Symbolic and Numerical Computation for Artificial Intelligence

edited by

Bruce Randall Donald
Department of Computer Science
Cornell University, USA

Deepak Kapur
Department of Computer Science
State University of New York, USA

Joseph L. Mundy
AI Laboratory
GE Corporate R&D, Schenectady, USA

Academic Press
Harcourt Brace Jovanovich, Publishers
London San Diego New York
Boston Sydney Tokyo Toronto

ACADEMIC PRESS LIMITED
24–28 Oval Road
London NW1

US edition published by
ACADEMIC PRESS INC.
San Diego, CA 92101

This book is printed on acid-free paper

A catalogue record for this book is available from the British Library

ISBN 0-12-220535-9

Printed and Bound in Great Britain by
The University Press, Cambridge

Chapter 4

Quantifier Elimination for Conjunctions of Linear Constraints via a Convex Hull Algorithm

Catherine Lassez

Jean-Louis Lassez

IBM T.J. Watson Research Center

P.O.Box 704, Yorktown Heights, NY 10598

We propose a new algorithm for quantifier (variable) elimination for systems of linear constraints using a generalized linear program formulation and an on-line convex hull construction. The algorithm provides an exact solution when the dimension of the output is small and an approximation in the general case when the size of the output is unmanageable. Apart from trivial cases, existing algorithms fail in both respect.

1. Introduction

Tarski (see Van De Vries, 1988) proved that most problems in elementary algebra and Euclidean geometry can be solved by a *single* algorithm: quantifier elimination. The enormous potential for applications to robotics, graphics, constructive solid geometry and to new constraint-based languages, such as CLP(R), PROLOG III, CHIP and systems like MATHEMATICA, has not, however, been realized. Few of the existing algorithms have been implemented. They are of great theoretical interest but, apart from theorem proving (Arnon, 1989), do not have yet real practical significance. A main reason is that the size of the output may be doubly exponential and even simple problems cannot yet be solved (Davenport, 1988; Davenport and Heintz, 1988). It has been remarked (Schwartz and Sharir, 1989; Van De Vries, 1988) that really practical solutions will come only by concentrating on special cases.

We address this problem in the restricted case of conjunctions of linear arithmetic constraints. This restriction still allows us to solve, in principle, a wide range of interesting problems (Huynh *et al.*, 1991; Huynh *et al.*, 1990; Helm *et al.*, 1991) and is an important case in the Constraint Logic Programming languages CLP(R), PROLOG III, CHIP and Constraint Query Languages (Kanellakis *et al.*, 1990) where auxiliary variables introduced during execution of a program have to be eliminated.

At the theoretical level we still have, despite the simplicity of the constraints, that the

size of the output may be exponential and we face, in practice, very serious problems of redundancy and degeneracy. Existing algorithms, in general, fail to produce any output (not even partial information) because of the size or the amount of intermediate computations. As a consequence, existing implementations use simple heuristics that work only for trivial input or when very few variables are to be eliminated (Duffin, 1974; Huynh and Lassez, 1990).

Many of the shortcomings of the existing algorithms are due to the fact that they perform algebraic manipulations and are based on the syntax of the constraints rather than on their semantics. However, it is known that quantifier elimination can be viewed geometrically as an operation of projection. The algorithm we propose exploits this observation systematically. A main feature of our approach is to transform the unbounded case so that it reduces to the conceptually simple bounded case (Taylor and Rajan, 1988). Then, the projection is computed by successive approximations using linear programming techniques to find extreme points in the projection space. At each step, the convex hull of these points provides the approximation. The complexity is essentially based on the dimension of the output. When the size of the projection is too large, the algorithm still provides an approximation which can be an upper or lower bound or both and whose size is user-defined.

Our main concern is to provide an algorithm which is as practical as possible. Consequently, we do not make any assumptions concerning redundancy, full dimensionality, degeneracy, general position, etc. Some of these problems are addressed by the nature of the approach, others by substantial preprocessing. In particular, we emphasize shortcuts that may be found when the output is simple, regardless of the complexity of the input. Even though these may appear as side issues as compared to the main ideas behind the algorithm, they are of considerable importance for a feasible implementation and therefore we will spend some time addressing them.

The rest of the paper is organized as follows. In the next section, we briefly discuss variable elimination as a general technique for symbolic computation problems and review some preliminary results. In section 3, we address the problem of unbounded input and discuss the transformation that leads to the bounded case. Section 4 contains an informal description of the main steps of the algorithm. In section 5, we present the general outline of the algorithm. In section 6, we discuss in more details the major parts of the algorithm as well as some possible variations and optimizations. In section 7, we present some initial empirical results to illustrate the feasibility of this approach. Finally we conclude with some remarks on implementation issues as well as future research directions.

2. Preliminaries

In this section we review briefly the basic tools from quantifier (variable) elimination that we need to motivate this study. The material is taken from Huynh *et al.* (1990) and Lassez (1990) which should be consulted for full details. We also explain the simple extensions or applications that are needed for the understanding of the algorithm we propose.

2.1. Variable elimination as a tool in Symbolic Computation

Fourier's procedure has been used in Linear Programming to prove many fundamental theorems. However, it is rarely (if at all) mentioned in the area of symbolic computation. Indeed the operation of Fourier's procedure corresponds to quantifier elimination in symbolic computation which is a very powerful tool. In geometric terms, eliminating a variable from $S = \{Ax \leq b\}$ (A is a m by n real matrix, b an m-real vector and x an n-vector) is equivalent to projecting the polyhedron represented by S on a subspace with one dimension less. We give here a few examples of the power of variable elimination to solve problems automatically from their specification (more will be found in the references). A typical problem in geometry is to find the convex hull of a given set of points. A point of coordinates $(x_1, \ldots, x_n)$ is in the convex hull of a set of points $\{(\alpha_{1,j}, \ldots, \alpha_{n,j})\}$ if and only if there exists λ_j's such that the system

$$\{x_1 = \Sigma\alpha_{1,j}\lambda_j, \ \ldots, \ x_n = \Sigma\alpha_{n,j}\lambda_j, \ \Sigma\lambda_j = 1, \ \lambda_j \geq 0\}$$

is satisfied. The representation of the convex hull is an equivalent set of relations solely between the x_i's. It is obtained by eliminating all λ_j's in the system.

Another interesting application is to compute the image of a polyhedral set mapped by a linear function. Suppose the function is given as

$$f(x_1, \ldots, x_n) = \begin{cases} X_1 = \alpha_{1,1}x_1 + \ \ldots \ + \alpha_{1,n}x_n + \beta_1 \\ \vdots \\ X_m = \alpha_{m,1}x_1 + \ \ldots \ + \alpha_{m,n}x_n + \beta_m \end{cases}$$

Let $S = \{Ax \leq b\}$ represent a polyhedral set. We want the image of S by f (or more formally by f's extension to the space of polyhedral sets). To obtain the image of S one simply eliminates all x_i's from the polyhedral set

$$\{X_1 = \alpha_{1,1}x_1 + \ \ldots \ + \alpha_{1,n}x_n + \beta_1, \ \ldots, \ X_m = \alpha_{m,1}x_1 + \ \ldots \ + \alpha_{m,n}x_n + \beta_m, \ Ax \leq b\}.$$

The result is a set of constraints in the space of $\{X_1, \ldots, X_m\}$ defining the image. For example, if

$$f(x, y) = \begin{cases} X = 2x + y \\ Y = x - y + 3 \\ Z = 3x + 2y - 5 \end{cases}$$

then the image of the triangle defined by

$$\{x + y \leq 1, \ x \geq 0, \ y \geq 0\}$$

is

$$\{10X - 2Y - 6Z = 24, \ -2Y + 4Z \leq -16, \ -2Y - Z \leq -1, \ 6Y - 2Z \leq 28\}.$$

The examples given above illustrate the fact that variable elimination provides a systematic way of characterizing interesting sets of constraints directly from a simple existential specification. Now we show how variable elimination can also deal with more complex problems.

Given a set of constraints S, we want to extract information from it in the form of

parametric queries:

$$\exists \alpha_1, \ldots, \alpha_n, \beta \, \forall x_1, \ldots, x_n : S \Rightarrow (\alpha_1 x_1 + \ldots + \alpha_n x_n \leq \beta) \wedge R(\alpha_1, \ldots, \alpha_n, \beta) \; ?$$

where $R(\alpha_1, \ldots, \alpha_n, \beta)$ represents a set of linear relations on the parameters such as, for instance, $\alpha_1 \geq 3\beta$. The answer to that query is a set of relations solely between the α's and β. This can be obtained by eliminating all the other variables. But we cannot immediately apply the preceding technique. We have an existential specification which now involves an implication and non-linearity. However, we shall show that the previous mechanism of variable elimination is still applicable for this class of problems.

We define a constraint C to be a quasi-linear combination of constraints of $S = \{Ax \leq b\}$ if and only if C is obtained by adding a positive number to the right hand side of a non-negative linear combination of constraints of S. From the Subsumption Theorem we know that a constraint C is implied by a set of constraints S if and only if C is a quasi-linear combination of constraints of S. This provides a basis to formulate the framework for solving the parametric linear query problem. Namely, a constraint

$$\alpha_1 x_1 + \ldots + \alpha_n x_n \leq \beta$$

is implied by S if and only if the system

$$\{\alpha_j = A_{.j}{}^T \lambda, \; \beta = b^T \lambda + q, \; \lambda_i \geq 0, \; q \geq 0\}$$

is solvable (where $A_{.j}$ denotes the j column of the matrix A).

Eliminating the λ_i's and q from the above system, we obtain a set of relations on the α_j's and β which is satisfied if and only if the query is satisfied. This set forms the answer to the query. To demonstrate the usefulness of parametric queries let us look at the following application: how to characterize all hyperplanes that separate two disjoint polyhedral sets.

Let S_1 and S_2 be the given disjoints polyhedral sets. The problem can be formulated with the following parametric queries:

$$\exists \alpha_1, \ldots, \alpha_n, \beta \, \forall x_1, \ldots, x_n : S_1 \Rightarrow \alpha_1 x_1 + \ldots + \alpha_n x_n \leq \beta \; ?$$

and

$$\exists \alpha_1, \ldots, \alpha_n, \beta \, \forall x_1, \ldots, x_n : S_2 \Rightarrow -\alpha_1 x_1 - \ldots - \alpha_n x_n \leq -\beta \; ?$$

The intersection of the polyhedral sets that are the answers to these two queries characterizes the separating hyperplanes. For example, given

$$S_1 = \{x \leq 5, \; -x \leq -1, \; y \leq 3, \; -y \leq -1\}$$

and

$$S_2 = \{x \leq 10, \; -x \leq -8, \; y \leq 6, \; -y \leq -2\}.$$

The answers to the two queries are obtained by eliminating the λ_i's and q from

$$\{\alpha_1 = \lambda_1 - \lambda_2, \; \alpha_2 = \lambda_3 - \lambda_4, \; \beta = 5\lambda_1 - \lambda_2 + 3\lambda_3 - \lambda_4 + q, \; \lambda_i \geq 0, \; q \geq 0\}$$

and the μ_i's and p from

$$\{-\alpha_1 = \mu_1 - \mu_2, \; -\alpha_2 = \mu_3 - \mu_4, \; -\beta = 10\mu_1 - 8\mu_2 + 6\mu_3 - 2\mu_4 + p, \; \mu_i \geq 0, \; p \geq 0\}$$

The intersection of the two answer sets is:

$$H = \begin{cases} \alpha_1 + \alpha_2 - \beta - \leq 0 \\ 5\alpha_1 + \alpha_2 - \beta - \leq 0 \\ \alpha_1 + 3\alpha_2 - \beta - \leq 0 \\ 5\alpha_1 + 3\alpha_2 - \beta \leq 0 \\ -8\alpha_1 - 2\alpha_2 + \beta \leq 0 \\ -10\alpha_1 - 2\alpha_2 + \beta \leq 0 \\ -8\alpha_1 - 6\alpha_2 + \beta \leq 0 \\ -10\alpha_1 - 6\alpha_2 + \beta \leq 0 \end{cases}$$

An hyperplane $\alpha_1 x + \alpha_2 y = \beta$ separates S_1 and S_2 if and only if α_1, α_2 and β satisfy H.

2.2. Generalized Linear Program

A method for eliminating variables, called the extreme points method, was derived from the formalism of parametric queries. This method is interesting as it shows that variable elimination can be viewed as a straightforward generalization of a linear program in its specification and as a generalization of the simplex in its execution.

Let $S = Ax \leq b$ and let $V = \{x_{k+1} \ldots x_m\}$ be the set of variables to be eliminated, the associated generalized linear program is then defined as:

Definition 2.1. *Generalized Linear Program (GLP)*

$$extr(\Phi(\Delta))$$

$$\Phi = \begin{cases} \sum \lambda_i a_{i_1} = \alpha_1 \\ \vdots \\ \sum \lambda_i a_{i_k} = \alpha_k \\ \sum \lambda_i b_i = \beta \end{cases}$$

$$\Delta = \begin{cases} \sum \lambda_i a_{i_{k+1}} = 0 \\ \vdots \\ \sum \lambda_i a_{i_m} = 0 \\ \sum \lambda_i = 1 \\ \lambda_i \geq 0 \end{cases}$$

where *extr* denotes the set of extreme points. Each equation in Δ represents the linear combination of the constraints of S that eliminate a variable of V. The normalization of the λ's ensures that Δ is a polytope.

Theorem 2.1. *$extr(\Phi(\Delta))$, solutions of GLP, determine a finite set of constraints which defines the projection of S.*

This means that the coordinates of each extreme point of $\Phi(\Delta)$ are the coefficients of one of the constraints that define the projection. The objective function in the usual linear program can be viewed as a mapping from R^n to R, the image of the polyhedron defined by the constraints being an interval in R. The optimization consists in finding a

maximum or a minimum, that is one of the extreme points of the interval. In a GLP, the objective function represents a mapping from R^n to R^m and instead of looking for one extreme point, we look for the set of all extreme points. At the operational level, we can execute this GLP by generalizing the simplex method. The extreme points of $\Phi(\Delta)$ are images of extreme points of Δ. So we compute the set of extreme points of Δ, map them by Φ and eliminate the images which are not extreme points. It is important to note that although the extreme points method is better that Fourier in general because it eliminates the costly intermediate steps, there are still two main problems: the computation of the extreme points of Δ can be extremely costly even when the size of the projection is small and also the method produces a highly redundant output. See Huynh and Lassez (1990), for a comparison between this method and Fourier's and the last section for empirical results.

3. Handling the unbounded case

In the case when the projection is bounded, there exists a conceptually simple method for computing it based on the fact that a polytope is completely determined by its extreme points. A natural generalization of this method to the unbounded case leads to a far more complex algorithm (see Golan, 1991). This is due to the fact that, in general, an unbounded polyhedral set cannot be fully described as the convex hull of its low-dimension faces such as extreme points and extreme rays whose projections are easy to compute. This is the prevalent case in the CLP situation where the output typically has more variables than constraints, and consequently, there are no extreme points and no extreme directions.

Therefore we propose here a different approach based on what has been presented in the previous section. Its aim is to transform via duality the unbounded case into a bounded one in order to use a simple projection method.

The algorithm we present in the next section uses a convex hull construction which assumes that the projection is bounded. An important feature of this algorithm is that, even though its aim is to produce the constraints that define the projection, it does so by also computing the set of extreme points of the projection. When the projection is not bounded, the problem is reformulated as a generalized linear program. In this case, instead of computing the extreme points of $\Phi(\Delta)$, we give as input to the algorithm the set of constraints that define Φ and Δ, requesting the computation by *projection* of $\Phi(\Delta)$, as we saw in the first subsection. This is possible as the input is now bounded. The output will contain the convex hull of $\Phi(\Delta)$ but also the set of its extreme points, from which we can derive the constraints which define the projection we request. The advantage over the previous method is that we compute directly the extreme points of the projection. We do not need to compute the extreme points of Δ, this being the source of enormous intermediate computation and high redundancy in the output. As we will see later, this passage to a generalized linear program formulation can also be done if the input is bounded to obtain an alternative approximation when the size of the output is unmanageable.

The extreme point method as applied to the generalized linear program has the inconvenience of going through the computation of the extreme points of Δ. But there is one

case where it is very advantageous. When the extreme point method does not generate any extreme points, the generalized linear program has no solutions. It implies that the output is trivial, that is, the projection is the whole subspace. This can be easily tested as it corresponds to a single run of phase I of the simplex. We will use this fact in the next section to optimize parts of the algorithm.

Even though minor, another interesting application of the parametric queries is the derivation of a simple test for boundedness of the polyhedral set. It requires only one run of the phase I of the simplex (albeit on a possibly larger set). We remark that a polyhedral set P associated with a set of constraints S is bounded if and only if, given any hyperplane H, it is always possible to translate H so that all points of P are on the same side of H. This means that for every set of coefficients of the variables in the equation that defines H one can always find an appropriate constant. Consider now the parametric query:

$$\exists \alpha_1, ..., \alpha_n \; \exists \beta \; \forall x_1, ..., x_n : S \Rightarrow \sum_i \alpha_i x_i \leq \beta \; ?$$

Its answer obtained by eliminating the x_i's will give us a characterization of all the constraints implied by S. If we go one step further and also eliminate β, we will find the relations that the coefficients of the variables must satisfy so that there exists a constant such that the query is satisfied. These relations will form the empty set if and only if P is bounded. Consequently, if we start running the extreme point method over the appropriate formulation, if it fails, P is bounded, and if a first extreme point is produced, P is unbounded. This can be accomplished with one run of phase I of the simplex.

The "projection proper" part of the algorithm will take as input a set of constraints whose associated polyhedral set is not empty and is full-dimensional. The tests for both solvability and full-dimensionality can be done with a single linear program, called the *quasi-dual* and described below. In fact this single linear program will also tell us in some cases that the polyhedral set is not bounded. It is derived from the formulation of parametric queries in a very natural way, and is a variant of the dual of S considered as a linear program with objective function set to zero. It is of interest because of its very strong relation with Fourier's algorithm, and its many properties described in more details in the references:

$$\text{minimize } \Phi(\Delta)$$

$$\Phi = \sum_i \lambda_i b_i$$

$$\Delta = \begin{cases} \sum_i \lambda_i a_{ij} = 0 \; \forall j \\ \sum_i \lambda_i = 1 \\ \lambda_i \geq 0 \; \forall i. \end{cases}$$

If Δ is not solvable then S is solvable and its associated polyhedral set is unbounded. Now assume that Δ is solvable. If the minimum is strictly positive then S is solvable and full-dimensional. If the minimum of $\Phi(\Delta)$ is 0 then S is solvable and not full-dimensional. Furthermore one can derive the set of implicit equalities that are the cause of reduced dimension. (This information is necessary in order to simplify the system). If during minimization a negative value is reached then S is unsolvable.

4. Informal description of the algorithm

The input is an arbitrary polyhedral set: bounded or not, full-dimensional or not, redundant or not, empty or not. In a first step, the solvability of the input set is tested. If solvable but not full-dimensional, it is simplified by standard linear programming techniques into a set of equations that defines its affine hull and a set of inequalities which defines a full-dimensional polyhedral set in a smaller space. A straightforward variable elimination in the set of equations gives the affine hull of the projection. This will be part of the final output. This simplification allows us to eliminate as many variables as possible by using only linear programming and Gaussian elimination before getting into the costly convex hull construction. The details of this simplification are only marginally relevant to the main point of the paper, so we refer the reader to Lassez and McAloon (1989) for further information.

Now we consider two cases depending on whether the polyhedral set defined by the input set of constraints is bounded or not.

In the bounded case, the algorithm works directly on the input constraints. It is a variant and refinement of a method proposed in Taylor and Rajan (1988). The projection is computed by successive refinements of an initial approximation. The initial approximation is obtained by computing enough extreme points of the projection so that their convex hull is full-dimensional. The points are obtained by running linear programs on the initial set of constraints. Next, successive refinements consist in adding new extreme points and updating the convex hull: if a constraint in the convex hull does not belong to the projection then a new extreme point is determined and the convex hull is updated. The costly convex hull construction is done in the projection space, thus the main complexity of the algorithm is linked to the dimension of the output. The process stops when either the projection has been found or the size of the approximation has reached a user-supplied bound.

In the unbounded case, the problem is reformulated using the generalized linear program representation which is bounded by definition. The algorithm proceeds then as in the bounded case. This leads to a dual interpretation of the results as it is the set of extreme points and not their convex hull that now represents the constraints of the projection. Consequently the algorithm incrementally generates constraints whose associated polyhedral set contains the projection as opposed to the bounded case where it is a subpolytope of the projection which is incrementally generated.

In summary, in the bounded case we obtain a lower bound approximation and in the unbounded case an upper bound approximation. Note that in the bounded case both approximations can be computed if desired and the number of constraints which belong to the projection gives some measure of the accuracy of the approximation.

5. General outline

The algorithm is in three parts. Part 1 deals with the testing for solvability, full-dimensionality and boundedness. Part 2 computes an initial set of extreme points whose convex hull is a full-dimensional approximation of the projection. Part 3 incrementally refines the initial approximation by adding new extreme points and updating the convex

hull until the projection is fully computed or the size of the current approximation exceeds the input bound.

5.1. Test for solvability, full-dimensionality and boundedness

1. Input = set of constraints S
2. Test the solvability and full-dimensionality of S with the quasi-dual

- **2.1.** If S is not solvable then STOP else
- **2.2.** If S contains implicit equalities then
  - **2.2.1.** Compute all implicit equalities
  - **2.2.2.** Simplify to S={Eq ∪ Ineq} where Eq=set of equations and Ineq=set of inequalities defining a full-dimensional polyhedral set

3. If Ineq does not contain any of the variables to be eliminated then output Ineq ∪ projection(Eq) and STOP
4. Test the boundedness of the projection of Ineq

- **4.1.** If unbounded then
  - **4.1.1.** Formulate the GLP Q of Ineq
  - **4.1.2.** If Q is not solvable then output projection(Eq) and STOP
  - **4.1.3.** else if Q is not full-dimensional then
    - **4.1.3.1.** Compute implicit equalities
    - **4.1.3.2.** Simplify to Q={Eq' ∪ Ineq'} where Eq'=set of equations and Ineq'=set of inequalities defining a full-dimensional polytope

In the bounded case, the output of this first phase is a solvable set of constraints consisting of a set of inequalities in standardized form, which defines a full-dimensional polytope, and a set of equalities defining the affine hull of S.

In the unbounded case, the output is a set of equations defining the affine hull of S plus a set of inequalities in the dual space that defines a full-dimensional polytope plus a set of equalities defining the affine hull of Q.

The input to the next phase of the algorithm is the bounded set of inequalities.

Note: the procedures *extreme_point, initial_hull* and *update_convex_hull* which appear in the next two subsections are described afterwards.

5.2. Initial approximation

For simplicity's sake we assume that the projection space is the space of $\{x_1, \ldots, x_d\}$.

1. Input

- **1.1.** X = Ineq (or Ineq')
- **1.2.** P (= $\emptyset$) = set of extreme points of current approximation.

2. Compute the first two points of P

- **2.1.** Maximize x_1 over X
- **2.2.** Get a corresponding extreme point of the projection
  Let p_1 = *extreme_point*(x_1, max(x_1), max)
- **2.3.** Minimize x_1 over X

2.4. Let p_2= *extreme_point*(x_1, min(x_1), min)

3. Compute d-1 extreme points

3.1 Let E = $\{p_1, p_2\}$

3.2. Repeat

3.2.1. Compute $\sum_{i=1}^{d} \alpha_i x_i = \alpha_0$ an hyperplane containing the points of E

3.2.2. Minimize h = $\sum_{i=1}^{d} \alpha_i x_i$ over X

3.2.3. If min(h)=α_0 then

3.2.3.1. Maximize h over X

3.2.3.2. Let p=*extreme_point*(h, max(h), max)

3.2.4. else let p=*extreme_point*(h, min(h), min)

3.2.5. Let E=E∪{p}

3.3. Until card(E)=d+1

4. Let CH = *initial_hull*(E)

5. Output:

5.1. E = $\{p_1, ..., p_{d+1}\}$, the set of extreme points of current approximation

5.2. CH = $\{C_1, ..., C_{d+1}\}$, the convex hull of E, all C's labeled *non terminal*

5.3. For each C_i, the set E_i = E - $\{p_i\}$ of points of E that belong to the facet defined by C_i

5.3. Incremental refinement

Compute new extreme points and update the convex hull.

1. Input:

1.1. X = set of inequality constraints (initial set Ineq or Ineq')

1.2. E = $\{p_1, ..., p_{d+1}\}$ set of extreme points

1.3. CH = $\{C_1, ..., C_{d+1}\}$ the convex hull of E

1.4. E_1, E_2, ..., E_{d+1} the sets of extreme points in each C_i of CH

1.5. b = user-supplied upper bound for the size of CH

2. Repeatedly compute new extreme points and update convex hull

For each non terminal $C_i = \{h_i \leq \alpha_i\}$ in CH and

while card(CH) $\leq$ b do

2.1. Maximize h_i over X

2.2. If max(h_i)= α_i then

2.2.1. Label C_i *terminal*

2.3. else

2.3.1. Let p = *extreme_point*(h_i,max(h_i),max)

2.3.2. Let E = E ∪ {p}

2.3.3. Let CH = *update_convex_hull*(p)

3. Output

3.1. Bounded case

Output CH ∪ projection(Eq)

3.2. Unbounded case

3.2.1. Compute E'= set of extreme points of Q (from E and Eq')

3.2.2. For each $e_k \in$ E' compute the corresponding constraint C_k

3.2.3. Output $\{C_k\}$∪ projection(Eq)

5.4. Auxiliary procedures

The first procedure computes an extreme point of the projection. The next procedure computes the convex hull of the initial set of extreme points and the last one performs the update on the current convex hull with a new extreme point added. Many variations and optimizations are possible; some will be discussed in the next section. Our objective here is to present a general method. To give the most appropriate convex hull construction to the problem of quantifier elimination is an interesting problem in its own right, but it is beyond the scope of this paper. Consequently we only give here simple procedures for the sake of completeness. They are in fact used in a prototype implementation briefly discussed in the conclusion.

5.4.1. Procedure *extreme_point*(h,h_0,opt)

Compute an extreme point p of the projection, $\mathbf{h}= \sum_i \alpha_i x_i$.

1. Let Y = X (input constraints set)
2. Replace x_k by $(\mathbf{h}_0 - \sum_{i \neq k} \alpha_i x_i)/\alpha_k$ in Y
3. For i = 1 to d, i $\neq$ k do
   - **3.1.** If **opt** = "max" then maximize x_i over Y else minimize x_i over Y
   - **3.2.** Replace x_i by its optimum m_i in Y
4. Let $m_k = (\mathbf{h}_0 - \sum_{i \neq k} \alpha_i m_i)/\alpha_k$
5. Return $p = \{m_1, ..., m_d\}$

5.4.2. Procedure *initial_hull*(E)

Compute the convex hull of the initial set of extreme points **E**.

1. Let CH=$\emptyset$
2. For each point $p \in \mathbf{E}$ do
   - **2.1.** Compute $\sum_{j=1}^{d} \alpha_j x_j = \alpha$ the equation of the hyperplane defined by $\mathbf{E}-\{p\}$
   - **2.2.** Let $h = \sum_{j=1}^{d} \alpha_j x_j$
   - **2.3.** If $h(p) \geq \alpha$ then CH = CH $\cup\{-h \leq -\alpha\}$ else CH = CH $\cup\{h \leq \alpha\}$
3. Return CH

5.4.3. Procedure *update_convex_hull*(new_pt)

Generate and test potential supporting hyperplanes of new facets of the convex hull and delete obsolete ones.

1. Let p=**new_pt**
2. Generate supporting hyperplanes of new facets:
   For each C $\in$ CH such that $p \notin$ C do
   - **2.1.** Generate all subsets SE of d-1 extreme points in C
   - **2.2.** For each SE do
     - **2.2.1.** If SE $\cup\{p\}$ determines a unique hyperplane $\sum_i \alpha_i x_i = \alpha_0$

2.2.2. Then let $h = \sum_i \alpha_i x_i$
2.2.3. If $\forall q \in E$, $h(q) \leq \alpha_0$
2.2.3.1. then $C = \sum_i \alpha_i x_i \leq \alpha_0$
2.2.3.2. else if $\forall q \in E$, $h(q) \geq \alpha_0$
2.2.3.2.1. then $C = -\sum_i \alpha_i x_i \leq -\alpha_0$
2.2.3.2.2. else $C=\emptyset$
2.2.4. Let $CH = CH \cup \{C\}$
3. Remove old facets:
$\forall C \in CH$ if $p \notin C$ then let $CH = CH - \{C\}$
4. Return CH

6. Comments and Precisions

6.1. Boundedness and GLP

The transformation to a GLP reduces the problem to the bounded case. The difference is in the interpretation of the results at the end of the computation. Simplifying *Ineq* by computing implicit equalities amounts to removing inequalities that do not contribute to the projection and helps eliminate some variables by a simple Gaussian elimination. Furthermore, if the GLP is not solvable, *Ineq* projects on the whole space which is described by the empty set of constraints. So *Ineq* does not contribute any constraints to the projection of S which reduces to the projection of the affine hull *Eq* computed in the previous step.

In both cases, bounded and unbounded, the following steps will apply to a full-dimensional *Ineq* or *Ineq'* whose projection is also full-dimensional, a fact that is used repeatedly in the later stages of the algorithm.

6.2. Initial construction

By taking the projection of arbitrary points in S and constructing their convex hull we obtain a sub-polytope that approximates the projection (see Taylor and Rajan, 1988). The problem is then one of choosing the points in a systematic way that will ensure efficiency and termination of the process. Many variations are possible on how to choose the points and how to construct the convex hull. Choosing a point is a linear programming operation in the initial space, constructing the convex hull is more costly (even though it is in the smaller projection space). To minimize that cost, we only compute points which are extreme points of the projection. The method we propose avoids two pitfalls. One is the repeated computation of the same point, the other is more subtle. If the extreme points are chosen arbitrarily, whatever their number, they could still be all in a cross section of the projection that is of smaller dimension than the projection itself. We would then have a weaker approximation.

Assume d is the dimension of the bounded set X (*Ineq* or *Ineq'*). A first point is obtained as follows. Let x be an arbitrary variable of the projection space. Maximizing x over X gives a value x_0 and defines a face F of X. The projection of F is a face of the projection of X of dimension at most d-1. What we want is a face of X that projects

onto an extreme point (if a simplex is used, a test can in some cases decide if we are in such a situation). We replace x by x_0 in X and we repeat the process by selecting another variable in the projection space. The process terminates with an extreme point P_1 (face of dimension 0) of the projection.

A second point P_2 is found by minimizing x and a similar process. As the projection is full-dimensional, this new point will necessarily be different from the first. We will not proceed with this method to generate a next point, as we could repeatedly compute the same point or compute different points without increasing the dimension of the convex hull. Instead, we compute hyperplanes and use them to generate new points in such a way that at each step the dimension of the convex hull increases until it reaches d, the dimension of the projection space. We compute an arbitrary hyperplane $\alpha x + \beta y + ... = \gamma$ containing the two points P_1 and P_2. By maximizing and minimizing $\alpha x + \beta y + ...$ over X we are guaranteed to find (because of the full-dimensionality) at least one if not two extreme points of the projection by the preceding process. We repeat this construction until we have d+1 extreme points whose convex hull is full-dimensional.

Up to that point there was no need to compute convex hulls. To do it would have been more expensive and would have slowed us in our construction of a full-dimensional approximation. Now we compute the convex hull of the d+1 points.

6.3. Incremental refinement

Let $\{C_1, ..., C_l\}$ be the constraints that define the convex hull and let $C_i = \sum \alpha_i x_i \leq \alpha$. To find a new extreme point we maximize $\sum \alpha_i x_i$. If the maximum is equal to α, it means that C_i is in fact a constraint of the projection; it is labeled *terminal*. If the maximum is not equal to α then we compute a new extreme point and the convex hull is updated.

In principle, any on-line convex hull algorithm (Aggarwal and Wein, 1988; Edelsbrunner, 1987; Preparata and Shamos, 1985) could be used, particularly for dimensions two and three. However for higher dimensions, the algorithms have been mainly designed for theoretical complexity studies (Yao, 1990; Seidel, 1986). They are rarely implemented, because they work under a number of assumptions which conflict with practical use. Also most of these algorithms do not strictly compute the convex hull but rather an incidence graph containing all the faces of the convex hull. Storing this information in memory cuts down on the amount of computation. But the trade-off might not always be desirable as one could have a small convex hull but an inordinately large incidence graph, due to the number of faces of intermediate dimensions. We faced a similar situation with an algorithm to generate extreme points where it was more important to preserve memory usage than speed (Huynh and Lassez, 1990).

Another possibility is to use a simpler and more direct algorithm computing only the constraints of the convex hull. As we are guaranteed that all the points generated are extreme points, it will be considerably more efficient than in the general case and does not require any particular assumption which makes it more palatable from a practical point of view. The idea is to compute hyperplanes that contain the new extreme point and a subset of d-1 previous extreme points. A simple test tells us if they support a facet of the updated convex hull, otherwise they are discarded. The amount of computation can be reduced by avoiding the generation of all the possible hyperplanes. For instance,

we need consider only subsets of d-1 points that belong to the same facet. Another possible reduction is to consider only C_i's such that the new point P does not satisfy the constraint C_i.

7. Empirical Results

We compare the performance of the convex hull method (CHM) presented in this paper with a variant of Fourier's algorithm (FV) and an extreme points method (EPM). The three algorithms were implemented in C and the programs executed on an IBM RS/6000 model 530 under AIX version 3.

Table 1 gives the timings of the three methods to eliminate 4 variables from an initial set of 20 constraints over 7 variables. It is important to note that the output from both Fourier's variant and the extreme points method is highly redundant. To the timing for these two methods, one must add approximately 83s to eliminate the redundant constraints. The times are in virtual CPU seconds.

Method:	FV	EPM	CHM
Output constraints	670	670	61
Runtime	1.4+83	1.4+83	0.3

Table 1.

Because of the doubly exponential complexity of Fourier's method, it is obvious that it is only reasonably practical in very particular cases. The extreme point method can be enhanced by a removal of redundancies in parallel. However there is still a price to be paid for the inherent redundancy and, as for Fourier, this method is applicable only in particular situations. The convex hull method, on the other hand, avoids this problem completely in the bounded case by directly computing the facets of the projection. Table 2 gives some running time for larger sets of constraints. For obvious reasons, only the convex hull method is used as the two other methods flounder on such large inputs. Using the fact that the method does not generate redundancies in the bounded case, the convex hull method is used here to compute the canonical form of an input set of constraints. This particular example is taken from an application to the domain of spatial reasoning that we are studying (Huynh *et al.*, 1991). Because the input set is bounded, projecting it onto itself gives the canonical form: the implicit equalities are detected in the first part of the algorithm and the output of the projection proper consists only of those constraints that define the facets so there is no redundancy.

The first two rows contain respectively the number of input inequalities and input variables. The next two rows contain respectively the number of output equalities and inequalities and the dimension of the canonical form. The last row gives the running time.

Input inequalities	116	375	924	1925
Input variables	16	30	48	70
Output equalities	14	27	44	64
Output inequalities	3	4	5	6
Output dimension	2	3	4	5
Run time	0.2	1.6	8.5	109

Table 2.

8. Conclusion

The main point in this paper is clearly the systematic use of geometric techniques for quantifier elimination (in the linear case). Our study has made clear the fact that much of the complexity of the algebraic approach which concentrates on the elimination of variables is due to the method itself rather than to the nature of the problem. The geometric approach which bypasses this elimination by constructing directly the projection shows that the essential complexity lies within the projection space not the whole space. An important consequence is that one can obtain approximations when the final size of the output is unmanageable. This is important from a theorem proving point of view: it is better to obtain a subset of solutions rather than no solution at all. On the other hand, this may not be an interesting feature in the constraint programming languages setting where, in most cases, an output with auxiliary variables will be preferable to an approximation.

Here we want to make clear a few points that we learned or that were confirmed by preliminary results with a prototype implementation. Not surprisingly, we get excellent runtime performance when the projection space is two-dimensional or when the output is small, as most of the time is spent on running linear programs rather than in the convex hull construction. When the number of variables to eliminate increases, the performance can become arbitrarily better than that of previous methods which in any case tend to give up rapidly, making the comparison more obvious. Another interesting point concerns redundancy in the input. The presence of redundancy compounds the main complexity of algebraic manipulations, as even a small amount can create havoc during variable elimination. In our approach, redundancy in the input does not create a significant problem in the bounded case: it affects only the linear programming part not the convex hull construction, and the output will not be redundant. In the unbounded case, however, redundancy reduces the efficiency of the algorithm as it leads to an increase in the number of extreme points, thus increasing the size of the constructed convex hull. However, we have not enough pragmatic evidence yet to decide whether redundancy in unbounded inputs should be removed prior to projection.

It seems clear that when there are few variables to eliminate, Fourier elimination or the extreme point method should be used, and the convex hull method in the other cases. However, one should note that even with very simple inputs and few variables to eliminate these methods can produce highly redundant output. If this is a concern

then the removal of redundancy will be more costly (far more costly in our experience) than the elimination of variables proper. Indeed, it is easy to construct such examples where the geometric approach is slower than Fourier or the extreme point method, but is far more efficient than these algorithms plus redundancy removal. For a treatment of redundancy in linear constraints see Huynh *et al.* (1989).

The connection that we established between a fundamental algorithm of computational geometry and (albeit in a particular case) a fundamental problem of symbolic computation will hopefully be only a first step. A number of interesting and challenging research problems should follow. We give here some immediate ones.

It seems important to design fast and practical on-line convex hull algorithms which work under the assumption that all points provided are extreme points, or otherwise optimized for being used in projection/quantifier elimination. Alternatively, one could consider randomized algorithms to generate only the constraints of the projection (for the bounded case) or only the extreme points of the projection (for the unbounded case), thus avoiding the case where one set is inordinately large while the other is of a reasonable size and happens to be the one we want. Finally, it seems interesting to consider a generalization of this approach to the nonlinear case. Retaining convexity, one could use or adapt tools from non-linear programming.

As a last remark, a way to handle the unbounded case is to give arbitrary bounds to the variables, thus computing a bounded approximation of the projection. One can then test which faces should be extended to infinity and try larger bounds when necessary. This technique is related to the one proposed in Golan (1991). It is not clear yet how practical this method is. The choice of the bounds greatly affects its efficiency. Bounds which are too small lead to useless computations and bounds which are too large lead to serious numerical instabilities.

Acknowledgment

The authors wish to thank Tien Huynh and Igal Golan for their useful comments.

References

A. Aggarwal and J. Wein (1988), *Computational Geometry*, Lectures Notes for 18.409, Laboratory for Comput. Sci., MIT, Cambridge, MA.

D.S. Arnon (1989), "Geometric reasoning with logic and algebra", *Geometric Reasoning*, D. Kapur and J.L. Mundy, eds., MIT Press, Cambridge, MA.

J.H. Davenport (1988), "Robot motion planning", *Geometric Reasoning*, J. Woodwark, ed., Oxford Science Publication.

J.H. Davenport and J. Heintz (1988), "Real quantifier elimination is doubly exponential", D.S. Arnon and B. Buchberger, eds., *Algorithms in Real Algebraic Geometry*, Academic Press, London.

R.J. Duffin (1974), "On Fourier's analysis of linear inequality systems", *Math. Prog. Study*, **1**, 71–95.

H. Edelsbrunner (1987), *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin-Heidelberg.

I. Golan (1991), *Direct Polyhedron Projection Algorithm*, IBM Research Report RC 16969, IBM T.J. Watson Research Center, NY.

R. Helm, T. Huynh, C. Lassez and K. Marriott (1991), *A Linear Constraint Technology for User Interfaces*, IBM Research Report RC 16913, IBM T.J. Watson Research Center, NY.

T. Huynh, L. Joskowicz, C. Lassez and J-L. Lassez (1991), "Practical tool for reasoning about linear constraints", *Fondamenta Informaticae J.*, special issue on Logics for Artificial Intelligence, to appear.

Preliminary version "Reasoning about linear constraints using parametric queries", *Foundations of Software Technology and Theoretical Computer Science,* Lecture Notes in Comput. Sci. 472, Springer-Verlag, NY, 1990.

T. Huynh and J-L. Lassez (1990), *Practical Issues on the Projection of Polyhedral Sets*, IBM Research Report RC 15872, IBM T.J. Watson Research Center, NY.

T. Huynh, C. Lassez and J-L. Lassez (1990), "Fourier algorithm revisited", *2nd Int. Conf. Algebraic and Logic Prog.*, Lecture Notes in Comput. Sci., Springer-Verlag, NY, 164–173.

T. Huynh, J-L. Lassez and K. McAloon (1989), "Simplification and elimination of redundant linear arithmetic constraints", *Proc. North American Conf. Logic Prog. 89*, MIT Press, Cambridge, MA, 37–51.

P.C. Kanellakis, G.M. Kuper and P.Z. Revesz (1990), "Constraint query languages", *Proc. ACM Conf. Principles of Database Syst.,* Nashville, TN, 298–313.

J-L. Lassez (1990), "Querying constraints", *Proc. ACM Conf. Principles of Database Syst.*, Nashville, TN, 288–298.

J-L. Lassez and K. McAloon (1989), *A Canonical Form for Generalized Linear Constraints*, IBM Research Report RC 15004, IBM T.J. Watson Research Center, NY. *J. Symbolic Computation,* to appear.

F.P. Preparata and M.I. Shamos (1985), *Computational Geometry, An Introduction*, Springer-Verlag, NY.

J.T. Schwartz and M. Sharir (1989), "A survey of motion planning and related geometric reasoning", *Geometric Reasoning,* D. Kapur and J.L. Mundy, eds., MIT Press, Cambridge, MA.

R. Seidel (1986), "Constructing higher dimensional convex hulls at logarithmic cost per face", *Proc. 18th ACM Symp. Theory of Computation*, Berkeley, CA, 404–413.

R.H. Taylor and V.T. Rajan (1988), *The Efficient Computation of Uncertainty Spaces for Sensor-Based Robot Planning,* IBM Research Report RC 13998, IBM T.J. Watson Research Center, NY.

L. Van De Vries (1988), "Alfred Tarski's elimination theory for closed fields", *J. Symbolic Logic*, **53**(1), 7–19.

F. Frances Yao (1990), "Computational geometry", *Handbook of Theoretical Computer Science*, J. van Leeuven, ed., The MIT Press/Elsevier, Cambridge, MA.