Integrating Web and Database Searches

CPS 296.1
Topics in Database Systems

Roadmap

- Rank aggregation: merging ranked results from different searches
- Proximity search: finding all shortest paths in the link structure of a database
- WSQ: enhancing database queries with Web searches

Rank aggregation problem

- Each object $R$ is graded using $m$ different criteria
  - Grades are $x_1, x_2, \ldots, x_m$
  - Example: each grade measures the relevance of $R$ for a particular search term
- The combined grade is calculated by an aggregation function $t(x_1, x_2, \ldots, x_m)$
  - Example: weighted sum, min, max, etc.
- Problem: find the top $k$ objects with the highest combined grade

Modes of access

- Sorted: sequentially request a list of objects (with their grades) ranked according to one criterion (from highest to lowest)
  - Conceptually, there are $m$ lists $L_1, L_2, \ldots, L_m$, one for each criterion
  - Example: search an image database for a list of “red” pictures, ranked by their “redness”
- Random: request the grade of an object according to one criterion
  - Example: query an image database for the “redness” of one particular picture

Assumptions

- Individual scores are in the interval $[0, 1]$
- Aggregation function is monotone
  - If $x_i \leq x'_i$ for every $i$, then $t(x_1, x_2, \ldots, x_m) \leq t(x'_1, x'_2, \ldots, x'_m)$
  - Examples: min, sum
  - Not an example: $1 - \text{sum}(x_1, x_2, \ldots, x_m)$
- Cost per sorted access (one object, one score): $c_S$
- Cost per random access (one object, one score): $c_R$

Naïve algorithm

- For each criterion, do sorted access to retrieve all objects and their grades
  - That is, access all $L_i$’s
- Calculate the combined grades for all objects
- Pick the top $k$
  - Accesses the entire database
  - Does not use the fact that $L_i$’s are sorted
**FA (Fagin’s Algorithm)**
- Do sorted access in parallel to all $L_i$'s, until there are $k$ “matches”
  - A “match” is an object that has been seen in all $L_i$’s
- For each object that has been seen, do random accesses to get all its grades
- Calculate the combined grades and pick the top $k$
- Pop quiz
  - Why not just consider the $k$ matches?
  - Why not consider objects seen after the $k$ matches?
- Needs to remember lots of objects: large buffer size
- Does not use the aggregation function effectively

**TA (Threshold Algorithm)**
- Do sorted access in parallel to all $L_i$’s
  - Whenever an object is seen, do random accesses to get all its grades, and compute the combined grade
  - Remember up to $k$ objects with top combined grades
  - Calculate the threshold value $\tau = \left( x_1, x_2, \ldots, x_m \right)$, where $x_i$’s are the bottom grades seen so far
  - If we have seen at least $k$ objects whose combined grade is at least $\tau$, stop
- Output the top $k$ objects we remembered

**Intuition behind TA**
- $\tau$ serves as an upper bound on the combined grade for objects that have never been seen
- When we stop, the top $k$ objects we have remembered all have combined grade of at least $\tau$
  - The top $k$ we have are the top $k$ overall
- “Gather what information you need to allow you to know the top $k$ answers, and then halt”

**FA versus TA**
- TA never stops later than FA
  - FA’s stopping condition ($k$ matches) implies that TA’s stopping condition ($k$ objects above threshold) has already been satisfied
- TA requires only bounded buffers (to remember the top $k$ objects)
- TA may perform more random accesses

**Instance optimality**
- $A$: a class of algorithms
- $D$: a class of legal inputs to algorithms
- $\text{cost}(A, D)$: cost of running algorithm $A$ with input $D$
- An algorithm $B \in A$ is instance optimal over $A$ and $D$ if for every $A \in A$ and every $D \in D$,
  $\text{cost}(B, D) = \Omega(\text{cost}(A, D))$
  - That is, $\text{cost}(B, D) \leq c \cdot \text{cost}(A, D) + c'$, where $c$ is called the optimality ratio
- Much stronger than worst-case optimality

**Instance optimality of TA**
- $A$: the class of all algorithms that do not make mistakes or wild guesses
  - “No wild guesses” means no random access for $R$ unless $R$ has been seen in sorted access
- $D$: the class of all possible inputs
- TA is instance optimal over $A$ and $D$, with optimality ratio of at most $m + m(m-1) c_R / c_S$
Intuition behind TA’s instance optimality
• Say an algorithm $A$ stops sooner than TA on some input (i.e., one of the top $k$ picked by $A$ has combined grade less than $\tau$)
• Construct another input by inserting a new object $R$ right below where $A$ stops looking
• Then $A$ will make a mistake by failing to pick $R$
➢ See paper for a rigorous proof and other results

Related work: Quick-Combine
• Instead of doing sorted access on all $L_i$’s in parallel, choose one $L_i$ to access next
• Prefer the list in which grades are declining at the fastest rate, so we can lower the threshold value faster
  – Rate is measured by the decrease in successive grades, weighted by $|\frac{\partial t}{\partial x_i}|$
  ➢ $|\frac{\partial t}{\partial x_i}|$ may be undefined (e.g., min)
➢ Quick-Combine may beat TA in some cases, but is not instance optimal unless we ensure every $L_i$ is accessed every now and then

Extending TA
• What if we only need approximate answers? $\text{TA}_\theta$
  – Example: Web search, with lots of good-quality answers
• What if we have no sorted access for some criteria? $\text{TA}_Z$
  – Example: find good restaurants near me (sorted and random accesses for restaurant ratings, random access only for distances)
• What if we have no random access at all? NRA
  – Example: Web search engines, which typically do not allow you to enter a URL and get its ranking
• What if consider the relative costs of random and sorted accesses? CA

Adding approximation: $\text{TA}_\theta$
• A $\theta$-approximation ($\theta > 1$) to the top $k$ answers is a collection of $k$ objects, such that
  – For each $R$ among these $k$ objects, and for each $R'$ not among these these $k$ objects, $\theta t(R) \geq t(R')$
• $\text{TA}_\theta$: same as TA, except that the stopping condition is “we have seen at least $k$ objects whose combined grade is at least $\tau / \theta$”

Restricting sorted access: $\text{TA}_Z$
• Suppose we only have sorted access to $L_i$ for $i \in Z$
• $\text{TA}_Z$: same as TA, except
  – Only do sorted accesses to $L_i$’s where $i \in Z$
  – Use $x_i = 1$ to calculate the threshold if sorted access to $L_j$ is not allowed ($j \notin Z$)
➢ Intuition: The first object that we see in $L_j$ (if we could) can have grade 1

No random access: NRA
• Key observation:
  Sometimes we do not have to know all individual grades of $R$ in order to tell that $R$ is among the top $k$
• Use bounds!

We can now tell $R_1$ and $R_2$ are the top 2
NRA

• Do sorted access in parallel to all $L_i$’s; at each depth:
  – Maintain bottom grades $L_1, L_2, \ldots, L_m$ seen in the lists
  – For each object $R$, calculate the lower and upper bounds for its combined grade
    • If $x_i$ is not available, use 0 in calculating lower bound, and use $x_i$ in calculating upper bound
  – Maintain a current top $k$ list containing the $k$ objects with the largest lower bounds (ties are broken by upper bounds)

• Stopping condition
  – For every $R$ not in the current top $k$ list, the calculated upper bound for $R$ is less than the calculated lower bound for all objects in the current top $k$ list

➢ Lots of note-keeping!

Combined algorithm: CA

• A compromise between TA (lots of random accesses) and NRA (no random accesses)
  – Uses random accesses, but considers their cost relative to sorted access

• Suppose $h = \lceil c_R / c_S \rceil \geq 1$
• CA: same as NRA, except
  – Every $h$ steps, pick an object with missing individual grades (see paper for details on which object to pick)
  – Do random accesses to get the missing grades, and then recalculate the bounds

Summary

• A very practical problem (rank aggregation)
• A simple algorithm (TA) and various extensions
  – Nothing wild; just clean implementations of remarkably simple ideas (thresholds, bounds)
  – But the amazing thing is that these simple algorithms are unbeatable for any input (instant optimality)!