S = \{\mathbf{y} : \exists \mathbf{x} \in R_1, \mathbf{y} = m_2(\mathbf{x})\}$. The size of S is at most K_1. Therefore the possible output set $\texttt{OUT}_{\mathbf{x},W}$ of the inputs $\mathbf{x} \in R_1$ can have size at most K_1. In the worst case, $K_1 = 1$ when m_1 is a constant function mapping all inputs to a fixed tuple $\mathbf{a}$, therefore $|S| = 1$, and the value of $m_2(\mathbf{a})$ can be exactly guessed from the visible attributes of m_2. Hence the maximum achievable workflow privacy of m_2 totally depends on the behavior of the public module m_1. In the best case, when m_1 is an onto function, the maximum achievable workflow privacy equals the maximum achievable standalone privacy of m_2 by hiding its input attributes. (the maximum achievable standalone privacy of m_2 by hiding its output attributes may be even higher).

A.5.2 Proof of Lemma 7.14

LEMMA 7.14. *Let m_i be a standalone private module with relation R_i, let $\mathbf{x}$ be an input to m_i, and let V_i be a subset of visible attributes such that $\overline{V_i} \subseteq O_i$ (only output attributes are hidden). If $\mathbf{y} \in \texttt{OUT}_{\mathbf{x},m_i}$ then $\mathbf{y} \equiv_{V_i} \mathbf{z}$ where $\mathbf{z} = m_i(\mathbf{x})$.*

Proof. If $\mathbf{y} \in \texttt{OUT}_{x,m_i}$ w.r.t. visible attributes V_i, then from Definition 6.3,

$$\exists R' \in \texttt{Worlds}(R, V_i),\ \exists \mathbf{t}' \in R' \ \ s.t \ \ \mathbf{x} = \pi_{I_i}(\mathbf{t}') \wedge \mathbf{y} = \pi_{O_i}(\mathbf{t}') \tag{A.28}$$

Further, from Definition 6.1, $R' \in \texttt{Worlds}(R, V_i)$ only if $\pi_{V_i}(R_i) = \pi_{V_i}(R')$. Hence there must exist a tuple $\mathbf{t} \in R_i$ such that

$$\pi_{V_i}(\mathbf{t}) = \pi_{V_i}(\mathbf{t}') \tag{A.29}$$

Since $\overline{V_i} \subseteq O_i$, $I_i \subseteq V_i$. From (A.29), $\pi_{I_i}(\mathbf{t}) = \pi_{I_i}(\mathbf{t}') = \mathbf{x}$. Let $\mathbf{z} = \pi_{O_i}(\mathbf{t})$, i.e. $\mathbf{z} = m_i(\mathbf{x})$. From (A.29), $\pi_{V_i \cap O_i}(\mathbf{t}) = \pi_{V_i \cap O_i}(\mathbf{t}')$, then $\pi_{V_i \cap O_i}(\mathbf{z}) = \pi_{V_i \cap O_i}(\mathbf{y})$. Tuples $\mathbf{y}$ and $\mathbf{z}$ are defined on O_i. Hence from Definition 7.5, $\mathbf{y} \equiv_{V_i} \mathbf{z}$. ☐

Corollary A.31. *For a module m_i, and visible attributes V_i such that $\overline{V_i} \subseteq O_i$, if two tuples $\mathbf{y}, \mathbf{z}$ defined on O_i are such that $\mathbf{y} \equiv_{V_i} \mathbf{z}$, then also $\mathbf{y} \equiv_V \mathbf{z}$ where V is a subset of visible workflows in the workflow such that $V = A \setminus H_i$, and $\overline{V_i} \subseteq H_i$.*

Proof. Since $\overline{V_i} = A_i \setminus V_i = (I_i \cup O_i) \setminus V_i \subseteq O_i$, and $\overline{V_i} \subseteq H_i$,

$$\begin{aligned} V_i \cap O_i &= O_i \setminus \overline{V_i} \\ &\supseteq O_i \setminus H_i \\ &= O_i \setminus (A \setminus V) \\ &= O_i \setminus \overline{V} \\ &= O_i \cap V \end{aligned}$$

(A is the set of all attributes).

Since $\pi_{V_i \cap O_i}(\mathbf{y}) = \pi_{V_i \cap O_i}(\mathbf{z})$, and, $V_i \cap O_i \supseteq V \cap O_i$, $\pi_{V \cap O_i}(\mathbf{y}) = \pi_{V \cap O_i}(\mathbf{z})$, i.e., $\mathbf{y} \equiv_V \mathbf{z}$. ☐

Note. Lemma 7.14 does not use any property of single-predecessor workflows and also works for general workflows. This lemma will be used again for the privacy theorem of general workflows (Theorem A.44).

A.5.3 Proof of Lemma 7.12

Definition A.32. *Given subsets of attributes $P, Q \subseteq A$, two tuples $\mathbf{p}, \mathbf{q}$ defined on P, and a tuple $\mathbf{u}$ defined on Q, $\textsc{Flip}_{\mathbf{p},\mathbf{q}}(\mathbf{u}) = \mathbf{w}$ is a tuple defined on Q constructed as follows: (i) if $\pi_{Q \cap P}(\mathbf{u}) = \pi_{Q \cap P}(\mathbf{p})$, then $\mathbf{w}$ is such that $\pi_{Q \cap P}(\mathbf{w}) = \pi_{Q \cap P}(\mathbf{q})$ and $\pi_{Q \setminus P}(\mathbf{w}) = \pi_{Q \setminus P}(\mathbf{w})$,* *(ii) else if $\pi_{Q\cap P}(\mathbf{u}) = \pi_{Q\cap P}(\mathbf{q})$, then $\mathbf{w}$ is such that $\pi_{Q\cap P}(\mathbf{w}) = \pi_{Q\cap P}(\mathbf{p})$ and $\pi_{Q\setminus P}(\mathbf{w}) = \pi_{Q\setminus P}(\mathbf{w})$, (iii) otherwise, $\mathbf{w} = \mathbf{u}$.*

The following observations capture the properties of $\texttt{FLIP}$ function.

Observation A.33.

1. If $\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{u}) = \mathbf{w}$, then $\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{w}) = \mathbf{u}$.
2. $\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{u})) = \mathbf{u}$.
3. If $P \cap Q = \emptyset$, $\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{u}) = \mathbf{u}$.
4. $\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{p}) = \mathbf{q}, \texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{q}) = \mathbf{p}$.
5. If $\pi_{Q\cap P}(\mathbf{p}) = \pi_{Q\cap P}(\mathbf{q})$, then $\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{u}) = \mathbf{u}$.
6. If $Q = Q_1 \cup Q_2$, where $Q_1 \cap Q_2 = \emptyset$, and if $\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\pi_{Q_1}(\mathbf{u})) = \mathbf{w_1}$ and $\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\pi_{Q_2}(\mathbf{u})) = \mathbf{w_2}$, then $\texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{u}) = \mathbf{w}$ such that $\pi_{Q_1}(\mathbf{w}) = \mathbf{w_1}$ and $\pi_{Q_2}(\mathbf{w}) = \mathbf{2}$.

The above definition of flipping will be useful when we consider the scenario where M does not have any successor. When M has successors, we need an extended definition of tuple flipping, denoted by $\texttt{EFLIP}$, as defined below.

Definition A.34. *Given subsets of attributes $P, Q, R \subseteq A$, where two tuples $\mathbf{p}, \mathbf{q}$ defined on $P \cup R$, a tuple $\mathbf{u}$ defined on Q and a tuple $\mathbf{v}$ defined on R, $\texttt{EFLIP}_{\mathbf{p},\mathbf{q};\mathbf{v}}(\mathbf{u}) = \mathbf{w}$ is a tuple defined on Q constructed as follows: (i) if $\mathbf{v} = \pi_R(\mathbf{p})$, then $\mathbf{w}$ is such that $\pi_{Q\cap P}(\mathbf{w}) = \pi_{Q\cap P}(\mathbf{q})$ and $\pi_{Q\setminus P}(\mathbf{w}) = \pi_{Q\setminus P}(\mathbf{w})$, (ii) else if $\mathbf{v} = \pi_R(\mathbf{q})$, then $\mathbf{w}$ is such that $\pi_{Q\cap P}(\mathbf{w}) = \pi_{Q\cap P}(\mathbf{p})$ and $\pi_{Q\setminus P}(\mathbf{w}) = \pi_{Q\setminus P}(\mathbf{w})$, (iii) otherwise, $\mathbf{w} = \mathbf{u}$.*

Note that $\texttt{EFLIP}_{\mathbf{p},\mathbf{q};\pi_{P\cap Q}(\mathbf{u})}(\mathbf{u}) = \texttt{FLIP}_{\mathbf{p},\mathbf{q}}(\mathbf{u})$, where $R = P \cap Q$.

Observation A.35.

1. If $\texttt{EFLIP}_{\mathbf{p},\mathbf{q};\mathbf{v}}(\mathbf{u}) = \mathbf{w}$, and $\mathbf{u}'$ is a tuple defined on $Q' \subseteq Q$, then $\texttt{EFLIP}_{\mathbf{p},\mathbf{q};\mathbf{v}}(\mathbf{u}') = \pi_{Q'}(\mathbf{w})$.

Next we prove the lemma assuming the existence of a composite public module $M = C(\overline{V_i})$.

Lemma 7.12. *Consider a single-predecessor workflow W; any private module m_i in W and any input $\mathbf{x} \in \pi_{I_i}(R)$; and any $\mathbf{y} \in \texttt{OUT}_{\mathbf{x},m_i}$ w.r.t. a set of visible attributes V_i. Given a set of hidden attributes $H_i \subseteq A$, such that (i) the hidden attributes $\overline{V_i} \subseteq H_i$, (ii) only output attributes* *from O_i are included in $\overline{V_i}$ (i.e. $\overline{V_i} \subseteq O_i$), and (iii) every module m_j in the public-closure $C(\overline{V_i})$ is UD-safe w.r.t. $A_j \setminus H_i$. Then $\mathbf{y} \in \texttt{OUT}_{\mathbf{x},W}$ w.r.t. visible attributes $V = A \setminus H_i$.*

Proof. We fix a module m_i, an input $\mathbf{x}$ to m_i and a candidate output $\mathbf{y} \in \texttt{OUT}_{\mathbf{x},m_i}$ for $\mathbf{x}$ w.r.t. visible attributes V_i. Let us refer to the set of public modules in $C(\overline{V_i})$ by simply C, and let us consider the composite module M with the modules in C. By the properties of single-predecessor workflows, M gets all its inputs from m_i and sends its outputs to zero or more than one private modules. By Lemma 7.16, it suffices to prove the lemma assuming M is UD-safe w.r.t. H_i, where we denote the inputs and outputs of M by I and O respectively. Clearly $I \subseteq O_i$ is the subset of output attributes of m_i that is input to M, and the hidden attributes $\overline{V_i} \subseteq I$.

However m_i can also send (i) its visible outputs to other public modules (these public modules will have m_i as its only predecessor, but these public modules will not have any public path in undirected sense to M), and it can send (ii) visible and hidden attributes to other private modules.

We will show that $\mathbf{y} \in \texttt{OUT}_{\mathbf{x},W}$ w.r.t. visible attributes $V = A \setminus \overline{H_i}$ by showing the existence of a possible world $R' \in \texttt{Worlds}(R, V)$, s.t. if $\pi_{I_i}(\mathbf{t}) = \mathbf{x}$ for some $\mathbf{t} \in R'$, then $\pi_{O_i}(\mathbf{t}) = \mathbf{y}$. Since $\mathbf{y} \in \texttt{OUT}_{x,m_i}$, by Lemma 7.14, $\mathbf{y} \equiv_{V_i} \mathbf{z}$ where $\mathbf{z} = m_i(\mathbf{x})$. We consider two cases separately based on whether M has no successor or at least one private successors.

Case I. First consider the easier case that M does not have any successor, so all outputs of M belong to the set of final outputs. We redefine the module m_i to $\widehat{m}_i$ as follows. For an input $\mathbf{u}$ to m_i, $\widehat{m}_i(\mathbf{u}) = \texttt{FLIP}_{\mathbf{y},\mathbf{z}}(m_i(\mathbf{u}))$. All public modules are unchanged, $\widehat{m}_j = m_j$. All private modules $m_j \neq m_i$ are redefined as follows: On an input $\mathbf{u}$ to m_j, $\widehat{m}_j(\mathbf{u}) = m_j(\texttt{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{u}))$. The required possible world R' is obtained by taking the join of the standalone relations of these $\widehat{m}_j$-s, $j \in [n]$.

First note that by the definition of $\widehat{m}_i$, $\widehat{m}_i(\mathbf{x}) = \mathbf{y}$ (since $\widehat{m}_i(x) = \texttt{FLIP}_{\mathbf{y},\mathbf{z}}(m_i(x)) = \texttt{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{z}) = \mathbf{y}$, from Observation A.33(4)). Hence if $\pi_{I_i}(\mathbf{t}) = \mathbf{x}$ for some $\mathbf{t} \in R'$, then $\pi_{O_i}(\mathbf{t}) = \mathbf{y}$.

Next we argue that $R' \in \texttt{Worlds}(R, V)$. Since R' is the join of the standalone relations for modules $\widehat{m}_j$-s, R' maintains all functional dependencies $I_j \to O_j$. Also none of the public modules are unchanged, hence for any public module m_j and any tuple t in R', $\pi_{O_j}(\mathbf{t}) = m_j(\pi_{I_j}(\mathbf{t}))$. So we only need to show that the projection of R and R' on the visible attributes are the same.

Let us assume, wlog. that the modules are numbered in topologically sorted order. Let I_0 be the initial input attributes to the workflow, and let $\mathbf{p}$ be a tuple defined on I_0. There are two unique tuples $\mathbf{t} \in R$ and $\mathbf{t}' \in R'$ such that $\pi_{I_1}(\mathbf{t}) = \pi_{I_1}(\mathbf{t}') = \mathbf{p}$. Since M does not have any successor, let us assume that $M = m_{n+1}$, also wlog. assume that the public modules in C are not counted in $j = 1$ to $n+1$ by renumbering the modules. Note that any intermediate or final attribute $a \in A \setminus I_0$ belongs to O_j, for a unique $j \in [1, n]$ (since for $j \neq \ell$, $O_j \cap O_\ell = \phi$). So it suffices to show that t, t' projected on O_j are equivalent w.r.t. visible attributes for all module j, $j = 1$ to $n+1$.

Let $\mathbf{c}_{j,m}, \mathbf{c}_{j,\widehat{m}}$ be the values of input attributes I_j and $\mathbf{d}_{j,m}, \mathbf{d}_{j,\widehat{m}}$ be the values of output attributes O_j of module m_j, in $\mathbf{t} \in R$ and $\mathbf{t}' \in R'$ respectively on initial input attributes $\mathbf{p}$ (i.e. $\mathbf{c}_{j,m} = \pi_{I_j}(\mathbf{t})$, $\mathbf{c}_{j,\widehat{m}} = \pi_{I_j}(\mathbf{t}')$, $\mathbf{d}_{j,m} = \pi_{O_j}(\mathbf{t})$ and $\mathbf{d}_{j,\widehat{m}} = \pi_{O_j}(\mathbf{t}')$). We prove by induction on $j = 1$ to n that

$$\forall j, 1 \leq j \leq n, \mathbf{d}_{j,\widehat{m}} = \text{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{d}_{j,m}) \tag{A.30}$$

First we argue that proving (A.30) shows that the join of $\langle \widehat{m}_i \rangle_{1 \leq i \leq n}$ is a possible world of R w.r.t. visible attributes V. (A) When m_j is a private module, note that $\mathbf{d}_{j,m}$ and $\mathbf{d}_{j,\widehat{m}} = \text{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{d}_{j,m})$ may differ only on attributes $O_j \cap O_i$ But $\mathbf{y} \equiv_{V_i} \mathbf{z}$, i.e. these tuples are equivalent on the visible attributes. Hence for all private modules, the t, t' are equivalent w.r.t. O_j. (actually for all $j \neq i$, $O_j \cap O_i = \emptyset$, so the outputs are equal and therefore equivalent). (B) When m_j is a public module, $j \neq n+1$, $O_j \cap O_i = \emptyset$, hence the values of t, t' on O_j are the same and therefore equivalent. (C) Finally, consider $M = m_{n+1}$ that is not covered by (A.30). M gets all its inputs from m_i. From (A.30),

$$\mathbf{d}_{i,\widehat{m}} = \text{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{d}_{i,m})$$

Since $\mathbf{y}, \mathbf{z}, \mathbf{d}_{i,m}, \mathbf{d}_{i,\widehat{m}}$ are all defined on attributes O_i, and input to m_{n+1}, $I_{n+1} \subseteq O_i$,

$$\mathbf{c}_{n+1,\widehat{m}} = \text{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{c}_{n+1,m})$$

Hence $\mathbf{c}_{n+1,\widehat{m}} \equiv_V \mathbf{c}_{n+1,m}$. Since these two inputs of m_{n+1} are equivalent w.r.t. V, by the UD-safe property of $M = m_{n+1}$, its outputs are also equivalent, i.e. $\mathbf{d}_{n+1,\widehat{m}} \equiv_V \mathbf{d}_{n+1,m}$. Hence the projections of t, t' on O_{n+1} are also equivalent. Combining (A), (B), (C), t, t' are equivalent w.r.t. V.

Proof of (A.30). The base case follows for $j = 1$. If $m_1 \neq m_i$ (m_j can be public or private), then $I_1 \cap O_i = \emptyset$, so for all input $\mathbf{u}$,

$$\widehat{m}_j(\mathbf{u}) = m_j(\text{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{u})) = m_j(\mathbf{u})$$

Since the inputs $\mathbf{c}_{1,\widehat{m}} = \mathbf{c}_{1,m}$ (both projections of initial input $\mathbf{p}$ on I_1), the outputs $\mathbf{d}_{1,\widehat{m}} = \mathbf{d}_{1,m}$. This shows (A.30). If $m_1 = m_i$, the inputs are the same, and by definition of $\widehat{m}_1$,

$$\begin{aligned}
\mathbf{d}_{1,\widehat{m}} &= \widehat{m}_1(\mathbf{c}_{1,\widehat{m}}) \\
&= \text{FLIP}_{\mathbf{y},\mathbf{z}}(m_i(\mathbf{c}_{1,\widehat{m}})) \\
&= \text{FLIP}_{\mathbf{y},\mathbf{z}}(m_i(\mathbf{c}_{1,m})) \\
&= \text{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{d}_{1,m})
\end{aligned}$$

This shows (A.30).

Suppose the hypothesis holds until $j-1$, consider m_j. From the induction hypothesis, $\mathbf{c}_{j,\widehat{m}} = \text{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{c}_{j,m})$, hence $\mathbf{c}_{j,m} = \text{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{c}_{j,\widehat{m}})$ (see Observation A.33 (1)).

(i) If $j = i$, again,

$$\begin{aligned}
\mathbf{d}_{i,\widehat{m}} &= \widehat{m}_i(\mathbf{c}_{i,\widehat{m}}) \\
&= \text{FLIP}_{\mathbf{y},\mathbf{z}}(m_i(\mathbf{c}_{i,\widehat{m}})) \\
&= \text{FLIP}_{\mathbf{y},\mathbf{z}}(m_i(\text{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{c}_{i,m}))) \\
&= \text{FLIP}_{\mathbf{y},\mathbf{z}}(m_i((\mathbf{c}_{i,m})) \\
&= \text{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{d}_{i,m})
\end{aligned}$$

$\text{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{c}_{i,m}) = \mathbf{c}_{i,m}$ follows due to the fact that $I_i \cap O_i = \emptyset$, $\mathbf{y}, \mathbf{z}$ are defined on O_i, whereas $\mathbf{c}_{i,m}$ is defined on I_i (see Observation A.33 (3)).

(ii) If $j \neq i$ and m_j is a private module,

$$\begin{aligned}
\mathbf{d}_{j,\widehat{m}} &= \widehat{m}_j(\mathbf{c}_{j,\widehat{m}}) \\
&= m_j(\texttt{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{c}_{j,\widehat{m}})) \\
&= m_j(\mathbf{c}_{j,m}) \\
&= \mathbf{d}_{j,m} \\
&= \texttt{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{d}_{j,m})
\end{aligned}$$

$\texttt{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{d}_{j,m}) = \mathbf{d}_{j,m}$ follows due to the fact that $O_j \cap O_i = \emptyset$, $\mathbf{y}, \mathbf{z}$ are defined on O_i, whereas $\mathbf{d}_{i,m}$ is defined on O_j (again see Observation A.33 (3)).

(iii) If m_j is a public module, $j \leq n$, $\widehat{m}_j = m_j$.

Here

$$\begin{aligned}
\mathbf{d}_{j,\widehat{m}} &= \widehat{m}_j(\mathbf{c}_{j,\widehat{m}}) \\
&= m_j(\mathbf{c}_{j,\widehat{m}}) \\
&= m_j(\texttt{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{c}_{j,m})) \\
&= m_j(\mathbf{c}_{j,m}) \\
&= \mathbf{d}_{j,m} \\
&= \texttt{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{d}_{j,m})
\end{aligned}$$

$\texttt{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{d}_{j,m}) = \mathbf{d}_{j,m}$ again follows due to the fact that $O_j \cap O_i = \emptyset$. $\texttt{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{c}_{j,m})$ $= \mathbf{c}_{j,m}$ follows due to following reason. If $I_j \cap O_i = \emptyset$, i.e. if m_j does not get any input from m_i, again this is true (Observation A.33(3)). If m_j gets an input from m_i, i.e. $I_j \cap O_i \neq \emptyset$, since $m_j \neq m_{n+1}$, $I_j \cap O_i$ does not include any hidden attributes from $\overline{V_i}$. But $\mathbf{y} \equiv_{V_i} \mathbf{z}$, i.e. the visible attribute values of $\mathbf{y}, \mathbf{z}$ are the same. In other words, $\pi_{I_j \cap O_i}(\mathbf{y}) = \pi_{I_j \cap O_i}(\mathbf{z})$, and from Observation A.33 (5), $\texttt{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{c}_{j,m}) = \mathbf{c}_{j,m}$.

This completes the proof of the lemma for Case-I.

Case II.

Now consider the case when M has one or more private successors (note that M cannot have any public successor by definition). Let $M = m_k$, and assume that the modules

$m_1, \cdots, m_n$ are sorted in topological order (again, the nodes inside M are not considered explicitly). Hence $I = I_k, O = O_k$, and $I_k \subseteq O_i$. Let $w_y = M(\pi_{I_k}(\mathbf{y}))$, $w_z = M(\pi_{I_k}(\mathbf{z}))$. Instead of $\mathbf{y}, \mathbf{z}$, the flip function will be w.r.t. $\mathbf{Y}, \mathbf{Z}$, where $\mathbf{Y}$ is the concatenation of $\mathbf{y}$ and w_y ($\pi_{O_i}(\mathbf{Y}) = \mathbf{y}$, $\pi_{O_k}(\mathbf{Y}) = w_y$), and $\mathbf{Z}$ is the concatenation of $\mathbf{z}$ and w_z. Hence $\mathbf{Y}, \mathbf{Z}$ are defined on attributes $O_i \cup O_k$.

We redefine the module m_i to $\widehat{m}_i$ as follows. Note that since input to M, $I_k \subseteq O_i$, O_i is disjoint union of I_k and $O_i \setminus I_k$. For an input $\mathbf{u}$ to m_i, $\widehat{m}_i(\mathbf{u})$, defined on O_i is such that

$$\pi_{O_i \setminus I_k}(\widehat{m}_i(\mathbf{u})) = \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\pi_{O_i \setminus I_k}(m_i(\mathbf{u})))$$

and

$$\pi_{I_k}(\widehat{m}_i(\mathbf{u})) = \texttt{EFLIP}_{\mathbf{Y},\mathbf{Z};M(\pi_{I_k}(m_i(\mathbf{u})))}(\pi_{I_k}(m_i(\mathbf{u})))$$

For the component with $\texttt{EFLIP}$, in terms of the notations in Definition A.34, $R = O_k$, $P = Q = O_i$. $\mathbf{p} = \mathbf{Y}, \mathbf{q} = \mathbf{Z}$, defined on $P \cup R = O_i \cup O_k$. $\mathbf{v} = M(\pi_{I_k}(m_i(\mathbf{u})))$, defined on O_k. $\mathbf{u}$ in Definition A.34 corresponds to $m_i(\mathbf{u})$. All public modules are unchanged, $\widehat{m}_j = m_j$. All private modules $m_j \neq m_i$ are redefined as follows: On an input $\mathbf{u}$ to m_j, $\widehat{m}_j(\mathbf{u}) = m_j(\texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{u}))$. The required possible world R' is obtained by taking the join of the standalone relations of these $\widehat{m}_j$-s, $j \in [n]$.

First note that by the definition of $\widehat{m}_i$, $\widehat{m}_i(\mathbf{x}) = \mathbf{y}$ due to the following reason:

(i) $M(\pi_{I_k}(m_i(x))) = M(\pi_{I_k}(\mathbf{z})) = w_z = \pi_{O_k}(\mathbf{Z})$, so

$$\begin{aligned}\pi_{I_k}(\widehat{m}_i(x)) &= \texttt{EFLIP}_{\mathbf{Y},\mathbf{Z};M(\pi_{I_k}(m_i(x)))}(\pi_{I_k}(m_i(x))) \\ &= \texttt{EFLIP}_{\mathbf{Y},\mathbf{Z};M(\pi_{I_k}(\mathbf{z}))}(\pi_{I_k}(\mathbf{z})) \\ &= \pi_{I_k}(\mathbf{y}))\end{aligned}$$

(ii)

$$\begin{aligned}\pi_{O_i \setminus I_k}(\widehat{m}_i(x)) &== \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\pi_{O_i \setminus I_k}(m_i(x))) \\ &= \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\pi_{O_i \setminus I_k}(\mathbf{z})) \\ &= \pi_{O_i \setminus I_k}(\mathbf{y})\end{aligned}$$

Taking union of (i) and (ii), $\widehat{m}_i(x) = \mathbf{y}$. Hence if $\pi_{I_i}(\mathbf{t}) = \mathbf{x}$ for some $\mathbf{t} \in R'$, then $\pi_{O_i}(\mathbf{t}) = \mathbf{y}$.

Again, next we argue that $R' \in \texttt{Worlds}(R, V)$, and it suffices to show that the projection of R and R' on the visible attributes are the same.

Let I_0 be the initial input attributes to the workflow, and let $\mathbf{p}$ be a tuple defined on I_0. There are two unique tuples $\mathbf{t} \in R$ and $\mathbf{t}' \in R'$ such that $\pi_{I_1}(\mathbf{t}) = \pi_{I_1}(\mathbf{t}') = \mathbf{p}$. Note that any intermediate or final attribute $a \in A \setminus I_0$ belongs to O_j, for a unique $j \in [1, n]$ (since for $j \neq \ell$, $O_j \cap O_\ell = \phi$). So it suffices to show that t, t' projected on O_j are equivalent w.r.t. visible attributes for all module j, $j = 1$ to $n+1$.

Let $\mathbf{c}_{j,m}, \mathbf{c}_{j,\widehat{m}}$ be the values of input attributes I_j and $\mathbf{d}_{j,m}, \mathbf{d}_{j,\widehat{m}}$ be the values of output attributes O_j of module m_j, in $\mathbf{t} \in R$ and $\mathbf{t}' \in R'$ respectively on initial input attributes $\mathbf{p}$ (i.e. $\mathbf{c}_{j,m} = \pi_{I_j}(\mathbf{t})$, $\mathbf{c}_{j,\widehat{m}} = \pi_{I_j}(\mathbf{t}')$, $\mathbf{d}_{j,m} = \pi_{O_j}(\mathbf{t})$ and $\mathbf{d}_{j,\widehat{m}} = \pi_{O_j}(\mathbf{t}')$). We prove by induction on $j = 1$ to n that

$$\forall j \neq i, 1 \leq j \leq n, \mathbf{d}_{j,\widehat{m}} = \text{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{d}_{j,m}) \tag{A.31}$$

$$\pi_{I_k}(\mathbf{d}_{i,\widehat{m}}) = \text{EFLIP}_{\mathbf{Y},\mathbf{Z};M(\pi_{I_k}(\mathbf{d}_{i,m}))}(\pi_{I_k}(\mathbf{d}_{i,m})) \tag{A.32}$$

$$\pi_{O_i \setminus I_k}(\mathbf{d}_{i,\widehat{m}}) = \text{FLIP}_{\mathbf{Y},\mathbf{Z}}(\pi_{O_i \setminus I_k}(\mathbf{d}_{i,m})) \tag{A.33}$$

First we argue that proving (A.31), (A.32) and (A.33) shows that the join of $\langle \widehat{m}_i \rangle_{1 \leq i \leq n}$ is a possible world of R w.r.t. visible attributes V.

(A) When m_j is a private module, $j \neq i$, note that $\mathbf{d}_{j,m}$ and $\mathbf{d}_{j,\widehat{m}} = \text{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{d}_{j,m})$ may differ only on attributes $(O_k \cup O_i) \cap O_j$. But for $j \neq i$ and $j \neq k$ (m_j is private module whereas m_k is the composite public module), $(O_k \cup O_i) \cap O_j = \emptyset$. Hence for all private modules other than m_i, the t, t' are equal w.r.t. O_j and therefore equivalent.

(B) For m_i, from (A.32), $\pi_{I_k}(\mathbf{d}_{i,\widehat{m}}) = \text{EFLIP}_{\mathbf{Y},\mathbf{Z};M(\pi_{I_k}(\mathbf{d}_{i,m}))}(\pi_{I_k}(\mathbf{d}_{i,m}))$. Here $\pi_{I_k}(\mathbf{d}_{i,m})$ and $\pi_{I_k}(\mathbf{d}_{i,\widehat{m}})$ may differ on I_k only if $M(\pi_{I_k}(\mathbf{d}_{i,m})) \in \{w_y, w_z\}$. By Corollary cor:out-equiv, $\mathbf{y} \equiv_V \mathbf{z}$, i.e. $\pi_{I_k}(\mathbf{y}) \equiv_V \pi_{I_k}(\mathbf{z})$. But since M is UD-safe, by the downstream-safety property, $w_y \equiv_V w_z$. Then by the upstream-safety property, all inputs $\pi_{I_k}(\mathbf{d}_{i,m})$ $\equiv_V$ $\mathbf{y} \equiv_V \mathbf{z}$ such that $M(\pi_{I_k}(\mathbf{d}_{i,m})) \in \{w_y, w_z\}$. In particular, if $M(\pi_{I_k}(\mathbf{d}_{i,m})) = w_y$, then $\pi_{I_k}(\mathbf{d}_{i,\widehat{m}}) = \pi_{I_k}(\mathbf{z})$, and $\pi_{I_k}(\mathbf{z}), \pi_{I_k}(\mathbf{d}_{i,m})$ will be equivalent w.r.t. V. Similarly, if $M(\pi_{I_k}(\mathbf{d}_{i,m})) = w_z$, then $\pi_{I_k}(\mathbf{d}_{i,\widehat{m}}) = \pi_{I_k}(\mathbf{y})$, and $\pi_{I_k}(\mathbf{y}), \pi_{I_k}(\mathbf{d}_{i,m})$ will be equivalent w.r.t. V. So t, t' are equivalent w.r.t. $V \cap I_k$.

Next we argue that t, t' are equivalent w.r.t. $V \cap (O_i \setminus I_k)$. From (A.32),

$$\pi_{O_i \setminus I_k}(\mathbf{d}_{i,\widehat{m}}) = \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\pi_{O_i \setminus I_k}(\mathbf{d}_{i,m}))$$

$\pi_{O_i \setminus I_k}(\mathbf{d}_{i,\widehat{m}})$ and $\pi_{O_i \setminus I_k}(\mathbf{d}_{i,m})$ differ only if $\pi_{O_i \setminus I_k}(\mathbf{d}_{i,m}) = \pi_{O_i \setminus I_k}(\mathbf{y})$, then $\pi_{O_i \setminus I_k}(\mathbf{d}_{i,\widehat{m}}) = \pi_{O_i \setminus I_k}(\mathbf{z})$, or, $\pi_{O_i \setminus I_k}(\mathbf{d}_{i,m}) = \pi_{O_i \setminus I_k}(\mathbf{z})$, then $\pi_{O_i \setminus I_k}(\mathbf{d}_{i,\widehat{m}}) = \pi_{O_i \setminus I_k}(\mathbf{y})$. But $\pi_{O_i \setminus I_k}(\mathbf{y})$, $\pi_{O_i \setminus I_k}(\mathbf{z})$ are equivalent w.r.t. visible attributes V. Hence $\pi_{O_i \setminus I_k}(\mathbf{d}_{i,m})$ and $\pi_{O_i \setminus I_k}(\mathbf{d}_{i,\widehat{m}})$ are equivalent w.r.t. V. Hence t, t' are equivalent on O_i.

(C) When m_j is a public module, $\mathbf{d}_{j,\widehat{m}} = \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{d}_{j,m})$. Here $\mathbf{d}_{j,m}, \mathbf{d}_{j,\widehat{m}}$ can differ only on $(O_k \cup O_i) \cap O_j$. If $j \neq k$, the intersection is empty, and we are done. If $j = k$, $\mathbf{d}_{j,m}, \mathbf{d}_{j,\widehat{m}}$ may differ only if $\mathbf{d}_{j,m} \in \{w_y, w_z\}$. But note that $\mathbf{y} \equiv_{V_i} \mathbf{z}$, so $\pi_{I_k}(\mathbf{y}) \equiv_{V_i} \pi_{I_k}(\mathbf{z})$, and $\pi_{I_k}(\mathbf{y}) \equiv_V \pi_{I_k}(\mathbf{z})$. Since m_k is UD-safe, for these two equivalent inputs the respective outputs w_y, w_z are also equivalent. Hence in all cases the values of t, t' on O_k are equivalent.

Combining (A), (B), (C), the projections of t, t' on O_j are equivalent for all $1 \leq j \leq n$; i.e. t, t' are equivalent w.r.t. V.

Proof of (A.31), (A.32) and (A.33). The base case follows for $j = 1$. If $m_1 \neq m_i$ (m_1 can be public or private, but $k \neq 1$ since m_i is its predecessor), then $I_1 \cap (O_i \cup O_k) = \emptyset$, so for all input $\mathbf{u}$, $\widehat{m}_j(\mathbf{u}) = m_j(\texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{u})) = m_j(\mathbf{u})$ (if m_1 is private) and $\widehat{m}_j(\mathbf{u}) = m_j(\mathbf{u})$ (if m_1 is public). Since the inputs $\mathbf{c}_{1,\widehat{m}} = \mathbf{c}_{1,m}$ (both projections of initial input $\mathbf{p}$ on I_1), the outputs $\mathbf{d}_{1,\widehat{m}} = \mathbf{d}_{1,m}$. This shows (A.31). If $m_1 = m_i$, the inputs are the same, and by definition of $\widehat{m}_1$,

$$\begin{aligned}
\pi_{I_k}(\mathbf{d}_{1,\widehat{m}}) &= \pi_{I_k}(\widehat{m}_1(\mathbf{c}_{1,\widehat{m}})) \\
&= \texttt{EFLIP}_{\mathbf{Y},\mathbf{Z};M(\pi_{I_k}(m_1(\mathbf{c}_{1,\widehat{m}})))}(\pi_{I_k}(m_1(\mathbf{c}_{1,\widehat{m}}))) \\
&= \texttt{EFLIP}_{\mathbf{Y},\mathbf{Z};M(\pi_{I_k}(m_1(\mathbf{c}_{1,m})))}(\pi_{I_k}(m_1(\mathbf{c}_{1,m}))) \\
&= \texttt{EFLIP}_{\mathbf{Y},\mathbf{Z};M(\pi_{I_k}(\mathbf{d}_{1,m}))}(\pi_{I_k}(\mathbf{d}_{1,m}))
\end{aligned}$$

This shows (A.32) for $i = 1$. Again, by definition of $\widehat{m}_1$,

$$\pi_{O_1 \setminus I_k}(\mathbf{d}_{1,\widehat{m}}) = \pi_{O_1 \setminus I_k}(\widehat{m}_1(\mathbf{c}_{1,\widehat{m}}))$$

$$\begin{aligned}
&= \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\pi_{O_1\setminus I_k}(m_1(\mathbf{c}_{1,\widehat{m}}))\\
&= \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\pi_{O_1\setminus I_k}(m_1(\mathbf{c}_{1,m}))\\
&= \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\pi_{O_1\setminus I_k}(\mathbf{d}_{1,m})
\end{aligned}$$

This shows (A.33).

Suppose the hypothesis holds until $j-1$, consider m_j. From the induction hypothesis, if $I_j \cap O_i = \emptyset$ (m_j does not get input from m_i) then $\mathbf{c}_{j,\widehat{m}} = \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{c}_{j,m})$, hence $\mathbf{c}_{j,m} = \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{c}_{j,\widehat{m}})$ (see Observation A.33(1)).

(i) If $j = i$, $I_i \cap O_i = \emptyset$, hence $\mathbf{c}_{i,\widehat{m}} = \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{c}_{i,m}) = \mathbf{c}_{i,m}$ ($I_i \cap (O_i \cup O_k) = \emptyset$, m_k is a successor of m_i, so m_i cannot be successor of m_k). By definition of $\widehat{m}_i$,

$$\begin{aligned}
\pi_{I_k}(\mathbf{d}_{i,\widehat{m}}) &= \pi_{I_k}(\widehat{m}_i(\mathbf{c}_{i,\widehat{m}}))\\
&= \texttt{EFLIP}_{\mathbf{Y},\mathbf{Z};M(\pi_{I_k}(m_i(\mathbf{c}_{i,\widehat{m}})))}(\pi_{I_k}(m_i(\mathbf{c}_{i,\widehat{m}})))\\
&= \texttt{EFLIP}_{\mathbf{Y},\mathbf{Z};M(\pi_{I_k}(m_i(\mathbf{c}_{i,m})))}(\pi_{I_k}(m_i(\mathbf{c}_{i,m})))\\
&= \texttt{EFLIP}_{\mathbf{Y},\mathbf{Z};M(\pi_{I_k}(\mathbf{d}_{i,m}))}(\pi_{I_k}(\mathbf{d}_{i,m}))
\end{aligned}$$

This shows (A.32).

Again,

$$\begin{aligned}
\pi_{O_i\setminus I_k}(\mathbf{d}_{i,\widehat{m}}) &= \pi_{O_i\setminus I_k}(\widehat{m}_i(\mathbf{c}_{i,\widehat{m}}))\\
&= \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\pi_{O_i\setminus I_k}(m_i(\mathbf{c}_{i,\widehat{m}})))\\
&= \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\pi_{O_i\setminus I_k}(m_i(\mathbf{c}_{i,m})))\\
&= \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\pi_{O_i\setminus I_k}(\mathbf{d}_{i,m}))
\end{aligned}$$

This shows (A.33).

(ii) If $j = k$, m_k gets all its inputs from m_i, so $\pi_{I_k}(\mathbf{d}_{i,m}) = \mathbf{c}_{k,m}$. Hence

$$\begin{aligned}
\mathbf{c}_{k,\widehat{m}} &= \texttt{EFLIP}_{\mathbf{Y},\mathbf{Z};M(\pi_{I_k}(\mathbf{d}_{i,m}))}(\mathbf{c}_{k,m})\\
&= \texttt{EFLIP}_{\mathbf{Y},\mathbf{Z};M(\mathbf{c}_{k,m})}(\mathbf{c}_{k,m})\\
&= \texttt{EFLIP}_{\mathbf{Y},\mathbf{Z};\mathbf{d}_{k,m}}(\mathbf{c}_{k,m})
\end{aligned}$$

Therefore,

$$\begin{aligned}\mathbf{d}_{k,\widehat{m}} &= \widehat{m}_k(\mathbf{c}_{k,\widehat{m}}) \\ &= m_k(\mathbf{c}_{k,\widehat{m}}) \\ &= m_k(\texttt{EF}\textsc{lip}_{\mathbf{Y},\mathbf{Z};\mathbf{d}_{k,m}}(\mathbf{c}_{k,m}))\end{aligned}$$

Lets evaluate the term $m_k(\texttt{EF}\textsc{lip}_{\mathbf{Y},\mathbf{Z};\mathbf{d}_{k,m}}(\mathbf{c}_{k,m}))$. This says that for an input to m_k is $\mathbf{c}_{k,m}$, and its output $\mathbf{d}_{k,m}$, (a) if $\mathbf{d}_{k,m} = w_y$, then

$$\texttt{EF}\textsc{lip}_{\mathbf{Y},\mathbf{Z};\mathbf{d}_{k,m}}(\mathbf{c}_{k,m}) = \pi_{I_k}(\mathbf{z}),$$

and in turn

$$\mathbf{d}_{k,\widehat{m}} = m_k(\texttt{EF}\textsc{lip}_{\mathbf{Y},\mathbf{Z};\mathbf{d}_{k,m}}(\mathbf{c}_{k,m})) = w_z;$$

(b) if $\mathbf{d}_{k,m} = w_z$, then

$$\texttt{EF}\textsc{lip}_{\mathbf{Y},\mathbf{Z};\mathbf{d}_{k,m}}(\mathbf{c}_{k,m}) = \pi_{I_k}(\mathbf{y}),$$

and in turn

$$\mathbf{d}_{k,\widehat{m}} = m_k(\texttt{EF}\textsc{lip}_{\mathbf{Y},\mathbf{Z};\mathbf{d}_{k,m}}(\mathbf{c}_{k,m})) = w_y;$$

(c) otherwise

$$\begin{aligned}\mathbf{d}_{k,\widehat{m}} &= m_k(\texttt{EF}\textsc{lip}_{\mathbf{Y},\mathbf{Z};\mathbf{d}_{k,m}}(\mathbf{c}_{k,m})) \\ &= m_k(\mathbf{c}_{k,m}) = \mathbf{d}_{k,m}\end{aligned}$$

According to Definition A.32, the above implies that

$$\begin{aligned}\mathbf{d}_{k,\widehat{m}} &= \texttt{F}\textsc{lip}_{w_y,w_z}(\mathbf{d}_{k,m}) \\ &= \texttt{F}\textsc{lip}_{\mathbf{Y},\mathbf{Z}}(\mathbf{d}_{k,m})\end{aligned}$$

This shows (A.31).

(iii) If $j \neq i$ and m_j is a private module, m_j can get inputs from m_i. (but since there is no data sharing $I_j \cap I_k = \emptyset$), and other private or public modules $m_\ell, \ell \neq i$ (ℓ can be equal to k). Let us partition the input to m_j ($\mathbf{c}_{j,m}$ and $\mathbf{c}_{j,\widehat{m}}$ defined on I_j) on attributes $I_j \cap O_i$ and $I_j \setminus O_i$ From (A.31), using the IH,

$$\pi_{I_j \setminus O_i}(\mathbf{c}_{j,\widehat{m}}) = \text{FLIP}_{\mathbf{Y},\mathbf{Z}}(\pi_{I_j \setminus O_i}(\mathbf{c}_{j,m})) \tag{A.34}$$

Now $I_k \cap I_j = \emptyset$, since there is no data sharing. Hence $(I_j \cap O_i) \subseteq (O_i \setminus I_k)$. From (A.33) using Observation A.35,

$$\pi_{I_j \cap O_i}(\mathbf{c}_{j,\widehat{m}}) = \text{FLIP}_{\mathbf{Y},\mathbf{Z}}(\pi_{I_j \cap O_i}(\mathbf{c}_{j,m})) \tag{A.35}$$

From (A.34) and (A.35), using Observation A.33 (6), and since $\mathbf{c}_{j,m}, \mathbf{c}_{j,\widehat{m}}$ are defined on I_j, so

$$\mathbf{c}_{j,\widehat{m}} = \text{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{c}_{j,m}) \tag{A.36}$$

From (A.36),

$$\begin{aligned}
\mathbf{d}_{j,\widehat{m}} &= \widehat{m}_j(\mathbf{c}_{j,\widehat{m}}) \\
&= m_j(\text{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{c}_{j,\widehat{m}})) \\
&= m_j(\mathbf{c}_{j,m}) \\
&= \mathbf{d}_{j,m} \\
&= \text{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{d}_{j,m})
\end{aligned}$$

$\text{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{d}_{j,m}) = \mathbf{d}_{j,m}$ follows due to the fact that $O_j \cap (O_i \cup O_k) = \emptyset$ ($j \neq \{i,k\}$), $\mathbf{Y}, \mathbf{Z}$ are defined on $O_i \cup O_k$, whereas $\mathbf{d}_{j,m}$ is defined on O_j (again see Observation A.33 (3)).

(iv) Finally consider m_j is a public module such that $j \neq k$. m_j can still get input from m_i, but none of the attributes in $I_j \cap O_i$ can be hidden by the definition of $m_k = M = C(\overline{V_i})$. Further, by the definition of $M = m_k$, m_j cannot get any input from m_k (M

is the closure of public module); so $I_j \cap O_k = \emptyset$. Let us partition the inputs to m_j ($\mathbf{c}_{j,m}$ and $\mathbf{c}_{j,\widehat{m}}$ defined on I_j) into three two disjoint subsets: (a) $I_j \cap O_i$, and (b) $I_j \setminus O_i$. Since there is no data sharing $I_k \cap I_j = \emptyset$, and we again get (A.36) that

$$\begin{aligned} \mathbf{c}_{j,\widehat{m}} &= \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{c}_{j,m}) \\ &= \mathbf{c}_{j,m} \end{aligned}$$

$\texttt{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{c}_{j,m}) = \mathbf{c}_{j,m}$ follows due to following reason. If $I_j \cap O_i = \emptyset$, i.e. if m_j does not get any input from m_i, again this is true (then $I_j \cap (O_i \cup O_k) = (I_j \cap O_i) \cup (I_j \cap O_k) = \emptyset$). If m_j gets an input from m_i, i.e. $I_j \cap O_i \neq \emptyset$, since $j \neq k$, $I_j \cap O_i$ does not include any hidden attributes from $\overline{V_i}$ (m_k is the closure $C(\overline{V_i})$). But $\mathbf{y} \equiv_{V_i} \mathbf{z}$, i.e. the visible attribute values of $\mathbf{y}, \mathbf{z}$ are the same. In other words, $\pi_{I_j \cap O_i}(\mathbf{y}) = \pi_{I_j \cap O_i}(\mathbf{z})$, and again from Observation A.33 (5),

$$\texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{c}_{j,m}) = \texttt{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{c}_{j,m}) = \mathbf{c}_{j,m}$$

(again, $I_j \cap O_k = \emptyset$).

Therefore,

$$\begin{aligned} \mathbf{d}_{j,\widehat{m}} &= \widehat{m}_j(\mathbf{c}_{j,\widehat{m}}) \\ &= m_j(\mathbf{c}_{j,\widehat{m}}) \\ &= m_j(\texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{c}_{j,m})) \\ &= m_j(\mathbf{c}_{j,m}) \\ &= \mathbf{d}_{j,m} \\ &= \texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{d}_{j,m}) \end{aligned}$$

$\texttt{FLIP}_{\mathbf{Y},\mathbf{Z}}(\mathbf{d}_{j,m}) = \mathbf{d}_{j,m}$ again follows due to the fact that $O_j \cap (O_i \cup O_k) = \emptyset$, since $j \notin \{i,k\}$.

Hence all the cases for the IH hold true, and this completes the proof of the lemma for Case-II.

$\square$

A.5.4 Proof of Lemma 7.16

Lemma 7.16. *Let M be a composite module consisting of public modules. Let H be a subset of hidden attributes such that every public module m_j in M is UD-safe w.r.t. $A_j \setminus H$. Then M is UD-safe w.r.t. $(I \cup O) \setminus H$.*

Proof. Let us assume, wlog., that the modules in M are $m_2, \cdots, m_k$ where modules are listed in topological order. For $j = 2$ to k, let M^j be the composite module comprising the modules $m_2, \cdots, m_j$, and let I^j, O^j be its input and output. Hence $M^p = M, I^p = I$ and $O^p = O$. We prove by induction on $2 \leq j \leq p$ that M^j is UD-safe w.r.t. $H \cap (I^j \cup O^j)$.

The base case directly follows for $j = 2$, since $A_2 = I_2 \cup O_2 = I^2 \cup O^2$. Let the hypothesis hold until $j = j$ and consider $j+1$. Hence M^j is UD-safe w.r.t. $(I^j \cup O^j) \setminus H$. The module m_{j+1} may consume some outputs of M^j (m_2 to m_j). Hence

$$I^{j+1} = I^j \cup I_{j+1} \setminus O^j \;\; \textit{and} \;\; O^{j+1} = O^j \cup O_{j+1} \setminus I_{j+1} \tag{A.37}$$

First we show that for two equivalent inputs the outputs are equivalent. Consider two inputs x_1, x_2 to M^{j+1} (on attributes I^{j+1}) and let $y_1 = M^{j+1}(x_1)$ and $y_2 = M^{j+1}(x_2)$ (on attributes O^{j+1}).

Let $x_1 \equiv_V x_2$ w.r.t. $V = (I^{j+1} \cup O^{j+1}) \setminus H$. By (A.37), $x_1 \equiv_{V_1} x_2$ where $V_2 = (I^j \cup O^j) \setminus H$. This is because $O^j \cap I^j = \emptyset$ and therefore $I^j \subseteq I^{j+1}$. Since modules are sorted in topological order, all of the attributes in O_{j+1} belong to O^{j+1}. Also recall that every attribute is produced by a unique module; hence $O^j \cap O_{j+1} = \emptyset$. Hence y_1 can be partitioned into y_{11} and y_{12}, where y_{11} is defined on O_{j+1} and y_{12} is defined on O^j; let y_{21}, y_{22} be the corresponding partitions of y_2.

Similarly, let x_{12} be projections of x_1 on I^j similarly define x_{22} for x_2. Since $x_1 \equiv_{V_2} x_2$, $x_{12} \equiv_{V_2} x_{22}$. Since $y_{12} = M^j(x_{12})$ and $y_{22} = M^j(x_{22})$, by IH

$$y_{12} \equiv_{V_2} y_{22} \tag{A.38}$$

Let x'_{11} (resp. x'_{21}) be projections of x_1 (resp. x_{22}) on $I_{j+1} \setminus O^j$. Hence $x'_{11} \equiv_V x'_{21}$. Let x''_{11} (resp. x''_{21}) be projections of y_{12} (resp. y_{22}) on $I_{j+1} \cap O^j$. Since $y_{12} \equiv_{V_2} y_{22}$, $x''_{11} \equiv_{V_2} x''_{21}$. Let x_{11} be concatenation of x'_{11}, x''_{11}; similarly x_{12}. Note that these are defined on I_{j+1}.

$V = (I^{j+1} \cup O^{j+1}) \setminus H$ and $V_2 = (I^j \cup O^j) \setminus H$. Therefore $x_{11} \equiv_{V_1} x_{12}$, where $V_1 = (I_{j+1} \cup O_{j+1}) \setminus H$. Since m_{j+1} is UD-safe w.r.t. V_1,

$$y_{11} \equiv_{V_1} y_{21} \tag{A.39}$$

(note that $y_{11} = m_{j+1}(x_{11}), y_{21} = m_{j+1}(x_{21})$).

$V_1 \cup V_2 \supseteq V$. Hence from (A.38) and (A.39), concatenating y_{11}, y_{12} to y_1 and y_{21}, y_{22} to y_2,

$$y_1 \equiv_V y_2$$

This shows that for two equivalent input the outputs are equivalent. The other direction, for two equivalent outputs all of their inputs are equivalent can be proved in similar way by considering modules in reverse topological order from m_k to m_2. □

A.5.5 Proof of Proposition 7.11

By Definition 7.8, a workflow W is *not* a single-predecessor workflow, if one of the following holds: (i) there is a public module m_j in W that belongs to the public-closure of a private module m_i but has no directed path from m_i, or, (ii) such a public module m_j has directed path from more than one private modules, or, (iii) W has data sharing.

To prove the proposition we provide three example workflows where exactly one of the violating conditions (i), (ii), (iii) holds, and Theorem 7.10 does not hold in those workflows.

Case (i) was shown in Section 7.3.2. To complete the proof we demonstrate here cases (ii) and (iii). We start by considering data sharing and give an example in which the second condition of Definition 7.8 holds, but in which the propagation only through public modules does not suffice due to data sharing.

Multiple private predecessor. Finally we give an example where a public module belonging to a public-closure has more than one private predecessors.

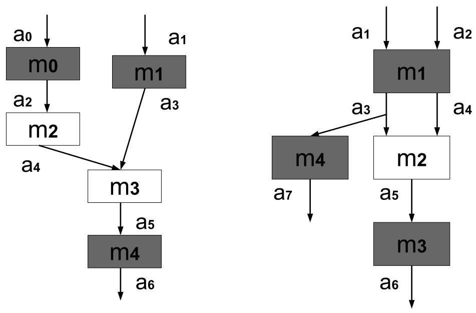
Example A.36. Consider the workflow W_b in Figure A.5a, which is a modification of W_a by the addition of private module m_0, that takes a_0 as input and produces $a_2 = m_0(a_0) = a_0$ as output. The public module m_3 is in public-closure of m_1, but has directed public paths

	a_0	a_1	a_2	a_3	a_4	a_5	a_6
r_1	0	0	0	0	0	0	0
r_2	1	0	1	0	1	1	1
r_3	0	0	1	0	1	1	1
r_4	1	1	1	1	1	1	1

Table A.1: Relation R_b for workflow W_b given in Figure A.5a

from both m_0 and m_1. The relation R_b for W_b in given in Table A.1 where the hidden attributes $\{a_2, a_3, a_4, a_5\}$ are colored in grey.

Now we have exactly the same problem as before: When $\widehat{m}_1$ maps 0 to 1, $a_5 = 1$ irrespective of the value of a_4. In the first row $a_6 = 0$, whereas in the second row $a_6 = 1$. However, whatever the new definitions of $\widehat{m}_0$ are for m_0 and $\widehat{m}_4$ for m_4, $\widehat{m}_4$ cannot map 1 to both 0 and 1. Hence $\Gamma = 1$. □ □



(a) m_3 has paths from m_0, m_1 (b) a_3 is shared as input to m_2, m_3

Figure A.5: White modules are public, Grey are private.

Example A.37. Consider the workflow, say W_d, given in Figure A.5b. All attributes take values in $\{0,1\}$. The initial inputs are a_1, a_2, and final outputs are a_6, a_7; only m_4 is public. The functionality of modules is as follows: (i) m_1 takes a_1, a_2 as input and produces $m_1(a_1, a_2) = (a_3 = a_1, a_4 = a_2)$. (ii) m_2 takes a_3, a_4 as input and produces $a_5 = m_2(a_3, a_4) = a_3 \vee a_4$ (OR). (iii) m_3 takes a_5 as input and produces $a_6 = m_3(a_5) = a_5$. (iv) m_4 takes a_3 as input and produces $a_7 = m_4(a_3) = a_3$. Note that data a_3 is input to both m_2, m_4, hence the workflow has data sharing.

Now focus on private module $m_1 = m_i$. Clearly hiding output a_3 of m_1 gives 2-standalone privacy. and for hidden attribute $h_i = \{a_3\}$, the public-closure $C(h_i) = \{m_2\}$. As given in the theorem, $H_i \subseteq O_i \cup \bigcup_{m_\ell \in M^-} A_\ell = \{a_3, a_4, a_5\}$ in this case.

We claim that hiding even all of $\{a_3, a_4, a_5\}$ gives only trivial 1-workflow-privacy of m_1, although the UD-safe condition is satisfied for m_2 (actually hiding a_3, a_4 gives 4-standalone-privacy for m_1). Table A.2 gives the relation R_d, where the hidden attribute values are in Grey.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7
r_1	0	0	0	0	0	0	0
r_2	0	1	0	1	1	1	0
r_3	1	0	1	0	1	1	1
r_4	1	1	1	1	1	1	1

Table A.2: Relation R_d for workflow W_d given in Figure A.5.

When a_3 (and also a_4) is hidden, a possible candidate output of input tuple $x = (0,0)$ to m_1 is $(\underline{1}, \underline{0})$. So we need to have a possible world where m_1 is redefined as $\widehat{m}_1(0,0) = (1,0)$. Then a_5 takes value 1 in the first row, and this is the only row with visible attributes $a_1 = 0, a_2 = 0$. So this requires that $\widehat{m}_3(a_5 = 1) = (a_6 = 0)$ and $\widehat{m}_4(a_3 = 1) = (a_7 = 0)$, to have the same projection on visible a_6, a_7.

The second, third and fourth rows, r_2, r_3, r_4, have $a_6 = 1$, so to have the same projection, we need $a_5 = 0$ for these three rows, so we need $\widehat{m}_3(a_5 = 0) = (a_6 = 1)$ (since we had to already define $\widehat{m}_3(1) = 0$). When a_5 is 0, since the public module m_2 is an OR function, the only possibility of the values of a_3, a_4 in rows r_2, r_3, r_4 are $(0,0)$. Now we have a conflict on the value of the visible attribute a_7, which is 0 for r_2 but 1 for r_3, r_4, whereas for all these rows the value of a_3 is 0. $\widehat{m}_4$ being a function with dependency $a_3 \rightarrow a_7$, cannot map a_3 to both 0 and 1. Similarly we can check that if $\widehat{m}_1(0,0) = (0,1)$ or $\widehat{m}_1(0,0) = (1,1)$ (both a_3, a_4 are hidden), we will have exactly the same problem. Hence all possible worlds of R_d with these hidden attributes must map $\widehat{m}_1(0,0)$ to $(0,0)$, and therefore $\Gamma = 1$. □ □

A.5.6 Proof of Theorem 7.18

Theorem 7.18. *Given public module m_i with total number of attributes k, and a subset of attributes V, checking whether m_i is UD-safe w.r.t V is coNP-hard in k.*

Proof. We do a reduction from UNSAT, where given n variables $x_1, \cdots, x_n$, and a Boolean formula $f(x_1, \cdots, x_n)$, the goal is to check whether f is *not* satisfiable. In our construction, m_i has $n+1$ inputs $x_1, \cdots, x_n$ and y, and the output is $z = m_i(x_1, \cdots, x_n, y) = f(x_1, \cdots, x_n) \vee y$ (OR). The set of visible attributes is $\{y, z\}$; so all of $x_1, \cdots, x_n$ are hidden. We claim that f is not satisfiable if and only if m_i is UD-safe w.r.t. V.

Suppose f is not satisfiable, so for all assignments of $x_1, \cdots, x_n$, $f(x_1, \cdots, x_n) = 0$. For output $z = 0$, then the visible attribute y must have 0 value in all the rows of the relation of m_i. Also for $z = 1$, the visible attribute y must have 1 value, since in all rows $f(x_1, \cdots, x_n) = 0$. Hence for equivalent inputs w.r.t. V, the outputs are equivalent and vice versa. Therefore m_i is UD-safe w.r.t. V.

Now suppose f is satisfiable, then there is at least one assignment of $x_1, \cdots, x_n$, such that $f(x_1, \cdots, x_n) = 1$. In this row, for $y = 0$, $z = 1$. However for all assignments of $x_1, \cdots, x_n$, whenever $y = 1$, $z = 1$. So for output $z = 1$, all inputs producing z are not equivalent w.r.t. the visible attribute y, therefore m_i is not upstream-safe and hence not UD-safe. □

A.5.7 Communication Complexity Lower Bound for UD-safe Subsets

Given a public module m_j and a subset $V \subseteq A_j$, in this section we give a lower bound on the communication complexity to verify if V is a UD-safe subset for m_j. This also gives a lower bound for the optimization problem of finding the optimal UD-safe subset for m_j: assume that each attribute in V has cost > 0 whereas all other attributes have cost zero; then the optimization problem has a solution of cost 0 if and only if V is a UD-safe subset.

If the standalone relation R_j of m_j has N rows, we show that deciding whether V is UD-safe needs $\Omega(N)$ time. The argument uses the similar ideas as given in [63] for the standalone `Secure-View` problem. Note that simply reading the relation R_j as input takes $\Omega(N)$ time. So the lower bound of $\Omega(N)$ does not make sense unless we assume the presence of a *data supplier* which supplies the tuples of R_j on demand: Given an assignment $\mathbf{x}$ of the input attributes I_j, the data supplier outputs the value $\mathbf{y} = m_j(\mathbf{x})$ of the output attributes O_j. The following theorem shows the $\Omega(N)$ communication complexity lower bound in terms of the number of calls to the data supplier; namely, that (up to a constant factor) one indeed needs to view the full relation.

Lemma A.38. *Given module m_j and a subset of attributes V, deciding whether m_j is UD-safe w.r.t V requires $\Omega(N)$ calls to the data supplier, where N is the number of tuples in the standalone relation R_j of m_j.*

Proof. We prove the theorem by a communication complexity reduction from the set disjointness problem: Suppose Alice and Bob own two subsets A and B of a universe U, $|U| = N$. To decide whether they have a common element (i.e. $A \cap B \neq \phi$) takes $\Omega(N)$ communications [115].

We construct the following relation R_j with $N + 1$ rows for the module m_j. m_j has three input attributes: a, b, id and one output attribute y. The attributes a, b and y are Boolean, whereas id is in the range $[1, N + 1]$. The input attribute id denotes the identity of every row in R and takes value $i \in [1, N + 1]$ for the i-th row. The module m_j computes the AND function of inputs a and b, i.e., $y = a \wedge b$.

Row i, $i \in [1, N]$, corresponds to element $i \in U$. In row i, value of a is 1 iff $i \in A$; similarly, value of b is 1 iff $i \in B$. The additional $N + 1$-th row has $a_{N+1} = 1$ and $b_{N+1} = 0$. The goal is to check if visible attributes $V = \{y\}$ (with hidden attributes $\overline{V} = \{id, a, b\}$) is UD-safe for m_j.

Note that if there is a common element $i \in A \cap B$, then there are two y values in the table: in the i-th row, $1 \leq i \leq n$, the value of $y = a \wedge b$ will be 1, whereas, in the $N + 1$-th row it is 0. Hence $\overline{V} = \{id, a, b\}$ or $V = \{y\}$ is not UD-safe for m_j – all inputs are equivalent, but their outputs are not equivalent, so the downstream-safety property is not maintained. If there is no such $i \in A \cap B$, the value of y in all rows $i \in [1, N + 1]$ will be zero, and $V = \{y\}$ will be UD-safe.

So $V = \{y\}$ is UD-safe if and only if the input sets are not disjoint. Hence we need to look at $\Omega(N)$ rows to decide whether $V = \{y\}$ is UD-safe. $\square$

This proof also shows the lower bound for verifying if a subset is downstream-safe.

A.5.8 Upper Bound to Find All UD-safe Solutions

The lower bounds studied for the second step of the four step optimization show that for a public module m_j, it is not possible to have poly-time algorithms (in $|A_j|$) even to decide if a given subset $V \subseteq A_j$ is UD-safe, unless $P = NP$. Here we present Algorithm 10 that finds *all* UD-safe solutions of m_j is time exponential in $k_j = |A_j|$, assuming that the maximum domain size of attributes Δ is a constant.

Time complexity. The outer for loop runs for all possible subsets of A_j, i.e. 2^{k_j} times. The inner for loop runs for maximum $\Delta^{|I_j \cap V|}$ times (this is the maximum number of such tuples $\mathbf{x}^+$), whereas the check if V is a valid downstream-safe subset takes $O(\Delta^{|I_j \setminus V|})$ time. Here we ignore the time complexity to check equality of tuples which will take only polynomial in $|A_i|$ time and will be dominated by the exponential terms. For the upstream-safety check, the number of $\langle \mathbf{x}^+, \mathbf{y}^+ \rangle$ pairs are at most $\Delta^{|I_j \cap V|}$, and to compute the distinct number of $\mathbf{x}^+, \mathbf{y}^+$ tuples from the pairs can be done in $O(\Delta^{2|I_j \cap V|})$ time by a naive search; the time complexity can be improved by the use of a hash function. Hence the total time complexity is dominated by $2^{k_j} \times O(\Delta^{|I_j \cap V|}) \times O(\Delta^{|I_j \setminus V|} + \Delta^{2|I_j \cap V|})$ $= O(2^{k_j} \Delta^{3k^j}) = O(2^{4k_j})$. By doing a tighter analysis, the multiplicative factor in the exponents can be improved, however, we make the point here that the algorithm runs in time exponential in $k_j = |A_j|$.

Correctness. The following lemma proves the correctness of Algorithm 10.

Lemma A.39. *Algorithm 10 adds $V \subseteq A_j$ to $UD-safe_j$ if and only if m_j is UD-safe w.r.t. V.*

Proof. (if) Suppose V is a UD-safe subset for m_j. Then V is downstream-safe, i.e. for equivalent inputs w.r.t. the visible attributes $I_j \cap V$, the projection of the output on the visible attributes $O_j \cap V$ will be the same, so V will pass the downstream-safety test.

Since V is UD-safe, V is also upstream-safe. Clearly, by definition, $n_1 \geq n_2$. Suppose $n_1 > n_2$. Then there are two $\mathbf{x}_1^+$ and $\mathbf{x}_2^+$ that pair with the same $\mathbf{y}^+$. By construction, $\mathbf{x}_1^+$ and $\mathbf{x}_2^+$ (and all input tuples $\mathbf{x}$ to m_j that project on these two tuples) have different value on the visible input attributes $I_j \cap V$, but they map to outputs $\mathbf{y}$-s that have the same value on visible output attributes $O_j \cap V$. Then V is not upstream-safe, which is a contradiction. Hence $n_1 = n_2$, and V will also pass the test for upstream-safety and be

Algorithm 10 Algorithm to find all UD-safe solutions $UD-safe_j$ for a public module m_j

```
1: – Set $UD-safe_j = \emptyset$.
2: for every subset $V$ of $A_j$ do
3:   /* Check if V is downstream-safe */
4:   for every assignment $\mathbf{x}^+$ of the visible input attributes in $I_j \cap V$ do
5:     –Check if for every assignment $\mathbf{x}^-$ of the hidden input attributes in $I_j \setminus V$, whether
       the value of $\pi_{O_j \cap V}(m_j(\mathbf{x}))$ is the same, where $\pi_{I_j \cap V}(\mathbf{x}) = \mathbf{x}^+$ and $\pi_{I_j \setminus V}(\mathbf{x}) = \mathbf{x}^-$
6:     if not then
7:       – V is not downstream-safe. Go to the next $V$.
8:     else
9:       – V is downstream-safe. Let $\mathbf{y}^+ = \pi_{O_j \cap V}(m_j(\mathbf{x}))$ = projection of all such tuples
         that have projection = $\mathbf{x}^+$ on the visible input attributes
10:      – Label this set of input-output pairs $(\mathbf{x}, m_j(\mathbf{x}))$ by $\langle \mathbf{x}^+, \mathbf{y}^+ \rangle$.
11:    end if
12:    /* Check if V is upstream-safe */
13:    – Consider the pairs $\langle \mathbf{x}^+, \mathbf{y}^+ \rangle$ constructed above. Let $n_1$ be the number of distinct
       $\mathbf{x}^+$ values, and let $n_2$ be the number of distinct $\mathbf{y}^+$ values
14:    if $n_1 == n_2$ then
15:      – V is upstream-safe. Add $V$ to $UD-safe_j$.
16:    else
17:      – V is not upstream-safe. Go to the next $V$.
18:    end if
19:  end for
20: end for
21: return The set of subsets $UD-safe_j$.
```

included in $UD-safe_j$.

(only if) Suppose V is not UD-safe, then it is either not upstream-safe or not downstream-safe. Suppose it is not downstream-safe. Then for at least one assignment $\mathbf{x}^+$, the values of $\mathbf{y}$ generated by the assignments $\mathbf{x}^-$ will not be equivalent w.r.t. the visible output attributes, and the downstream-safety test will fail.

Suppose V is downstream-safe but not upstream-safe. Then there are Then there are two $\mathbf{x}_1^+$ and $\mathbf{x}_2^+$ that pair with the same $\mathbf{y}^+$. This makes $n_1 > n_2$, and the upstream-safety test will fail. $\square$

A.5.9 Proof of Lemma 7.20

The following proposition will be useful to prove the lemma.

Proposition A.40. *For a public module m_j, for two UD-safe sets $U_1, U_2 \subseteq A_j$, if $\overline{U_1} \cap O_j = \overline{U_2} \cap O_j$, then $U_1 = U_2$.*

Proof. Assume the contradiction, and wlog. assume that there is an attribute $a \in U_1 \setminus U_2$. UD-safe condition says that if the inputs to m_j are equivalent then the corresponding outputs are also equivalent w.r.t. the visible attributes and vice versa. Let U_1 be the set visible attributes. This allows us to partition the set of outputs into K equivalence classes $Y^1, \cdots, Y^K$, and the set of inputs into K equivalent classes $X^1, \cdots, X^K$ w.r.t. the same projection on input attribute values, such that for all $1 \leq \ell \leq K$,

$$Y^\ell = \{\mathbf{y} = m_j(\mathbf{x}) : \mathbf{x} \in X^\ell\}$$

Again for visible attributes U_2, since $U_1 \cap O_j = U_2 \cap O_j$, the set of outputs can be partitioned into the same K equivalence classes $Y^1, \cdots, Y^K$, and the set of inputs into K equivalent classes $Z^1, \cdots, Z^K$ w.r.t. the same projection on input attribute values, such that for all $1 \leq \ell \leq K$,

$$Y^\ell = \{\mathbf{y} = m_j(\mathbf{x}) : \mathbf{x} \in Z^\ell\}$$

Hence

$$\{\mathbf{y} : \exists \mathbf{x} \in X^\ell, m_j(\mathbf{x}) = \mathbf{y}\} = \{\mathbf{y} : \exists \mathbf{x} \in Z^\ell, m_j(\mathbf{x}) = \mathbf{y}\}$$

and

$$X^\ell = Z^\ell$$

for all $1 \leq \ell \leq K$. Suppose not, and wlog. there is a $\mathbf{x} \in X^\ell$, i.e. $m_j(\mathbf{x}) \in Y^\ell$. Suppose $\mathbf{x} \notin Z^\ell$, i.e. $m_j(\mathbf{x}) \notin Y^\ell$ which is a contradiction.

For all ℓ, $X^\ell = Z^\ell$. Consider any tuple $\mathbf{x} \in X^\ell$. Since $a \in U_1 \setminus U_2$, and both $U_1, U_2 \subseteq A_j$, $a \in \overline{U_2} \setminus \overline{U_1}$. Consider two tuples $\mathbf{x_1}, \mathbf{x_2} \in Z^\ell$, such that $\pi_a(\mathbf{x_1}) \neq \pi_a(\mathbf{x_2})$. Since a is hidden for U_2, two such tuples must exist. But a is visible in U_1, and for all tuples in the same group X^ℓ, value of a must be the same (the reverse is not necessarily true). In other words, since the value of a is different, $\mathbf{x_1}, \mathbf{x_2}$ must belong to different groups w.r.t. U_1. But we argued that $X^\ell = Z^\ell$, so these two tuples belong to the same group. This is a contradiction, and therefore no such a, and $U_1 \subseteq U_2$. By the same argument, $U_2 \subseteq U_1$, and therefore $U_1 = U_2$. □

Lemma 7.20. *For $1 \leq j \leq k$, the entry $Q[j,\ell]$, $1 \leq \ell \leq p_j$, stores the minimum cost of the hidden attributes $\cup_{x=1}^{j} A_x \supseteq H^{j\ell} \supseteq \overline{S_{i*}}$ such that $A_j \setminus H^{j\ell} = U_{j\ell}$, and every module $m_x, 1 \leq x \leq j$ in the chain is UD-safe w.r.t. $A_x \setminus H^{j\ell}$.*

Proof. We prove this by induction from $j = 1$ to k. The base case follows by the definition of $Q[1,\ell]$, for $1 \leq \ell \leq p_1$. Here the requirements are $A_1 \supseteq H^{1\ell} \supseteq \overline{S_{i*}}$, and $A_1 \setminus H^{1\ell} = U_{1\ell}$, i.e. $\overline{U_{1\ell}} = H^{1\ell}$. So we set the cost at $Q[1,\ell]$ to $c(\overline{U_{1\ell}}) = c(H^{1\ell})$, if $\overline{U_{1\ell}} \supseteq \overline{S_{i*}}$.

Suppose the hypothesis holds until $j-1$, and consider j. Let $H^{j\ell}$ be the minimum solution s.t. $A_j \setminus H^{j\ell} = U_{j\ell}$ and satisfies the other conditions of the lemma.

First consider the case when there is no q such that $\overline{U_{j-1,q}} \cap O_{j-1} = \overline{U_{j,\ell}} \cap I_j$, where we set the cost to be ∞. If there is no such q. i.e. for all $q \leq p_{j-1}$, $\overline{U_{j-1,q}} \cap O_{j-1} \neq \overline{U_{j,\ell}} \cap I_j$, then clearly there cannot be any solution $H^{j\ell}$ that contains $\overline{U_{j,\ell}}$ and also guarantees UD-safe properties of all $x < j$ (in particular for $x = j-1$). In that case the cost of the solution is indeed ∞.

Otherwise (when such a q exists), let us divide the cost of the solution $c(H^{j\ell})$ into two disjoint parts:

$$c(H^{j\ell}) = c(H^{j\ell} \cap O_j) + c(H^{j\ell} \setminus O_j)$$

We argue that $c(O_j \cap H^{j\ell}) = c(O_j \cap \overline{U_{j\ell}})$. $A_j \setminus H^{j\ell} = U^{j\ell}$. So $\overline{U_{j\ell}} = A_j \setminus U_{j\ell} = A_j \cap H^{j\ell}$. Then $O_j \cap \overline{U_{j\ell}}$ = $O_j \cap A_j \cap H^{j\ell}$ = $O_j \cap H^{j\ell}$, since $O_j \subseteq A_j$. Hence $c(O_j \cap H^{j\ell}) = c(O_j \cap \overline{U_{j\ell}})$. This accounts for the cost of the first part of $Q[j,\ell]$.

Next we argue that $c(H^{j\ell} \setminus O_j) =$ minimum cost $Q[j-1,q]$, $1 \leq q \leq p_j$, where the minimum is over those those q where $\overline{U_{j-1,q}} \cap O_{j-1} = \overline{U_{j,\ell}} \cap I_j$. Due to the chain structure of C_W, $O_j \cap \bigcup_{x=1}^{j-1} A_j = \emptyset$, and $O_j \cup \bigcup_{x=1}^{j-1} A_x = \bigcup_{x=1}^{j} A_x$. Since $\cup_{x=1}^{j} A_x \supseteq H^{j\ell}$, $H^{j\ell} \setminus O_j = H^{j\ell} \cap \bigcup_{x=1}^{j-1} A_x$.

Consider $H' = H^{j\ell} \cap \bigcup_{x=1}^{j-1} A_x$. By definition of $H^{j\ell}$, H' must satisfy the UD-safe requirement of all $1 \leq x \leq j-1$. Further, $\bigcup_{x=1}^{j-1} A_x \supseteq H'$. $A_j \setminus H^{j\ell} = U_{j,\ell}$, hence

$$\overline{U_{j,\ell}} = A_j \setminus U_{j,\ell} = A_j \cap H^{j\ell} \subseteq H_{j\ell}$$

We are considering the case where there is a q such that

$$\overline{U_{j-1,q}} \cap O_{j-1} = \overline{U_{j,\ell}} \cap I_j \tag{A.40}$$

and therefore

$$\overline{U_{j-1,q}} \cap O_{j-1} \subseteq \overline{U_{j,\ell}} \subseteq H^{j\ell}$$

We claim that if q satisfies (A.40), then $A_{j-1} \setminus H' = U_{j-1,q}$. Therefore, by IH, $Q[j-1,\ell]$ stores the minimum cost solution H' that includes $\overline{U_{j-1,q}}$, and part of the the optimal solution cost $c(H^{j\ell} \setminus O_j)$ for m_j is the minimum value of such $Q[j-1,q]$.

So it remains to show that $A_{j-1} \setminus H' = U_{j-1,q}$. $A_{j-1} \setminus H' = A_{j-1} \setminus H^{j\ell} \in UD-safe_{j-1}$, since $H^{j\ell}$ gives UD-safe solution for m_{j-1}. Suppose $A_{j-1} \setminus H^{j\ell} = U_{j-1,y}$.

Next we argue that $U_{j-1,q} = U_{j-1,y}$, which will complete the proof. $\overline{U_{j-1},y} \cap O_{j-1} = (A_{j-1} \setminus U_{j-1,y}) \cap O_{j-1} = (A_{j-1} \cap H^{j\ell}) \cap O_{j-1} = H^{j\ell}) \cap O_{j-1}, = H^{j\ell} \cap I_j = (A_j \setminus U_{j,\ell}) \cap I_j$ i.e.

$$\overline{U_{j-1},y} \cap O_{j-1} = \overline{U_{j,\ell}} \cap I_j \tag{A.41}$$

From (A.40) and (A.41),

$$\overline{U_{j-1},q} \cap O_{j-1} = \overline{U_{j-1},y} \cap O_{j-1}$$

, since both $U_{j-1,q}, U_{j-1,y} \in UD-safe_{j-1}$, from Proposition A.40, $U_{j-1,q} = U_{j-1,y}$. This completes the proof of the lemma. □

A.5.10 Optimal Algorithm for Tree Workflows

Theorem 7.21. *The single-subset problem can be solved in PTIME (in n, $|A|$ and L) for tree workflows.*

Proof. Similar to the proof of Theorem 7.19, to obtain an algorithm of time polynomial in L, for a given module m_i, we can go over all choices of safe subsets $S_{i\ell} \in \mathbf{S}_i$ of m_i, compute the public-closure $C(\overline{S_{i\ell}})$, and choose a minimal cost subset $H_i = H_i(S_{i\ell})$ that satisfies the UD-safe properties of all modules in the public-closure. Then, output, among them, a subset having the minimum cost. Consequently, it suffices to explain how, given a safe subset $S_{i\ell} \in \mathbf{S}_i$, one can solve, in PTIME, the problem of finding a minimum cost hidden subset H_i that satisfies the UD-safe property of all modules in a subgraph formed by a given $C(\overline{S_{i\ell}})$.

To simplify notations, the given safe subset $S_{i\ell}$ will be denoted below by S_{i*}, the closure $C(\overline{S_{i\ell}})$ will be denoted by T', and the output hidden subset will be denoted by H. Our PTIME algorithm uses dynamic programming to find the optimal H.

First note that since T' is the public-closure of (some) output attributes for a tree workflow, T' is a collection of trees all of which are rooted at the private module m_i. Let us consider the tree T rooted at m_j with the subtrees in T', (note that m_i can have private children that are not considered in T). Let k be the number of modules in T, and the modules in T be renumbered as $m_i, m_1, \cdots, m_k$, where the private module m_i is the root, and the rest are public modules.

Now we solve the problem by dynamic programming as follows. Let Q be an $k \times L$ two-dimensional array, where $Q[j, \ell]$, $1 \leq j \leq k, 1 \leq \ell \leq p_j$ denotes the cost of minimum cost hidden subset $H^{j\ell}$ that (i) satisfies the UD-safe condition for all public modules in the subtree of T rooted at m_j, that we denote by T_j; and, (ii) $A_j \setminus H^{j\ell} = U_{j\ell}$. Here A_j is the attribute set of m_j); the actual solution can be stored easily by standard argument. The algorithm is described below.

- **Initialization for leaf nodes**. The initialization step handles all leaf nodes m_j in T. For a leaf node m_j, $1 \leq \ell \leq p_j$,

$$Q[j, \ell] \quad = \quad c(\overline{U_{j,\ell}})$$

- **Internal nodes**. The internal nodes are considered in a bottom-up fashion (by a post-order traversal), and $Q[j, \ell]$ is computed for a node m_j after its children are processed.

  For an internal node m_j, let $m_{i_1}, \cdots, m_{i_x}$ be its children in T. Then for $1 \leq \ell \leq p_j$,

  1. Consider UD-safe subset $U_{j,\ell}$.

2. For $y=1$ to x, let $\overline{U^y} = \overline{U_{j,\ell}} \cap I_{i_y}$ (intersection of $\overline{U_{j,\ell}}$ with the inputs of m_{i_y}). Since there is no data sharing, U^y-s are disjoint

3. For $y=1$ to x,

$$\begin{aligned} k^y &= \text{argmin}_k Q[i_y,k] \quad \text{where the minimum is over} \\ &\qquad 1 \le k \le p_{i_y} \quad \text{s.t.} \quad \overline{U_{i_y,k}} \cap I_{i_y} = \overline{U^y} \\ &= \perp \text{ (undefined)}, \qquad \text{if there is no such } k \end{aligned}$$

4. $Q[j,\ell]$ is computed as follows.

$$\begin{aligned} Q[j,\ell] &= \infty \qquad\qquad \text{if } \exists y, 1 \le y \le x, \quad k^y = \perp \\ &= c(I_j \cap \overline{U_{j\ell}}) + \sum_{y=1}^{x} Q[i_y,k^y] \qquad \text{(otherwise)} \end{aligned}$$

- **Final solution for S_{i*}.** Now consider the private module m_i that is the root of T. Recall that we fixed a safe solution S_{i*} for doing the analysis. Let $m_{i_1}, \cdots, m_{i_x}$ be the children of m_i in T (which are public modules). Similar to the step before, we consider the min-cost solutions of its children which exactly match the hidden subset $\overline{S_{i*}}$ of m_i.

1. Consider safe subset S_{i*} of m_i.

2. For $y=1$ to x, let $\overline{S^y} = \overline{S_{i*}} \cap I_{i_y}$ (intersection of $\overline{S_{i*}}$ with the inputs of m_{i_y}). Since there is no data sharing, again, S^y-s are disjoint

3. For $y=1$ to x,

$$\begin{aligned} k^y &= \text{argmin}_k Q[i_y,k] \quad \text{where the minimum is over} \\ &\qquad 1 \le k \le p_{i_y} \quad \text{s.t.} \quad \overline{U_{i_y,k}} \cap I_{i_y} \supseteq \overline{S^y} \\ &= \perp \text{ (undefined)}, \qquad \text{if there is no such } k \end{aligned}$$

4. The cost of the optimal H (let us denote that by c^*) is computed as follows.

$$\begin{aligned} c^* &= \infty \qquad\qquad \text{if } \exists y, 1 \le y \le x, \quad k^y = \perp \\ &= \sum_{y=1}^{x} Q[i_y,k^y] \qquad \text{(otherwise)} \end{aligned}$$

It is not hard to see that the trivial solution of UD-safe subsets that include all attributes of the modules gives a non-infinite cost solution by the above algorithm.

The following lemma (whose proof is given later) shows that $Q[j,\ell]$ correctly stores the desired value. Then the optimal solution H has cost $\min_{1 \leq \ell \leq p_k} Q[k,\ell]$; the corresponding solution H can be found by standard procedure.

Lemma A.41. *For $1 \leq j \leq k$, let T_j be the subtree rooted at m_j and let $\texttt{Att}_j = \bigcup_{m_q \in T_j} A_q$ the entry $Q[j,\ell]$, $1 \leq \ell \leq p_j$, stores the minimum cost of the hidden attributes $H^{j\ell} \subseteq \texttt{Att}_j$ such that $A_j \setminus H^{j\ell} = U_{j\ell}$, and every module $m_q \in T_j$, is UD-safe w.r.t. $A_q \setminus H^{j\ell}$.*

Given this lemma, the correctness of the algorithm easily follows. For hidden subset $H \supseteq \overline{S_{i*}}$ in the closure, for every public child m_{i_y} of m_i, $H \cap I_{i_y} \supseteq \overline{S_{i*}} \cap I_{i_y} = \overline{S^y}$. Further, each such m_{i_y} has to be UD-safe w.r.t. $H^{j\ell}$. In other words, for each m_{i_y}, $H \cap I_{i_y}$ must equal U_{i_y,k^y} for some $1 \leq k^y \leq p_{i_y}$. The last step in our algorithm (that computes c^*) tries to find such a k^y that has the minimum cost $Q[i_y,k^y]$, and the total cost c^* of H is $\sum_{m_{i_y}} Q[i_y,k^y]$ where the sum is over all children of m_i in the tree T (the trees rooted at m_{i_y} are disjoint, so the c^* is sum of those costs). □

To complete the proof of Theorem 7.21, we need to prove Lemma A.41, that we prove the next.

Proof of Lemma A.41

Proof. We prove the lemma by an induction on all nodes at depth $h = H$ down to 1 of the tree T, where depth H contains all leaf nodes and depth 1 contains the children of the root m_i (which is at depth 0).

First consider any leaf node m_j at height H. Then T_j contains only m_j and $\texttt{Att}_j = A_j$. For any $1 \leq \ell \leq p_j$, since $\texttt{Att}_j = A_j \supseteq H^{j\ell}$ and $A_j \setminus H^{j\ell} = U_{j,\ell}$. hence $H^{j\ell} = \overline{U_{j,\ell}}$. In this case $H^{j\ell}$ is unique and $Q[j,\ell]$ correctly stores $c(\overline{U_{j,\ell}}) = c(H^{j\ell})$.

Suppose the induction holds for all nodes up to height $h+1$, and consider a node m_j at height h. Let $m_{i_1}, \cdots, m_{i_x}$ be the children of m_j which are at height $h+1$. Let $H^{j\ell}$ be the min-cost solution, which is partitioned into two disjoint component:

$$c(H^{j\ell}) = c(H^{j\ell} \cap I_j) + c(H^{j\ell} \setminus I_j)$$

First we argue that $c(H^{j\ell} \cap I_j) = c(\overline{U_{j,\ell}})$. $A_j \setminus H^{j\ell} = U^{j\ell}$. So $\overline{U_{j\ell}} = A_j \setminus U_{j\ell} = A_j \cap H^{j\ell}$. Then $I_j \cap \overline{U_{j\ell}} = I_j \cap A_j \cap H^{j\ell} = I_j \cap H^{j\ell}$, since $I_j \subseteq A_j$. Hence $c(I_j \cap H^{j\ell}) = c(I_j \cap \overline{U_{j\ell}})$. This accounts for the cost of the first part of $Q[j,\ell]$.

Next we analyze the cost $c(H^{j\ell} \setminus I_j)$. This cost comes from the subtrees $T_{i_1}, \cdots, T_{i_x}$ which are disjoint due to the tree structure and absence of data-sharing. Let us partition the subset $H^{j\ell} \setminus I_j$ into disjoint parts $(H^{j\ell} \setminus I_j) \cap \texttt{Att}_{i_y}$, $1 \leq y \leq x$. We claim that $c((H^{j\ell} \setminus I_j) \cap \texttt{Att}_{i_y}) = Q[i_y, k^y]$, $1 \leq y \leq x$, where k^y is computed as in the algorithm. This will complete the proof of the lemma.

Proof of claim: $c((H^{j\ell} \setminus I_j) \cap \mathbf{Att}_{i_y}) = Q[i_y, k^y]$, $1 \leq y \leq x$. Let $H' = (H^{j\ell} \setminus I_j) \cap \texttt{Att}_{i_y}$. Clearly, $\texttt{Att}_{i_y} \supseteq H'$. Since every $m_q \in T_j$ is UD-safe w.r.t. $A_q \setminus H^{j\ell}$. If also $m_q \in T_{i_y}$, then $A_q \setminus H' = A_q \setminus H^{j\ell}$, and therefore all $m_q \in T_{i_y}$ are also UD-safe w.r.t. H'. In particular, m_{i_y} is UD-safe w.r.t. H', and therefore $A_{i_y} \setminus H' = U_{i_y, k^y}$, since U_{i_y,k^y} was chosen as the UD-safe set by our algorithm.

Finally we argue that $c(H') = c(H^{i_y,k^y})$, where H^{i_y,k^y} is the min-cost solution for m_{i_y} among all such subsets. This easily follows from our IH, since m_{i_y} is a node at depth $h+1$. Therefore, $c(H') = c(H^{i_y,k^y}) = Q[i_y, k^y]$, i.e.

$$c((H^{j\ell} \setminus I_j) \cap \texttt{Att}_{i_y}) = Q[i_y, k^y]$$

as desired. This completes the proof of the lemma. $\square$

A.5.11 Proof of Theorem 7.22

Theorem 7.22. *The problem of testing whether the single-subset problem has a solution with cost smaller than a given bound is NP-complete when the public-closure forms an arbitrary subgraph. This is the case even when both number of attributes and the number of safe and UD-safe subsets of the individual modules is bounded by a (small) constant.*

Proof. Given a CNF formula ψ on n variables $z_1, \cdots, z_n$ and m clauses $\psi_1, \cdots, \psi_m$, we construct a graph as shown in Figure A.6. Let variable z_i occurs in m_i different clauses (as positive or negative literals). In the figure, the module p_0 is the single-private module

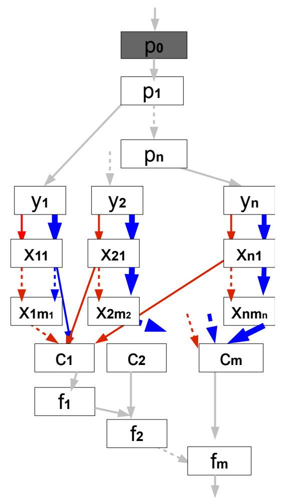


Figure A.6: Reduction from 3SAT. Module p_0 is private, the rest are public. The narrow dark edges (in red) denote TRUE assignment, the bold dark edges (in blue) denote FALSE assignment.

(m_i), having a single output attribute a. The rest of the modules are the public modules in the public-closure $C(\{a\})$.

For every variable z_i, we create $m_i + 2$ nodes: p_i,y_i and $x_{i,1}, \cdots, x_{i,m_i}$. For every clause ψ_j, we create 2 modules C_j and f_j.

The **edge connections** are as follows:

1. p_0 sends its single output a to p_1.

2. For every $i = 1$ to $n - 1$, p_i has two outputs; one is sent to p_{i+1} and the other is sent to y_i. p_n sends its single output to y_n.

3. Each y_i, $i = 1$ to n, sends two outputs to $x_{i,1}$. The blue outgoing edge from y_i denotes positive assignment of the variable z_i, whereas the red edge denotes negative assignment of the variable z_i.

4. Each $x_{i,j}$, $i = 1$ to n, $j = 1$ to $m_i - 1$, sends two outputs (blue and red) to $x_{i,j+1}$. In addition, if $x_{i,j}$, $i = 1$ to n, $j = 1$ to m_i sends a blue (resp. red) edge to clause node C_k if the variable z_i is a positive (resp. negative) in the clause C_k (and C_k is the j-th such clause containing z_i).

5. Each C_j, $j = 1$ to m, sends its single output to f_j.

6. Each f_j, $j = 1$ to $m - 1$, sends its single output to f_{j+1}, f_m outputs the single final output.

The **UD-safe** sets are defined as follows:

1. For every $i = 1$ to $n - 1$, p_i has a single UD-safe set: hide all its inputs and outputs.

2. Each y_i, $i = 1$ to n, has three UD-safe choices: (1) hide its unique input and blue output, (2) hide its unique input and red output, (3) hide its single input and both blue and red outputs.

3. Each $x_{i,j}$, $i = 1$ to n, $j = 1$ to m_i, has three choices: (1) hide blue input and all blue outputs, (2) hide red input and all red outputs, (3) hide all inputs and all outputs.

4. Each C_j, $j = 1$ to m, has choices: hide the single output and at least of the three inputs.

5. Each f_j, $j = 1$ to m, has the single choice: hide all its inputs and outputs.

Cost. The outputs from y_i, $i = 1$ to n has unit cost, the cost of the other attributes is 0.

The claim is that there is a solution of single-module problem of cost $= n$ **if and only if the 3SAT formula** ψ **is satisfiable**.

(if) Suppose the 3SAT formula is satisfiable, so there is an assignment of the variables z_i that makes Ψ true. If z_i is set to True (resp False), choose the blue (resp. red) outgoing edge from y_i. Then choose the other edges accordingly: (1) choose outgoing edge from p_0, (2) choose all input and outputs of p_i, $i = 1$ to n; (3) if blue (resp. red) input of $x_{i,j}$ is chosen, all its blue (resp. red) outputs are chosen; and, (4) all inputs and outputs of f_j are chosen. Clearly, all these are UD-safe sets by construction.

So we have to only argue about the clause nodes C_j. Since ψ is satisfied by the given assignment, there is a literal $z_i \in C_j$ (positive or negative), whose assignment makes it true. Hence at least one of the inputs to C_j will be chosen. So the UD-safe requirements of all the UD-safe clauses are satisfied. The total cost of the solution is n since exactly one output of the y_i nodes, $i = 1$ to n, have been chosen.

(only if) Suppose there is a solution to the single-module problem of cost n. Then each y_i can choose exactly one output (at least one output has to be chosen to satisfy UD-safe property for each y_i, and more than one output cannot be chosen as the cost is n). If y_i chooses blue (resp. red) output, this forces the $x_{i,j}$ nodes to select the corresponding blue (resp. red) inputs and outputs. No $x_{i,j}$ can choose the UD-safe option of selecting all its inputs and outputs as in that case finally y_i be forced to select both outputs which will exceed the cost. Since C_j satisfies UD-safe condition, this in turn forces each C_j to select the corresponding blue (resp. red) inputs.

If the blue (resp. red) output of y_i is chosen, the variable is set to True (resp. False). By the above argument, at least one such red or blue input will be chosen as input to each C_j, that satisfies the corresponding clause ψ_j. □

A.5.12 General Workflows

In this section we discuss the privacy theorem for general workflows as outlined in Section 7.5.

First we define directed-path and downward-closure as follows (similar to public path and public-closure)

Definition A.42. *A module m_1 has* **a directed** *path to another module m_2, if there are modules $m_{i_1}, m_{i_2}, \cdots, m_{i_j}$ such that $m_{i_1} = m_1$, $m_{i_j} = m_2$, and for all $1 \leq k < j$, $O_{i_k} \cap I_{i_{k+1}} \neq \emptyset$.*

An attribute $a \in A$ has a directed path from to module m_j, if there is a module m_k such that $a \in I_k$ and m_k has a directed path to m_j.

Definition A.43. *Given a private module m_i and a set of hidden output attributes $h_i \subseteq O_i$ of m_i, the* **downward-closure** *of m_i w.r.t. h_i, denoted by $D(h_i)$, is the set of modules m_j (both private and public) such that there is a directed path from some attribute $a \in h_i$ to m_j.*

The goal of this section is to prove the following theorem:

Theorem A.44. (**Privacy Theorem for General workflows**) *Let W be any workflow. For a private module m_i in W, let V_i be a safe subset for Γ-standalone-privacy s.t. only output attributes of m_i are hidden (i.e. $\overline{V_i} \subseteq O_i$). Let $D(\overline{V_i})$ be the downward-closure of m_i w.r.t. hidden attributes $\overline{V_i}$. Let H_i be a set of hidden attributes s.t. $\overline{V_i} \subseteq H_i \subseteq O_i \cup \bigcup_{k \in D(\overline{V_i})} A_k$ and where for every module $m_j \in D(\overline{V_i})$ (m_j can be private or public), m_j is downstream-safe (D-safe) w.r.t. $A_j \setminus H_i$. Then the workflow W is Γ-private w.r.t the set of visible attributes $V = A \setminus (\bigcup_{i \in M^-} H_i)$.*

In the proof of Theorem 7.10 from Lemma 7.12 used the fact that for single-predecessor workflows, for two distinct private modules m_i, m_k, the public-closures and the hidden subsets H_i, H_j are disjoint. However, it is not hard to see that this is not the case for general workflows, where the downward-closure and the subsets H_i may overlap. Further, the D-safe property is not monotone (hiding more output attributes will maintain the D-safe property, but hiding more input attributes may destroy the D-safe property). So we need to argue that the D-safe property is maintained when we take union of H_i sets in the workflow. The following proposition will be helpful to prove the privacy theorem for general workflows.

Lemma A.45. *If a module m_j is D-safe w.r.t. sets $V_1, V_2 \subseteq A_j$, then m_j is D-safe w.r.t. $V = V_1 \cap V_2$.*

Proof. (Sketch) Given two equivalent inputs $\mathbf{x_1} \equiv_V \mathbf{x_2}$ w.r.t. $V = V_1 \cap V_2$, we have to show that their outputs are equivalent: $m_j(\mathbf{x_1}) \equiv_V m_j(\mathbf{x_2})$. Even if $\mathbf{x_1}, \mathbf{x_2}$ are equivalent w.r.t. V, they may not be equivalent w.r.t. V_1 or V_2. In the proof we construct a new tuple $\mathbf{x_3}$ such that $\mathbf{x_1} \equiv_{V_1} \mathbf{x_3}$, and $\mathbf{x_2} \equiv_{V_2} \mathbf{x_3}$. Then using the D-safe properties of V_1 and V_2, we show that $m_j(\mathbf{x_1}) \equiv_V m_j(\mathbf{x_3}) \equiv_V m_j(\mathbf{x_2})$. □

The full proof is given in Appendix A.5.13.

Along with this lemma, two other simple observations will be useful.

Observation A.46.

1. Any module m_j is D-safe w.r.t. A_j (hiding nothing maintains downstream-safety property).

2. If m_j is D-safe w.r.t. V, and if V' is such that $V' \subseteq V$, but $V' \cap I_j = V \cap I_j$, then m_j is also D-safe w.r.t. V' (hiding more output attributes maintains downstream-safety property).

Proof of Theorem A.44. Using Lemma A.47 and Lemma A.45, we present the proof of Theorem A.44 below.

Proof. We again argue that if H_i satisfies the conditions in Theorem A.44, then $H'_i = \bigcup_{\ell \in M^-} H_\ell$ satisfies the conditions in Lemma A.47. The first two conditions are easily satisfied by H'_i: (i) $\overline{V_i} \subseteq H_i \subseteq \bigcup_{\ell \in M^-} H_\ell = H'_i$; (ii) $\overline{V_i} \subseteq O_i$ is unchanged.

Now we argue that (iii) every module m_j in the downward-closure $D(\overline{V_i})$ is D-safe w.r.t. H'_i. From the conditions in the theorem, each module $m_j \in D(\overline{V_i})$ is D-safe w.r.t. $A_j \setminus H_i$. We show that for any other private module $m_k \neq m_i$, m_j is also D-safe w.r.t. $A_j \setminus H_k$. There may be three such cases as discussed below.

Case-I: If $m_j \in D(\overline{V_k})$, by the D-safe conditions in the theorem, m_j is D-safe w.r.t. $A_j \setminus H_k$.

Case-II: If $m_j \notin D(\overline{V_k})$ and $m_j \neq m_k$, for any private module $m_k \neq m_i$, then $A_j \cap H_k = \emptyset$ (since $H_k \subseteq O_k \cup \bigcup_{\ell \in D(\overline{V_k})} A_\ell$ from the theorem). Hence $A_j \setminus H_k = A_j$, and from Observation A.46, m_j is D-safe w.r.t. $A_j \setminus H_k$.

Case-III: If $m_j \notin D(\overline{V_k})$ but $m_j = m_k$ (or $j = k$), then $H_k \cap A_j \subseteq O_j$ (again since $H_k \subseteq O_k \cup \bigcup_{\ell \in D(\overline{V_k})} A_\ell$ and $O_k = O_j$). From Observation A.46, m_j is UD-safe w.r.t. A_j, and $A_j \setminus H_k \subseteq A_j$ is such that $A_j \cap I_j = I_j = (A_j \setminus H_k) \cap I_j$. Then again from the same observation, m_j is UD-safe w.r.t. $A_j \setminus H_k$.

Hence m_j is D-safe w.r.t. $A_j \setminus H_i$ and for all $m_k \in M^-$, $m_k \neq m_i$, m_j is D-safe w.r.t. $A_j \setminus H_k$. By Lemma A.45, then m_j is D-safe w.r.t. $(A_j \setminus H_i) \cap (A_j \setminus H_k) = A_j \setminus (H_i \cup H_k)$. (this is by the simple fact that $(A_j \setminus H_i) \cap (A_j \setminus H_k)$ = $(A_j \cap \overline{H_i}) \cap (A_j \cap \overline{H_k})$ = $A_j \cap \overline{H_i \cup H_k}$ = $A_j \setminus (H_i \cup H_j)$). By a simple induction on all $k \in M^-$, m_j is D-safe w.r.t. $A_j \setminus (\bigcup_{k \in M^-}) H_k$ = $A_j \setminus H'_i$. Hence H'_i satisfies the conditions stated in the lemma. The rest of the proof follows by the same argument as in the proof of Theorem 7.10.

Now the proof can be completed using exactly the same argument as in Theorem 7.10. Theorem A.44 also states that each private module m_i is Γ-standalone-private w.r.t. visible attributes V_i, i.e., $|\texttt{OUT}_{\mathbf{x},m_i}| \geq \Gamma$ for all input $\mathbf{x}$ to m_i, for $i \in M^-$ (see Definition 6.3). From Lemma 7.12, using H'_i in place of H_i, this implies that for all input $\mathbf{x}$ to private modules m_i, $|\texttt{OUT}_{\mathbf{x},W}| \geq \Gamma$ w.r.t $V = A \setminus H'_i = A \setminus \bigcup_{\ell \in M^-} H_\ell$. From Definition 6.6, this implies that each $m_i \in M^-$ is Γ-workflow-private w.r.t. $V = A \setminus \bigcup_{i \in M^-} H_i$; equivalently W is Γ-private w.r.t. V. $\square$

Again, similar to Lemma 7.12, we present the following lemma that allows us to analyze every private module separately to prove Theorem A.44.

Lemma A.47. *Consider any workflow W; any private module m_i in W and any input $\mathbf{x} \in \pi_{I_i}(R)$; and any $\mathbf{y} \in \texttt{OUT}_{\mathbf{x},m_i}$ w.r.t. a set of visible attributes V_i. Given a set of hidden attributes $H_i \subseteq A$, such that (i) the hidden attributes $\overline{V_i} \subseteq H_i$, (ii) only output attributes from O_i are included in $\overline{V_i}$ (i.e. $\overline{V_i} \subseteq O_i$), and (iii) every module m_j (private or public) in the downward-closure $D(\overline{V_i})$ is D-safe w.r.t. $A_j \setminus H_i$. Then $\mathbf{y} \in \texttt{OUT}_{\mathbf{x},W}$ w.r.t. visible attributes $V = A \setminus H_i$.*

Proof. We fix a module m_i, an input $\mathbf{x}$ to m_i and a candidate output $\mathbf{y} \in \texttt{OUT}_{\mathbf{x},m_i}$ for $\mathbf{x}$ w.r.t. visible attributes V_i. for simplicity, let us refer to the set of modules in $D(\overline{V_i})$ by D. We will show that $\mathbf{y} \in \texttt{OUT}_{\mathbf{x},W}$ w.r.t. visible attributes $V = A \setminus \overline{H_i}$ by showing the existence of a possible world $R' \in \texttt{Worlds}(R, V)$, s.t. if $\pi_{I_i}(\mathbf{t}) = \mathbf{x}$ for some $\mathbf{t} \in R'$, then $\pi_{O_i}(\mathbf{t}) = \mathbf{y}$. Since $\mathbf{y} \in \texttt{OUT}_{x,m_i}$, by Lemma 7.14, $\mathbf{y} \equiv_{V_i} \mathbf{z}$ where $\mathbf{z} = m_i(\mathbf{x})$.

We will use the $\texttt{FLIP}$ function used in the proof of Lemma 7.12 (see Appendix A.5.3). We redefine the module m_i to $\widehat{m}_i$ as follows. For an input $\mathbf{u}$ to m_i, $\widehat{m}_i(\mathbf{u}) = \texttt{FLIP}_{\mathbf{y},\mathbf{z}}(m_i(\mathbf{u}))$. All other public and private modules are unchanged, $\widehat{m}_j = m_j$. The required possible world R' is obtained by taking the join of the standalone relations of these $\widehat{m}_j$-s, $j \in [n]$.

First note that by the definition of $\widehat{m}_i$, $\widehat{m}_i(\mathbf{x}) = \mathbf{y}$ (since $\widehat{m}_i(x) = \texttt{FLIP}_{\mathbf{y},\mathbf{z}}(m_i(x)) = \texttt{FLIP}_{\mathbf{y},\mathbf{z}}(\mathbf{z}) = \mathbf{y}$, from Observation A.33). Hence if $\pi_{I_i}(\mathbf{t}) = \mathbf{x}$ for some $\mathbf{t} \in R'$, then $\pi_{O_i}(\mathbf{t}) = \mathbf{y}$.

Next we argue that $R' \in \texttt{Worlds}(R, V)$. Since R' is the join of the standalone relations for modules $\widehat{m}_j$-s, R' maintains all functional dependencies $I_j \to O_j$. Also none of the public modules are unchanged, hence for any public module m_j and any tuple t in R', $\pi_{O_j}(\mathbf{t}) = m_j(\pi_{I_j}(\mathbf{t}))$. So we only need to show that the projection of R and R' on the visible attributes are the same.

Let us assume, wlog. that the modules are numbered in topologically sorted order. Let I_0 be the initial input attributes to the workflow, and let p be a tuple defined on I_0. There are two unique tuples $\mathbf{t} \in R$ and $\mathbf{t}' \in R'$ such that $\pi_{I_1}(\mathbf{t}) = \pi_{I_1}(\mathbf{t}') = \mathbf{p}$. Note that any intermediate or final attribute $a \in A \setminus I_0$ belongs to O_j, for a unique $j \in [1, n]$ (since for $j \neq \ell$, $O_j \cap O_\ell = \phi$). So it suffices to show that t, t' projected on O_j are equivalent w.r.t. visible attributes for all module j, $j = 1$ to n.

Let $\mathbf{c}_{j,m}, \mathbf{c}_{j,\widehat{m}}$ be the values of input attributes I_j and $\mathbf{d}_{j,m}, \mathbf{d}_{j,\widehat{m}}$ be the values of output attributes O_j of module m_j, in $\mathbf{t} \in R$ and $\mathbf{t}' \in R'$ respectively on initial input attributes $\mathbf{p}$ (i.e. $\mathbf{c}_{j,m} = \pi_{I_j}(\mathbf{t})$, $\mathbf{c}_{j,\widehat{m}} = \pi_{I_j}(\mathbf{t}')$, $\mathbf{d}_{j,m} = \pi_{O_j}(\mathbf{t})$ and $\mathbf{d}_{j,\widehat{m}} = \pi_{O_j}(\mathbf{t}')$). We prove by induction on $j = 1$ to n that

$$\mathbf{d}_{j,\widehat{m}} \equiv_V (\mathbf{d}_{j,m}) \qquad \text{if } j = i \text{ or } m_j \in D \tag{A.42}$$

$$\mathbf{d}_{j,\widehat{m}} = \mathbf{d}_{j,m} \qquad \text{otherwise} \tag{A.43}$$

If the above is true for all j, then $\pi_{O_j}(\mathbf{t}) \equiv_V \pi_{O_j}(\mathbf{t})$, along with the fact that the initial inputs $\mathbf{p}$ are the same, this implies that $\mathbf{t} \equiv_V \mathbf{t}'$.

Proof of (A.43) and (A.43). The base case follows for $j = 1$. If $m_1 \neq m_i$ (m_j can be public or private), then $I_1 \cap O_i = \emptyset$, so for all input $\mathbf{u}$, $\widehat{m}_j(\mathbf{u}) = m_j(\textsc{Flip}_{\mathbf{y},\mathbf{z}}(\mathbf{u})) = m_j(\mathbf{u})$. Since the inputs $\mathbf{c}_{1,\widehat{m}} = \mathbf{c}_{1,m}$ (both projections of initial input p on I_1), the outputs $\mathbf{d}_{1,\widehat{m}} = \mathbf{d}_{1,m}$. This shows (A.43). If $m_1 = m_i$, the inputs are the same, and by definition of $\widehat{m}_1$, $\mathbf{d}_{1,\widehat{m}} = \widehat{m}_1(\mathbf{c}_{1,\widehat{m}}) = \textsc{Flip}_{\mathbf{y},\mathbf{z}}(m_i(\mathbf{c}_{1,\widehat{m}})) = \textsc{Flip}_{\mathbf{y},\mathbf{z}}(m_i(\mathbf{c}_{1,m})) = \textsc{Flip}_{\mathbf{y},\mathbf{z}}(\mathbf{d}_{1,m})$. Since $\mathbf{y}, \mathbf{z}$ only differ in the hidden attributes, by the definition of the $\textsc{Flip}$ function $\mathbf{d}_{1,\widehat{m}} \equiv_V \mathbf{d}_{1,m}$. This shows (A.43). Note that the module m_1 cannot belong to D since then it will have predecessor m_i and cannot be the first module in topological order.

Suppose the hypothesis holds until $j - 1$, consider m_j. There will be three cases to consider.

(i) If $j = i$, for all predecessors m_k of m_i ($O_k \cap I_i \neq \emptyset$), $k \neq i$ and $m_k \notin D$, since the workflow is a DAG. Therefore from (A.43), using the induction hypothesis, $\mathbf{c}_{i,\widehat{m}} = \mathbf{c}_{i,m}$. Hence $\mathbf{d}_{i,\widehat{m}} = \widehat{m}_i(\mathbf{c}_{i,\widehat{m}}) = \textsc{Flip}_{\mathbf{y},\mathbf{z}}(m_i(\mathbf{c}_{i,\widehat{m}})) = \textsc{Flip}_{\mathbf{y},\mathbf{z}}(m_i(\mathbf{c}_{i,m})) = \textsc{Flip}_{\mathbf{y},\mathbf{z}}(\mathbf{d}_{i,m})$. Again, $\mathbf{y}, \mathbf{z}$ are equivalent w.r.t. V, so $\mathbf{d}_{i,\widehat{m}} \equiv_V \mathbf{d}_{i,m}$. This shows (A.43) in the inductive step.

(ii) If $j \neq i$ ($\widehat{m}_j = m_j$) and $m_j \notin D$, then m_j does not get any of its inputs from any module in D, or any hidden attributes from m_i (then by the definition of D, $m_j \in D$). Using IH, from (A.43) and from (A.43), using the fact that $\mathbf{y}, \mathbf{z}$ are equivalent on visible attributes, $\mathbf{c}_{j,\widehat{m}} = \mathbf{c}_{j,m}$. Then $\mathbf{d}_{j,\widehat{m}} = m_j(\mathbf{c}_{j,\widehat{m}}) = m_j(\mathbf{c}_{j,m}) = \mathbf{d}_{j,m}$. This shows (A.43) in the inductive step.

(iii) If $j \neq i$, but $m_j \in D$, m_j can get all its inputs either from m_i, from other modules in D, or from modules not in D. Using the IH from (A.43) and (A.43), $\mathbf{c}_{j,\widehat{m}} \equiv_V \mathbf{c}_{j,m}$. Since $m_j \in D$, by the condition of the lemma, m_j is D-safe w.r.t. V. Therefore the corresponding outputs $\mathbf{d}_{j,\widehat{m}} = m_j(\mathbf{c}_{j,\widehat{m}})$ and $\mathbf{d}_{j,m} = m_j(\mathbf{c}_{j,m})$ are equivalent, or $\mathbf{d}_{j,\widehat{m}} \equiv_V \mathbf{d}_{j,m}$. This again shows (A.43) in the inductive step.

Hence the IH holds for all $j = 1$ to n and this completes the proof of the lemma. □

A.5.13 Proof of Lemma A.45

LEMMA A.45. *If a module m_j is D-safe w.r.t. sets $V_1, V_2 \subseteq A_j$, then m_j is D-safe w.r.t. $V = V_1 \cap V_2$.*

Proof. Let $V = V_1 \cap V_2 = A_j \setminus (\overline{V_1} \cup \overline{V_2})$. Let $\mathbf{x_1}$ and $\mathbf{x_2}$ be two input tuples to m_j such that $\mathbf{x_1} \equiv_V \mathbf{x_2}$. i.e.

$$\pi_{V \cap I_j}(\mathbf{x_1}) = \pi_{V \cap I_j}(\mathbf{x_2}) \tag{A.44}$$

For $a \in I_j$, let $x_3[a]$ denote the value of a-th attribute of x_3 (similarly $x_1[a], x_2[a]$). From (A.44), for $a \in V \cap I_j$, $x_1[a] = x_2[a]$. Let us define a tuple $\mathbf{x_3}$ as follows on four disjoint subsets of I_j (since $V = V_1 \cap V_2$):

$$\begin{aligned} x_3[a] &= x_1[a] \quad \text{if } a \in I_j \setminus (V_1 \cup V_2) \\ &= x_1[a] \quad \text{if } a \in I_j \cap (V_1 \setminus V_2) \\ &= x_2[a] \quad \text{if } a \in I_j \cap (V_2 \setminus V_1) \\ &= x_1[a] = x_2[a] \quad \text{if } a \in I_j \cap V \end{aligned}$$

For instance, on attribute set $I_j = \langle a_1, \cdots, a_5 \rangle$, let $\mathbf{x_1} = \langle \underline{2}, \underline{3}, \underline{2}, 6, 7 \rangle$, $\mathbf{x_2} = \langle \underline{4}, \underline{5}, \underline{9}, 6, 7 \rangle$, $V_1 = \{a_3, a_4, a_5\}$ and $V_2 = \{a_1, a_4, a_5\}$, $V = \{a_4, a_5\}$ (in $\mathbf{x_1}, \mathbf{x_2}$, the hidden attribute values in $\overline{V} = \{a_1, a_2, a_3\}$ are underlined). Then $\mathbf{x_3} = \langle \underline{4}, \underline{3}, \underline{2}, 6, 7 \rangle$.

(1) First we claim that, $\mathbf{x_1} \equiv_{V_1} \mathbf{x_3}$, or,

$$\pi_{V_1 \cap I_j}(\mathbf{x_1}) = \pi_{V_1 \cap I_j}(\mathbf{x_3}) \tag{A.45}$$

Partition $V_1 \cap I_j$ into two disjoint subsets, $I_j \cap (V_1 \setminus V_2)$, and, $I_j \cap (V_1 \cup V_2) = I_j \cap V$. From the definition of $\mathbf{x_3}$, for all $a \in I_j \cap (V_1 \setminus V_2)$ and all $a \in I_j \cap V$, $x_1[a] = x_3[a]$. This shows (A.45).

(2) Next we claim that, $\mathbf{x_2} \equiv_{V_2} \mathbf{x_3}$, or,

$$\pi_{V_2 \cap I_j}(\mathbf{x_2}) = \pi_{V_2 \cap I_j}(\mathbf{x_3}) \tag{A.46}$$

Again partition $V_2 \cap I_j$ into two disjoint subsets, $I_j \cap (V_2 \setminus V_1)$, and, $I_j \cap (V_1 \cup V_2) = I_j \cap V$. From the definition of $\mathbf{x_3}$, for all $a \in I_j \cap (V_2 \setminus V_1)$ and all $a \in I_j \cap V$, $x_2[a] = x_3[a]$. This shows (A.46). (A.45) and (A.46) can also be verified from the above example.

(3) Now by the condition stated in the lemma, m_j is D-safe w.r.t. V_1 and V_2. Therefore, using (A.45) and (A.46), $m_j(\mathbf{x_1}) \equiv_{V_1} m_j(\mathbf{x_3})$ and $m_j(\mathbf{x_2}) \equiv_{V_2} m_j(\mathbf{x_3})$ or,

$$\pi_{V_1 \cap O_j}(m(\mathbf{x_1})) = \pi_{V_1 \cap O_j}(m_j(\mathbf{x_3})) \tag{A.47}$$

and

$$\pi_{V_2 \cap O_j}(m(\mathbf{x_2})) = \pi_{V_2 \cap O_j}(m_j(\mathbf{x_3})) \tag{A.48}$$

Since $V \cap O_j = (V_1 \cap V_2) \cap O_j \subseteq V_1 \cap O_j$, from (A.47)

$$\pi_{V \cap O_j}(m(\mathbf{x_1})) = \pi_{V \cap O_j}(m_j(\mathbf{x_3})) \tag{A.49}$$

Similarly, $V \cap O_j \subseteq V_2 \cap O_j$, from (A.48)

$$\pi_{V \cap O_j}(m(\mathbf{x_2})) = \pi_{V \cap O_j}(m_j(\mathbf{x_3})) \tag{A.50}$$

From (A.49) and (A.50),

$$\pi_{V \cap O_j}(m(\mathbf{x_1})) = \pi_{V \cap O_j}(m_j(\mathbf{x_2})) \tag{A.51}$$

In other words, the output tuples $m(\mathbf{x_1})$,$m(\mathbf{x_2})$, that are defined on attribute set O_j,

$$m(\mathbf{x_1}) \equiv_V m_j(\mathbf{x_2}) \tag{A.52}$$

Since we started with two arbitrary input tuples $\mathbf{x_1} \equiv_V \mathbf{x_2}$, this shows that for all equivalent input tuples the outputs are also equivalent. In other words, m_j is D-safe w.r.t. $V = V_1 \cap V_2$. $\square$

Bibliography

[1] In *www.mturk.com*.

[2] In *www.myexperiment.org*.

[3] In *www.census.gov*.

[4] In *www.geonames.org*.

[5] *Automatic Content Extraction 2005 Evaluation Dataset*. 2005.

[6] Charu C. Aggarwal and Philip S. Yu. *Privacy-Preserving Data Mining: Models and Algorithms*. Springer, 2008.

[7] Gagan Aggarwal, Tomás Feder, Krishnaram Kenthapadi, Samir Khuller, Rina Panigrahy, Dilys Thomas, and An Zhu. Achieving anonymity via clustering. In *PODS*, pages 153–162, 2006.

[8] Eugene Agichtein and Luis Gravano. *Snowball*: Extracting Relations from Large Plain-Text Collections. In *ACM DL*, pages 85–94, 2000.

[9] Paola Alimonti and Viggo Kann. Some apx-completeness results for cubic graphs. *TCS*, 237(1-2):123 – 134, 2000.

[10] Lyublena Antova, Thomas Jansen, Christoph Koch, and Dan Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, pages 983–992, 2008.

[11] Sanjeev Arora, L Babai, Jacques Stern, and Z. Sweedyk. The hardness of approximate optimia in lattices, codes, and systems of linear equations. In *FOCS*, pages 724–733, 1993.

[12] Naveen Ashish, Sharad Mehrotra, and Pouria Pirzadeh. XAR: An Integrated Framework for Information Extraction. In *WRI Wold Congress on Computer Science and Information Engineering*, 2009.

[13] Lars Backstrom, Cynthia Dwork, and Jon M. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW*, pages 181–190, 2007.

[14] Zhuowei Bao, Susan B. Davidson, Sanjeev Khanna, and Sudeepa Roy. An optimal labeling scheme for workflow provenance using skeleton labels. In *SIGMOD*, pages 711–722, 2010.

[15] Zhuowei Bao, Susan B. Davidson, and Tova Milo. Labeling recursive workflow executions on-the-fly. In *SIGMOD Conference*, pages 493–504, 2011.

[16] Zhuowei Bao, Susan B. Davidson, and Tova Milo. Labeling workflow views with fine-grained dependencies. In *Accepted in VLDB*, 2012.

[17] Omar Benjelloun, Anish Das Sarma, Alon Halevy, and Jennifer Widom. Uldbs: databases with uncertainty and lineage. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 953–964. VLDB Endowment, 2006.

[18] Deepavali Bhagwat, Laura Chiticariu, Wang-Chiew Tan, and Gaurav Vijayvargiya. An annotation management system for relational databases. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 900–911. VLDB Endowment, 2004.

[19] Olivier Biton, Sarah Cohen-Boulakia, Susan Davidson, and Carmem Hara. Querying and managing provenance through user views in scientific workflows. In *ICDE*, 2008.

[20] Olivier Biton, Susan B. Davidson, Sanjeev Khanna, and Sudeepa Roy. Optimizing user views for workflows. In *ICDT '09: Proceedings of the 12th International Conference on Database Theory*, pages 310–323, 2009.

[21] Barbara T. Blaustein, Adriane Chapman, Len Seligman, M. David Allen, and Arnon Rosenthal. Surrogate parenthood: Protected and informative graphs. *PVLDB*, 4(8):518–527, 2011.

[22] Béla Bollobás. *Modern Graph Theory*. Springer, corrected edition, July 1998.

[23] Magnus Bordewich, Martin Dyer, and Marek Karpinski. Path coupling using stopping times and counting independent sets and colorings in hypergraphs. *Random Struct. Algorithms*, 32:375–399, May 2008.

[24] Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv.*, 37(1):1–28, March 2005.

[25] Jihad Boulos, Nilesh N. Dalvi, Bhushan Mandhani, Shobhit Mathur, Christopher Ré, and Dan Suciu. Mystiq: a system for finding more answers by using probabilities. In *SIGMOD Conference*, pages 891–893, 2005.

[26] S. Bowers and B. Ludäscher. Actor-oriented design of scientific workflows. In *Int. Conf. on Concept. Modeling*, pages 369–384, 2005.

[27] Shawn Bowers, Timothy M. McPhillips, and Bertram Ludäscher. Provenance in collection-oriented scientific workflows. In *Concurrency and Computation: Practice and Experience*. Wiley, 2007 (in press).

[28] Uri Braun, Avraham Shinnar, and Margo Seltzer. Securing provenance. In *USENIX HotSec*, pages 1–5, 2008.

[29] Anna Bretscher, Derek Corneil, Michel Habib, and Christophe Paul. A simple linear time LexBFS cograph recognition algorithm. *SIAM J. Discrete Math.*, 22(4):1277–1296, 2008.

[30] Randal E. Bryant. Symbolic manipulation of boolean functions using a graphical representation. In *DAC*, pages 688–694, 1985.

[31] Russ Bubley and Martin Dyer. Graph orientations with no sink and an approximation for a hard case of #sat. In *SODA*, pages 248–257, 1997.

[32] Peter Buneman, Adriane Chapman, and James Cheney. Provenance management in curated databases. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 539–550, New York, NY, USA, 2006. ACM.

[33] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *Proceedings of the 8th International Conference on Database Theory*, ICDT '01, pages 316–330, London, UK, UK, 2001. Springer-Verlag.

[34] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. On propagation of deletions and annotations through views. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 150–158, New York, NY, USA, 2002. ACM.

[35] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. On propagation of deletions and annotations through views. In *PODS*, pages 150–158, 2002.

[36] Peter Buneman and Wang-Chiew Tan. Provenance in databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 1171–1173, New York, NY, USA, 2007. ACM.

[37] Tyrone Cadenhead, Murat Kantarcioglu, and Bhavani M. Thuraisingham. A framework for policies over provenance. In *Workshop on the Theory and Practice of Provenance*, 2011.

[38] Tyrone Cadenhead, Vaibhav Khadilkar, Murat Kantarcioglu, and Bhavani M. Thuraisingham. A language for provenance access control. In *CODASPY*, pages 133–144, 2011.

[39] Tyrone Cadenhead, Vaibhav Khadilkar, Murat Kantarcioglu, and Bhavani M. Thuraisingham. Transforming provenance using redaction. In *SACMAT*, pages 93–102, 2011.

[40] A. Campan and T. M. Truta. A clustering approach for data and structural anonymity in social networks. In *PinKDD'08, in Conjunction with KDD'08*.

[41] Roger Cavallo and Michael Pittarelli. The theory of probabilistic databases. In *VLDB*, pages 71–81, 1987.

[42] Artem Chebotko, Seunghan Chang, Shiyong Lu, Farshad Fotouhi, and Ping Yang. Scientific workflow provenance querying with security views. In *WAIM*, pages 349–356, 2008.

[43] James Cheney, Laura Chiticariu, and Wang Chiew Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[44] Wang chiew Tan. Containment of relational queries with annotation propagation. In *In Proceedings of the International Workshop on Database and Programming Languages (DBPL*, pages 37–53, 2003.

[45] L. Chiticariu, R. Krishnamurthy, Y. Li, F. Reiss, and S. Vaithyanathan. Domain Adaptation of Rule-based Annotators for Named-Entity Recognition Tasks. In *EMNLP*, 2010.

[46] Laura Chiticariu and Wang-Chiew Tan. Debugging schema mappings with routes. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 79–90. VLDB Endowment, 2006.

[47] D. G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discrete Appl. Math.*, 3(3):163–174, 1981.

[48] D. G. Corneil and Y. Perl. Clustering and domination in perfect graphs. *Discrete Applied Mathematics*, 9(1):27 – 39, 1984.

[49] D. G. Corneil, Y. Perl, and L. K. Stewart. A linear recognition algorithm for cographs. *SIAM J. Comput.*, 14(4):926–934, 1985.

[50] Yingwei Cui and Jennifer Widom. Run-time translation of view tuple deletions using data lineage. Technical Report 2001-24, Stanford InfoLab, 2001.

[51] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, June 2000.

[52] H. Cunningham. JAPE: a Java Annotation Patterns Engine. Research Memorandum CS – 99 – 06, University of Sheffield, May 1999.

[53] Francisco Curbera, Yurdaer Doganata, Axel Martens, Nirmal K. Mukhi, and Aleksander Slominski. Business provenance — a technology to increase traceability of end-to-end operations. In *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems:*, OTM '08, pages 100–119, Berlin, Heidelberg, 2008. Springer-Verlag.

[54] Nilesh N. Dalvi, Karl Schnaitter, and Dan Suciu. Computing query probability with incidence algebras. In *PODS*, pages 203–214, 2010.

[55] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[56] Nilesh N. Dalvi and Dan Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.

[57] Nilesh N. Dalvi and Dan Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, 2007.

[58] Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics*, 11(1-2):11–34, 2001.

[59] Adnan Darwiche. A compiler for deterministic, decomposable negation normal form. In *AAAI*, pages 627–634, 2002.

[60] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res. (JAIR)*, 17:229–264, 2002.

[61] Susan B. Davidson, Zhuowei Bao, and Sudeepa Roy. Hiding data and structure in workflow provenance. In *DNIS*, pages 41–48, 2011.

[62] Susan B. Davidson and Juliana Freire. Provenance and scientific workflows: challenges and opportunities. In *SIGMOD Conference*, pages 1345–1350, 2008.

[63] Susan B. Davidson, Sanjeev Khanna, Tova Milo, Debmalya Panigrahi, and Sudeepa Roy. Provenance views for module privacy. In *PODS*, pages 175–186, 2011.

[64] Susan B. Davidson, Sanjeev Khanna, Debmalya Panigrahi, and Sudeepa Roy. Preserving module privacy in workflow provenance. *CoRR*, abs/1005.5543, 2010.

[65] Susan B. Davidson, Sanjeev Khanna, Sudeepa Roy, Julia Stoyanovich, Val Tannen, Yi Chen, and Tova Milo. Enabling privacy in provenance-aware workflow systems. In *CIDR*, 2011.

[66] Susan B. Davidson, Tova Milo, and Sudeepa Roy. A propagation model for provenance views of public/private workflows. *Manuscript*, 2011.

[67] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[68] Daniel Deutch, Christoph Koch, and Tova Milo. On probabilistic fixpoint and markov chain query languages. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '10, pages 215–226, New York, NY, USA, 2010. ACM.

[69] Daniel Deutch and Tova Milo. A quest for beauty and wealth (or, business processes for database researchers). In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '11, pages 1–12, New York, NY, USA, 2011. ACM.

[70] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *PODS*, pages 202–210, 2003.

[71] Cynthia Dwork. Differential privacy: A survey of results. In *TAMC*, pages 1–19, 2008.

[72] Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2003.

[73] Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. Harvesting relational tables from lists on the web. *PVLDB*, pages 1078–1089, 2009.

[74] David Eppstein and Daniel S. Hirschberg. Choosing subsets with maximum weighted average. *J. Algorithms*, 24(1):177–193, 1997.

[75] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Methods for domain-independent information extraction from the web: an experimental comparison. In *AAAI*, 2004.

[76] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[77] Robert Fink, Dan Olteanu, and Swaroop Rath. Providing support for full relational algebra in probabilistic databases. In *ICDE*, pages 315–326, 2011.

[78] Juliana Freire, Cláudio T. Silva, Steven P. Callahan, Emanuele Santos, Carlos Eduardo Scheidegger, and Huy T. Vo. Managing rapidly-evolving scientific workflows. In *IPAW*, pages 10–18, 2006.

[79] Norbert Fuhr and Thomas Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[80] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[81] Floris Geerts, Anastasios Kementsietsidis, and Diego Milano. Mondrian: Annotating and querying databases through colors and blocks. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 82–, Washington, DC, USA, 2006. IEEE Computer Society.

[82] Johannes Gehrke, Edward Lui, and Rafael Pass. Towards privacy for social networks: A zero-knowledge based definition of privacy. In *TCC*, pages 432–449, 2011.

[83] Yolanda Gil, William K. Cheung, Varun Ratnakar, and Kai kin Chan. Privacy enforcement in data analysis workflows. In *PEAS*, 2007.

[84] Yolanda Gil and Christian Fritz. Reasoning about the appropriate use of private data through computational workflows. In *Intelligent Information Privacy Management*, pages 69–74, 2010.

[85] Martin Charles Golumbic, Aviad Mintz, and Udi Rotics. Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees. *Discrete Appl. Math.*, 154(10):1465–1477, 2006.

[86] Erich Grädel, Yuri Gurevich, and Colin Hirsch. The complexity of query reliability. In *PODS*, pages 227–234, 1998.

[87] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[88] Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Update exchange with mappings and provenance. In *VLDB*, pages 675–686, 2007.

[89] Alexander J. T. Gurney, Andreas Haeberlen, Wenchao Zhou, Micah Sherr, and Boon Thau Loo. Having your cake and eating it too: routing security with privacy protections. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets '11, pages 15:1–15:6, New York, NY, USA, 2011. ACM.

[90] V. A. Gurvich. Repetition-free Boolean functions. *Uspehi Mat. Nauk*, 32(1(193)):183–184, 1977.

[91] Vladimir A. Gurvich. Criteria for repetition-freeness of functions in the algebra of logic. *Soviet Math. Dokl.*, 43(3):721–726, 1991.

[92] Michel Habib and Christophe Paul. A simple linear time algorithm for cograph recognition. *Discrete Applied Mathematics*, 145(2):183 – 197, 2005.

[93] Andreas Haeberlen, Benjamin C. Pierce, and Arjun Narayan. Differential privacy under fire. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 33–33, Berkeley, CA, USA, 2011. USENIX Association.

[94] Ragib Hasan, Radu Sion, and Marianne Winslett. Introducing secure provenance: problems and challenges. In *StorageSS*, pages 13–18, 2007.

[95] John P. Hayes. The fanout structure of switching functions. *J. Assoc. Comput. Mach.*, 22(4):551–571, 1975.

[96] Thomas Heinis and Gustavo Alonso. Efficient lineage tracking for scientific workflows. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1007–1018, New York, NY, USA, 2008. ACM.

[97] L. Hellerstein and Marek Karpinski. Learning read-once formulas using membership queries. In *COLT*, pages 146–161, 1989.

[98] Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[99] Zachary Ives, Nitin Khandelwal, Aneesh Kapur, and Murat Cakir. Orchestra: Rapid, collaborative sharing of dynamic data. In *In CIDR*, 2005.

[100] Abhay Jha, Dan Olteanu, and Dan Suciu. Bridging the gap between intensional and extensional query evaluation in probabilistic databases. In *EDBT*, pages 323–334, 2010.

[101] Abhay Jha and Dan Suciu. Knowledge compilation meets database theory: compiling queries to decision diagrams. In *ICDT*, pages 162–173, 2011.

[102] D. Jurafsky and J.H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall series in artificial intelligence. Pearson Prentice Hall, 2009.

[103] Mauricio Karchmer, Nathan Linial, Ilan Newman, Michael E. Saks, and Avi Wigderson. Combinatorial characterization of read-once formulae. *Discrete Mathematics*, 114(1-3):275–282, 1993.

[104] Richard M. Karp and Michael Luby. Monte-carlo algorithms for enumeration and reliability problems. In *FOCS*, pages 56–64, 1983.

[105] Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Querying data provenance. In *SIGMOD*, pages 951–962, 2010.

[106] Sanjeev Khanna, Sudeepa Roy, and Val Tannen. Queries with difference on probabilistic databases. *PVLDB*, 4(11):1051–1062, 2011.

[107] Benny Kimelfeld. A dichotomy in the complexity of deletion propagation with functional dependencies. In *PODS*, pages 191–202, 2012.

[108] Benny Kimelfeld, Jan Vondrák, and Ryan Williams. Maximizing conjunctive views in deletion propagation. In *PODS*, pages 187–198, 2011.

[109] Christoph Koch. Approximating predicates and expressive queries on probabilistic databases. In *PODS*, pages 99–108, 2008.

[110] Christoph Koch. On query algebras for probabilistic databases. *SIGMOD Rec.*, 37:78–85, March 2009.

[111] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, Cambridge, MA, 2009.

[112] Aleksandra Korolova, Rajeev Motwani, Shubha U. Nabar, and Ying Xu. Link privacy in social networks. In *CIKM*, pages 289–298. ACM, 2008.

[113] Zornitsa Kozareva. Bootstrapping named entity recognition with automatically generated gazetteer lists. In *EACL: Student Research Workshop*, 2006.

[114] Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, Shivakumar Vaithyanathan, and Huaiyu Zhu. SystemT: a system for declarative information extraction. *SIGMOD Record*, 37(4):7–13, 2008.

[115] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, November 2006.

[116] Laks V. S. Lakshmanan, Nicola Leone, Robert B. Ross, and V. S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3), 1997.

[117] Jian Li, Barna Saha, and Amol Deshpande. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1):502–513, 2009.

[118] Bin Liu, Laura Chiticariu, Vivian Chu, H. V. Jagadish, and Frederick R. Reiss. Automatic Rule Refinement for Information Extraction. *PVLDB*, 2010.

[119] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

[120] John Lyle and Andrew Martin. Trusted computing and provenance: better together. In *TAPP*, page 1, 2010.

[121] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. ℓ-diversity: Privacy beyond k-anonymity. In *ICDE*, page 24, 2006.

[122] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *COMPUTATIONAL LINGUISTICS*, 19(2):313–330, 1993.

[123] D. Maynard, K. Bontcheva, and H. Cunningham. Towards a semantic extraction of named entities. In *Recent Advances in Natural Language Processing*, 2003.

[124] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45, 2010.

[125] Alexandra Meliou, Wolfgang Gatterbauer, Suman Nath, and Dan Suciu. Tracing data errors with view-conditioned causality. In *SIGMOD*, 2011.

[126] Andrei Mikheev, Marc Moens, and Claire Grover. Named entity recognition without gazetteers. In *EACL*, pages 1–8, 1999.

[127] Gerome Miklau and Dan Suciu. A formal analysis of information disclosure in data exchange. In *SIGMOD*, pages 575–586, 2004.

[128] Simon Miles, Sylvia C. Wong, Weijian Fang, Paul Groth, Klaus peter Zauner, and Luc Moreau. Provenance-based validation of e-science experiments. In *In ISWC*, pages 801–815. Springer-Verlag, 2005.

[129] Luc Moreau, Juliana Freire, Joe Futrelle, Robert E. McGrath, Jim Myers, and Patrick Paulson. The open provenance model: An overview. In *IPAW*, pages 323–326, 2008.

[130] Rajeev Motwani, Shubha U. Nabar, and Dilys Thomas. Auditing sql queries. In *ICDE*, pages 287–296, 2008.

[131] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. *ACM Comput. Surv.*, 28:33–37, March 1996.

[132] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. *ACM Comput. Surv.*, 28(1):33–37, 1996.

[133] Shubha U. Nabar, Bhaskara Marthi, Krishnaram Kenthapadi, Nina Mishra, and Rajeev Motwani. Towards robustness in query auditing. In *VLDB*, pages 151–162. VLDB Endowment, 2006.

[134] David Nadeau, Peter D. Turney, and Stan Matwin. Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity. In *Canadian Conference on AI*, pages 266–277, 2006.

[135] Qun Ni, Shouhuai Xu, Elisa Bertino, Ravi S. Sandhu, and Weili Han. An access control language for a general provenance model. In *Secure Data Management*, pages 68–88, 2009.

[136] T.M. 'Oinn *et al.* Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(1):3045–3054, 2003.

[137] Dan Olteanu and Jiewen Huang. Using obdds for efficient query evaluation on probabilistic databases. In *SUM*, pages 326–340, 2008.

[138] Dan Olteanu and Jiewen Huang. Secondary-storage confidence computation for conjunctive queries with inequalities. In *SIGMOD*, pages 389–402, 2009.

[139] Dan Olteanu, Jiewen Huang, and Christoph Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, pages 640–651, 2009.

[140] Dan Olteanu, Jiewen Huang, and Christoph Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, pages 145–156, 2010.

[141] J. Peer and R Pinter. Minimal decomposition of boolean functions using non-repeating literal trees. In *IFIP Workshop on Logic and Architecture Synthesis*, 1995.

[142] Vibhor Rastogi, Michael Hay, Gerome Miklau, and Dan Suciu. Relationship privacy: output perturbation for queries with joins. In *PODS*, pages 107–116, 2009.

[143] Ran Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27:763–803, 1998.

[144] Christopher Re, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.

[145] Christopher Ré and Dan Suciu. Efficient evaluation of having queries on a probabilistic database. In *DBPL*, pages 186–200, 2007.

[146] Christopher Ré and Dan Suciu. The trichotomy of having queries on a probabilistic database. *VLDB J.*, 18(5):1091–1116, 2009.

[147] Jason Reed, Adam J. Aviv, Daniel Wagner, Andreas Haeberlen, Benjamin C. Pierce, and Jonathan M. Smith. Differential privacy for collaborative security. In *Proceedings of the Third European Workshop on System Security*, EUROSEC '10, pages 1–7, New York, NY, USA, 2010. ACM.

[148] Frederick Reiss, Sriram Raghavan, Rajasekar Krishnamurthy, Huaiyu Zhu, and Shivakumar Vaithyanathan. An algebraic approach to rule-based information extraction. In *ICDE*, pages 933–942, 2008.

[149] Theodoros Rekatsinas, Amol Deshpande, and Lise Getoor. Local structure and determinism in probabilistic databases. In *Proceedings of the 2012 international conference on Management of Data*, SIGMOD '12, pages 373–384, New York, NY, USA, 2012. ACM.

[150] E. Riloff. Automatically constructing a dictionary for information extraction tasks. In *KDD*, 1993.

[151] Sudeepa Roy, Laura Chiticariu, Vitaly Feldman, Frederick R. Reiss, and Huaiyu Zhu. Optimization problems for dictionary refinement in information extraction *(Manuscript.)*. *http://www.cis.upenn.edu/∼sudeepa/dictionary.pdf*.

[152] Sudeepa Roy, Vittorio Perduca, and Val Tannen. Faster query answering in probabilistic databases using read-once functions. In *ICDT*, pages 232–243, 2011.

[153] Szabolcs Rozsnyai, Aleksander Slominski, and Yurdaer Doganata. Large-scale distributed storage system for business provenance. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, CLOUD '11, pages 516–524, Washington, DC, USA, 2011. IEEE Computer Society.

[154] Prithviraj Sen and Amol Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, pages 596–605, 2007.

[155] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Exploiting shared correlations in probabilistic databases. *PVLDB*, 1(1):809–820, 2008.

[156] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Read-once functions and query evaluation in probabilistic databases. *PVLDB*, 3(1):1068–1079, 2010.

[157] Warren Shen, AnHai Doan, Jeffrey F. Naughton, and Raghu Ramakrishnan. Declarative information extraction using datalog with embedded extraction predicates. In *VLDB*, pages 1033–1044, 2007.

[158] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, September 2005.

[159] Mohamed A. Soliman, Ihab F. Ilyas, and Shalev Ben-David. Supporting ranking queries on uncertain and incomplete data. *VLDB J.*, 19(4):477–501, 2010.

[160] Joshua Spiegel and Neoklis Polyzotis. Graph-based synopses for relational selectivity estimation. In *SIGMOD Conference*, pages 205–216, 2006.

[161] Dan Suciu, Dan Olteanu, R. Christopher, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool Publishers, 1st edition, 2011.

[162] Sudeepa Roy Susan Davidson, Sanjeev Khanna and Sarah Cohen Boulakia. Privacy issues in scientific workflow provenance. In *WANDS'10: 1st International Workshop on Workflow Approaches to New Data-centric Science*, 2010.

[163] Latanya Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.

[164] Victor Tan, Paul T. Groth, Simon Miles, Sheng Jiang, Steve Munroe, Sofia Tsasakou, and Luc Moreau. Security issues in a SOA-based provenance system. In *IPAW*, pages 203–211, 2006.

[165] Val Tannen. Provenance for database transformations. In *EDBT*, page 1, 2010.

[166] Nicholas E. Taylor and Zachary G. Ives. Reconciling while tolerating disagreement in collaborative data sharing. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 13–24, New York, NY, USA, 2006. ACM.

[167] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: language-independent named entity recognition. In *HLT-NAACL*, 2003.

[168] Salil P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM J. Comput.*, 31(2):398–427, 2001.

[169] L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8(2):189 – 201, 1979.

[170] Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

[171] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[172] Vassilios S. Verykios, Elisa Bertino, Igor Nai Fovino, Loredana Parasiliti Provenza, Yucel Saygin, and Yannis Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Rec.*, 33(1):50–57, 2004.

[173] Y. Richard Wang and Stuart E. Madnick. A polygon model for heterogeneous database systems: the source tagging perspective. In *Proceedings of the sixteenth international conference on Very large databases*, pages 519–533, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[174] Jennifer Widom. Trio: a system for integrated management of data, accuracy, and lineage," presented at cidr 2005, 2005.

[175] Allison Woodruff and Michael Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *Proceedings of the Thirteenth International Conference on Data Engineering*, ICDE '97, pages 91–102, Washington, DC, USA, 1997. IEEE Computer Society.

[176] Mingji Xia and Wenbo Zhao. #3-regular bipartite planar vertex cover is #p-complete. In *TAMC*, pages 356–364, 2006.

[177] A. Yates, M. Banko, M.Broadhead, M. J. Cafarella, O. Etzioni, and S. Soderland. TextRunner: Open Information Extraction on the Web. In *HLT-NAACL (Demonstration)*, pages 25–26, 2007.

[178] Esteban Zimányi. Query evaluation in probabilistic relational databases. *Theor. Comput. Sci.*, 171(1-2):179–219, 1997.