Annals of Mathematics and Artificial Intelligence 6 (1992) 295–316

Practical issues on the projection of polyhedral sets

Tien Huynh, Catherine Lassez and Jean-Louis Lassez
IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA

Projection of polyhedral sets is a fundamental operation in both geometry and symbolic computation. In most cases, however, it is not practically feasible to generate projections as the size of the output can be exponential in the size of the input. Even when the size of the output is manageable, we still face two serious problems: overwhelming redundancy and degeneracy. Here, we address these problems from a practical point of view. We discuss three algorithms based on algebraic and geometric techniques and we compare their performance in order to assess the feasibility of these approaches.

1. Introduction

Projection is a very basic operation in geometry, but as projection is also the geometric interpretation of the algebraic operation of variable elimination and of the logical operation of quantifier elimination, it is indeed a most fundamental concept in symbolic computation (see papers and references in [11]). Further motivations from AI and programming languages points of view can be found in [14] and [8]. In principle, the projection of a polyhedral set can be easily achieved by quantifier (variable) elimination. In practice, the fact that the size of the representation of the projected image and of the intermediate computations can be intractable is a serious limitation.

We discuss here three algorithms for projection which do not require any particular hypothesis on the input polyhedral sets. The first two are algebraic in nature. The first one is based on modifications of the classic Fourier's elimination algorithm and is efficient for small sparse systems. The second one uses a formulation of the problem called the Extreme Point Method [14] which works particularly well in dense systems (as opposed to Fourier's). The last algorithm exploits the geometric aspects of the projection operation. It computes the projection by successive approximations of the convex hull of its extreme points using linear programming techniques to find the points directly in the projection space. The complexity of this algorithm depends on the dimension and size of the projection and not on that of the input set.

In a first section we address the problem of constraints representation and present the concept of *parametric form*. Transforming an input set of constraints into parametric form before starting the projection will provide some initial simplifications and also ensure that the input to the algorithms is a full-dimensional polyhedral set. As we shall see later, this can greatly simplify matters.

In the next section, we describe the Fourier Variant algorithm (FV) which incorporates several heuristics to help reduce the number of constraints generated in the course of the projection by eliminating on the fly as many redundant constraints as possible.

In the third section, we describe informally the Extreme Point Method (EPM) which is based on an algorithm to enumerate the extreme points (vertices) presented in [7]. One bottleneck of EPM is degeneracy; another problem is that the number of vertices can be enormous even for small inputs. There are many vertex enumeration algorithms in the literature, but they require storing all vertices until termination and assume no degeneracy. The vertex enumeration algorithm used here not only minimizes the storage of the vertices but also integrates efficiently with Kruse's algorithm [13] to handle degeneracy.

In the fourth section, we discuss the Convex Hull Method (CHM) [16]. A main feature of CHM is that it transforms the unbounded case so that it reduces to the conceptually simple bounded case. The projection is then computed by successive refinements of an initial approximation computed as the convex hull of a number of extreme points of the projection along lines similar to those of [21]. If the size of the projection is too large to be fully computed, the algorithm still provides an approximation which can be an upper or lower bound or both and whose size is user-defined.

In the fifth section, we address the problem of redundancy removal. Redundancy is a very serious factor that accounts for the unmanageable size of the data at various stages of the projection. We show here that redundancy can be very efficiently removed in particular instances by using the concept of subsumption cone. The subsumption cone of a polyhedral set is a characterization of all constraints implied by the given set. Once the subsumption cone of the original polyhedral set is generated then the subsumption cone of any projected image can be derived directly from it. By using the subsumption cone as a filter, the testing of redundancy is simplified from a large linear program for each tested constraint to a straightforward evaluation.

In the last section, we compare the runtime performance of the three algorithms on a number of examples. It greatly depends on the type of constraints given as input. In other words, no algorithms or methods are universally better. Issues often bypassed in the theoretical descriptions of algorithms such as redundancy, round-off errors and degeneracy are of paramount importance. Also, we found that performance is greatly affected by code optimization, so the

design and implementation of a practical system is truly a major task at all levels.

2. Parametric representation

Let $\mathbf{S} = \{\mathbf{Ax} \leq \mathbf{b}\}$ be the set of inequalities describing a given polyhedral set **P**. **A** is the matrix of the coefficients of the variables in **S** and **x** and **b** are respectively column vectors of variables and constants. We denote $\mathbf{A}_{i.}$ the ith row and $\mathbf{A}_{.j}$ the jth column of **A**. A constraint $\mathbf{A}_{i.}\mathbf{x} \leq b_i$ is an *implicit equality* if it can be replaced by $\mathbf{A}_{i.}\mathbf{x} = b_i$ without changing the semantics of **S**. **P** is *full-dimensional* if **S** does not contain any implicit equalities. If **P** is not full-dimensional, collecting all implicit equalities in **S** and applying Gauss–Jordan reduction give the set of equations $\{\mathbf{y} = \mathbf{Ex}' + \mathbf{c}\}$ which defines the *affine hull* of **P**. **E** and **c** are respectively a matrix and a column vector of constants. The variables of $\mathbf{x}'$, which form a subset of the variables of **x**, are called the *parameters*, and the variables of **y** are called the *bound variables* ($\mathbf{y} = \mathbf{x} \setminus \mathbf{x}'$). By eliminating all the bound variables from the remaining inequalities of **S**, we obtain the *parametric representation* $\{\mathbf{A}'\mathbf{x} \leq \mathbf{b}'\} \cup \{\mathbf{y} = \mathbf{Ex}' + \mathbf{c}\}$ where $\{\mathbf{A}'\mathbf{x}' \leq \mathbf{b}'\}$ is full-dimensional in the space of the parameters.

Although not essential in order to compute the projection, a parametric representation has several advantages. First, if **S** initially contains implicit equalities, these can be used to eliminate some variables in a straightforward manner by selecting bound variables from the set **V** of the variables to eliminate. The projection of **P** is $\{\mathbf{A}'\mathbf{x}' \leq \mathbf{b}'\} \cup \{\mathbf{y}' = \mathbf{E}'\mathbf{x}' + \mathbf{c}'\}$, where $\mathbf{y}'$ is a vector of variables of **y** which are not in **V**. Second, transforming a parametric representation into canonical form [18] simply requires the removal of redundancies. The canonical form can help substantially reduce the number of redundant constraints generated in the course of projection. Third, we know (e.g. [15]) that if **S** is in parametric form so is its projection. As we will see later, this property will allow us to better exploit the subsumption cone for redundancy removal. Finally, because the projection of a full-dimensional polyhedral set is also full-dimensional, the construction of the initial convex hull and its incremental refinements in CHM are greatly simplified.

Obviously the price to pay for obtaining a parametric representation is the identification of all implicit equalities. The naive way consists in running, for each inequality $\mathbf{A}_{i.}\mathbf{x} \leq b_i$ in **S**, the following linear program:

$$\begin{aligned} &\text{minimize} && \mathbf{A}_{i.}\mathbf{x} \\ &\text{subject to} && \mathbf{Ax} \leq \mathbf{b}. \end{aligned}$$

If the optimum returned by the above program is b_i, then $\mathbf{A}_{i.}\mathbf{x} \leq b_i$ is an implicit equality. This operation is very costly because it requires running as many linear programs as there are inequalities in **S**. Furthermore if, in fact, there are no

implicit equalities, all this work is wasted. What we need is a test that will first detect the presence of implicit equalities before attempting to identify them. To this end, we use the following formulation called the *quasi-dual* proposed in [14]:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{b} \\ \text{subject to} \quad & \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{A} = \mathbf{0}, \\ & \textstyle\sum \boldsymbol{\lambda} = \mathbf{1}, \\ & \boldsymbol{\lambda} \geq \mathbf{0}, \end{aligned}$$

where $\boldsymbol{\lambda}$ is a column vector of variables. This formulation, in fact, corresponds to Fourier's algorithm, the $\boldsymbol{\lambda}$'s being the multiplicators of the inequalities of **S** that eliminate all variables. When the above linear program returns a minimum of 0, **S** is solvable AND contains implicit equalities (this corresponds to the case when Fourier generates the tautology $0 \leq 0$). This is a key advantage of this method as the search for implicit equalities will be carried out only after it is ascertained that there exist some. Also, the fact that it is, at the same time, a test for solvability is of particular importance for constraint languages design where the solver is a main issue. If the quasi-dual is not solvable or if the minimum is positive then **S** is solvable and does not contain implicit equalities (the associated polyhedral set is full-dimensional). If, during the computation, $\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{b}$ becomes negative then **S** is not solvable. A detailed description of this property of Fourier's algorithm can be found in [17].

When the existence of implicit equalities is detected, that is when $min(\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{b}) = \mathbf{0}$, the indices of the non-zero $\boldsymbol{\lambda}_i$'s in the optimal solution point to a first minimal subset of implicit equalities in **S**.

These implicit equalities are removed from **S**, Gauss–Jordan reduction is applied and the resulting bound variables are eliminated from the remaining inequalities. Once this is done, the quasi-dual is applied to the updated inequalities. The whole process is repeated until no more implicit equalities remain.

In practice, this can still be expensive as, if **S** contains many implicit equalities, many executions of the quasi-dual are required. One improvement consists in trying to derive as many alternative optimal solutions as possible each time a quasi-dual detects the presence of implicit equalities. For instance, in one experiment where **S** consisted of 1819 constraints over 68 variables, simplifying **S** immediately after one subset of implicit equalities was found led to 29 executions of the quasi-dual and took a total of 259 virtual CPU seconds to find all implicit equalities and rewrite **S** into parametric representation. On the other hand, by looking at the coefficients of the objective function, one can find immediately some further solutions. In that case, the process was reduced to only 6 executions of the quasi-dual in 67 seconds. The affine hull contained 58 equalities and 690 inequalities remained before redundant constraints were

removed. A systematic search for all optimal solutions is unfortunately impractical. Further research in this area is in progress.

3. Fourier's Variant

The Fourier's Variant (FV) we present here is a modification of Fourier's algorithm that incorporates three heuristics. The first one, given in [1], helps reduce the number of redundant constraints generated at each stage by selecting to eliminate first the variable that gives the smallest number of combinations. The second one is a rule proposed by Kohler [12] that restricts each combination to be generated with at most $n + 1$ initial constraints where n is the number of variables already eliminated (the rule takes effect only for $n \geq 2$). Although very simple, Kohler's rule helps reduce dramatically the number of redundant constraints generated. However, it does not remove all redundancies and, as is shown in the next example, using arbitrarily other techniques simultaneously can lead to erroneous results. We project

$$\mathbf{P}: \begin{cases} x+y+z \leq 1 \\ x-y+z \leq 1 \\ -x+y+z \leq 1 \\ -x-y+z \leq 1 \end{cases}$$

on the $\{z\}$-plane. After eliminating x, we have

$$\mathbf{P}'_{\{y,z\}}: \begin{cases} y+z \leq 1 \\ z \leq 1 \\ z \leq 1 \\ -y+z \leq 1 \end{cases}$$

At this stage, Kohler's rule cannot detect any redundancies since only one variable has been eliminated. The second and third constraints are, however, both redundant and are removed giving

$$\mathbf{P}'_{\{y,z\}}: \begin{cases} y+z \leq 1 \\ -y+z \leq 1 \end{cases}$$

Eliminating y finally gives $\mathbf{P}'_{\{z\}}$: $\{z \leq 1\}$. This only remaining constraint being a combination of the four original ones is identified as redundant by Kohler's rule and is removed. As a result, the projection becomes the empty set which is obviously wrong.

It is, however, possible to combine Kohler's rule with some other redundancy removal techniques. In particular, the *singular matrix rule* which we describe next forms our third heuristic. This rule is derived from the Extreme Point Method presented in the next section. The idea is that a constraint is redundant

if the coefficients of the eliminated variables in the parent constraints form a singular matrix. For example, to eliminate x_1, x_2 and x_3 from the set:

$$\mathbf{P}: \begin{cases} x_2+x_3-x_4 \le 1 \\ x_1+x_3-x_4 \le 1 \\ -2x_1+x_3-x_4 \le 1 \\ 2x_1-x_3+x_4 \le 1 \\ -x_2-x_3-x_4 \le 1 \\ -x_1-x_3-x_4 \le 1 \end{cases}$$

Fourier's algorithm generates

$$-\tfrac{2}{3}x_4 \le 1$$

using the second, third, fourth and sixth constraints. This combination passes Kolher's rule even though it is redundant but the singular matrix rule will detect it: the submatrix

$$\begin{bmatrix} 1 & -2 & 2 & -1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

which corresponds to the above combination is readily detected to be singular, therefore the associated constraint is redundant. The reason why both Kohler's and the singular matrix rules are safe and why they can be used simultaneously is that both rules identify as redundant constraints which fall outside the set of constraints **Q** generated by the Extreme Point Method. This set **Q**, which is a subset of the set of constraints generated by Fourier's algorithm, contains the projection. So, any constraint outside **Q** can be safely removed as it does not contribute to the projection.

It should be noted that we do not attempt to identify all singular submatrices, only those trivially singular as the one above. The effectiveness of the singular matrix rule depends on the distribution of the zero entries in the matrix of the coefficients of the variables to be eliminated in the given constraints. Obviously, it performs best with sparse systems. For example, in a case of 52 constraints over 12 variables with only 93 non-zero coefficients, FV with this rule returns 71142 constraints after the elimination of 10 variables. Without the rule, the size of the output more than doubles to 153103 constraints.

For an application of these techniques to the case of CLP(R) see [9,10].

4. Extreme Point Method

The Extreme Point Method (EPM) is based on a formulation proposed in [14] which generalizes the problem of linear programming. Let $\mathbf{S} = \{\mathbf{Ax} \le \mathbf{b}\}$ and let

$\mathbf{V}$ be the set of the variables to eliminate. The formulation is as follows. Let Δ be the polytope:

$$\Delta: \begin{cases} \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{A}_{\cdot j} = \mathbf{0} & \forall j \text{ s.t. } x_j \in \mathbf{V} \\ \sum \boldsymbol{\lambda} = \mathbf{1} \\ \boldsymbol{\lambda} \geq \mathbf{0} \end{cases}$$

defined by the combinations of constraints of $\mathbf{S}$ that eliminate the variables of $\mathbf{V}$. Note that the normalization of the $\boldsymbol{\lambda}$'s ensures that Δ is bounded but does not affect the projection. The remaining combinations define the function Φ:

$$\Phi: \begin{cases} \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{A}_{\cdot i} = \boldsymbol{\alpha}_i & \forall i \text{ s.t. } x_i \notin \mathbf{V} \\ \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{b} = \boldsymbol{\beta} \end{cases}$$

where $\boldsymbol{\lambda}$ is a column vector.

The solutions to the generalized linear program $extr(\Phi(\Delta))$, where $extr(\ )$ denotes a function that computes the extreme points, determine a finite set of constraints which defines the projection of $\mathbf{S}$. The extreme points of $\Phi(\Delta)$ are images of extreme points of Δ. Δ is bounded by definition and therefore has a finite number of extreme points. To compute the extreme points of $\Phi(\Delta)$, we first compute the extreme points of Δ and then map them via Φ. However, not all extreme points of Δ map onto extreme points of $\Phi(\Delta)$ so some redundancy will need to be eliminated from the images.

An interesting case occurs when Δ is empty, which means that the associated system of constraints is unsolvable. In this case, the projection is the whole space. This is very useful because we know immediately what the projection is. With FV, many, and possible all, variables of $\mathbf{V}$ may have to be eliminated to reach the same conclusion. In the worst case, FV may exhaust all memory and abort before it finds the answer even though it is a trivial one.

The mechanism of EPM is illustrated by the following example. Let $\mathbf{P}$ be the polyhedral set represented by:

$$\begin{cases} 12x_1 + x_2 - 3x_3 + x_4 \leq 1 \\ -36x_1 - 2x_2 + 18x_3 - 11x_4 \leq -2 \\ -18x_1 - x_2 + 9x_3 - 7x_4 \leq -1 \\ 45x_1 + 4x_2 - 18x_3 + 13x_4 \leq 4 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$$

and let $\mathbf{V} = \{x_1, x_2\}$. The formulation of EPM is:

$$\Phi: \begin{cases} \alpha_3 = -3\lambda_1 + 18\lambda_2 + 9\lambda_3 - 18\lambda_4 \\ \alpha_4 = \lambda_1 - 11\lambda_2 - 7\lambda_3 + 13\lambda_4 \\ \beta = \lambda_1 - 2\lambda_2 - \lambda_3 + 4\lambda_4 \end{cases}$$

$$\Delta: \begin{cases} 12\lambda_1 - 36\lambda_2 - 18\lambda_3 + 45\lambda_4 - \lambda_5 = 0 \\ \lambda_1 - 2\lambda_2 - \lambda_3 + 4\lambda_4 - \lambda_6 = 0 \\ \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 = 1 \\ \lambda_1 \geq 0, \quad \lambda_2 \geq 0, \quad \lambda_3 \geq 0, \quad \lambda_4 \geq 0, \quad \lambda_5 \geq 0, \quad \lambda_6 \geq 0. \end{cases}$$

Δ has six extreme points:

$$\lambda_A = \{\tfrac{1}{14}, 0, 0, 0, \tfrac{6}{7}, \tfrac{1}{14}\},$$
$$\lambda_B = \{\tfrac{3}{5}, \tfrac{1}{5}, 0, 0, 0, \tfrac{1}{5}\},$$
$$\lambda_C = \{\tfrac{1}{2}, 0, \tfrac{1}{3}, 0, 0, \tfrac{1}{6}\},$$
$$\lambda_D = \{0, 0, 0, \tfrac{1}{50}, \tfrac{9}{10}, \tfrac{2}{25}\},$$
$$\lambda_E = \{0, \tfrac{1}{3}, 0, \tfrac{4}{15}, 0, \tfrac{2}{5}\},$$
$$\lambda_F = \{0, 0, \tfrac{1}{2}, \tfrac{1}{5}, 0, \tfrac{3}{10}\},$$

which give:

$$\Phi(\lambda_A) = \{\alpha_3 = -\tfrac{3}{14}, \alpha_4 = \tfrac{1}{14}, \beta = \tfrac{1}{14}\},$$
$$\Phi(\lambda_B) = \{\alpha_3 = \tfrac{9}{5}, \alpha_4 = -\tfrac{8}{5}, \beta = \tfrac{1}{5}\},$$
$$\Phi(\lambda_C) = \{\alpha_3 = \tfrac{9}{6}, \alpha_4 = -\tfrac{11}{6}, \beta = \tfrac{1}{6}\},$$
$$\Phi(\lambda_D) = \{\alpha_3 = -\tfrac{18}{50}, \alpha_4 = \tfrac{13}{50}, \beta = \tfrac{4}{50}\},$$
$$\Phi(\lambda_E) = \{\alpha_3 = \tfrac{18}{15}, \alpha_4 = -\tfrac{3}{15}, \beta = \tfrac{6}{15}\},$$
$$\Phi(\lambda_F) = \{\alpha_3 = \tfrac{9}{10}, \alpha_4 = -\tfrac{9}{10}, \beta = \tfrac{3}{10}\},$$

respectively. The projection of $\mathbf{P}$ on the $\{x_3, x_4\}$-plane is represented by

$$\mathbf{P}'_{\{x_3,x_4\}}: \begin{cases} -3x_3 + x_4 \leq 1 \\ 9x_3 - 8x_4 \leq 1 \\ 9x_3 - 11x_4 \leq 1 \\ -18x_3 + 13x_4 \leq 4 \\ 6x_3 - x_4 \leq 2 \\ 9x_3 - 9x_4 \leq 3 \end{cases}$$

As we remarked earlier, in general, not all extreme points of Δ map onto an extreme point of $\Phi(\Delta)$; some correspond to redundant constraints, like $9x_3 - 9x_4 \leq 3$, and must be removed.

The key operation of EPM is clearly the computation of the extreme points. In consequence, its efficiency will depend largely on the extreme point enumeration algorithm chosen.

A natural algorithm for enumerating the vertices of Δ is based on the Simplex tableau. Degeneracy is a well-known problem, and as the number of extreme points can be enormous even with inputs of small size, this creates a serious bottleneck problem for EPM particularly when the matrix of Δ is sparse. For example, to eliminate x and y from

$$\mathbf{P}: \begin{cases} x+y+z \leq 1 \\ -x-y+z \leq 1 \\ -y+z \leq 1 \\ -x+z \leq 1 \end{cases}$$

we formulate

$$\Delta: \begin{cases} \lambda_1-\lambda_2-\lambda_4=0 \\ \lambda_1-\lambda_2-\lambda_3=0 \\ \lambda_1+\lambda_2+\lambda_3+\lambda_4=1 \\ \lambda_1 \geq 0, \quad \lambda_2 \geq 0, \quad \lambda_3 \geq 0, \quad \lambda_4 \geq 0. \end{cases}$$

Δ has two extreme points, $\lambda_A = \{\frac{1}{2}, \frac{1}{2}, 0, 0\}$ and $\lambda_B = \{\frac{1}{3}, 0, \frac{1}{3}, \frac{1}{3}\}$. λ_A is a degenerate extreme point; there are more than one basis associated with it (we call these different bases *variants*). In general, when the matrix of Δ is sparse there will be many degenerate extreme points.

There are a number of algorithms designed for extreme point enumeration such as those given in the survey of [19]. Unfortunately, they cannot be applied directly for EPM, because they assume no degeneracy and require maintaining the list of all extreme points until termination. This is not feasible in our situation, as it has been shown that the number of variants caused by degeneracy can be enormous [5]. In [7] we proposed an algorithm tailored for EPM. Similarly to the one proposed in [2], it starts with an arbitrary basis as the root and constructs a spanning tree of the edge–vertex graph in a breadth-first search manner. But because we efficiently utilize the indices of the basic variable associated with each node, we substantially reduce the storage required and bypass completely the complex adjacency test used in [2]. Furthermore, this algorithm integrates efficiently the algorithm proposed in [13] to minimize the number of variants in case of degeneracy. For more details see [7].

5. Convex Hull Method

The two previous algorithms are based on algebraic manipulations of the constraints; the Convex Hull Method (CHM) proposed in [16] takes a more geometric approach. It works directly in the projection space and the projection

is incrementally built by successive refinements of an initial approximation given by the convex hull of a number of its extreme points.

In the case of a polytope (bounded polyhedral set) one could use algorithms from computational geometry [3]. That is, first generate the set of extreme points of the polytope, project this set, eliminate the redundant projections and finally construct the convex hull of the remaining points. A natural generalization of this method to the unbounded case, however, leads to a far more complex algorithm (see [6]). 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. Also we face two potential combinatorial explosions in the generation of extreme points and in the generation of the intermediate faces during the construction of the convex hull.

A main feature of CHM is that it transforms the unbounded case so that it reduces to the conceptually simple bounded case. This is done by using the dual formulation presented for EPM. This reformulation can also be used in the bounded case as it leads to an alternative approximation when the size of the output is unmanageable. Linear programming techniques are used to find the extreme points in the projection space. Thus the complexity of the algorithm depends essentially on the dimension of the output not the size of the input. Furthermore, when the size of the projection is too large, we still get an approximation which can be an upper or lower bound or both and whose size can be user-defined.

It is important to state that contrary to many algorithms in computational geometry, we do not make any assumptions on the structure of the input set. CHM tests the input and performs necessary transformations so that the projection proper is carried out on a solvable, bounded and full-dimensional set. In the bounded case, CHM works directly in the projection space and the output consists of the constraints corresponding to the facets of the projection without any redundancy (the left-hand side of each inequality defines a hyperplane supporting a facet; the equalities determine the affine hull of the projection).

Basically, there are two phases: the construction of an initial approximation and the successive refinements. The initial approximation is a minimal full-dimensional convex hull of extreme points of the projection. The first two extreme points are obtained by first minimizing and maximizing, subject to the input set of constraints, an arbitrary variable of the projection space and then computing the corresponding extreme points of the projection. Subsequent points are obtained by optimizing the left-hand side of the equation of an arbitrary hyperplane containing the already computed points. Once $d+1$ distinct extreme points are found, where d is the dimension of the projection space, the initial convex hull is constructed, giving a first full-dimensional approximation. Because at this stage, CHM works on a full-dimensional set of constraints, the

required full-dimensional first approximation of the projection can always be computed with at most $d+1$ points.

In the second phase, this initial approximation is refined by adding new extreme points, if any. To search for new extreme points, each facet of the initial convex hull is swept outward in search of new extreme points; that is, the left hand side of the associated constraint is maximized subject to the input constraints, and if a new point is found, the corresponding extreme point of the projection is computed. Each time such a point is found, the associated constraint is removed and the constraints corresponding to the new facets of the convex hull are added. If no new points are found, the constraint is marked *terminal*, meaning that it corresponds to an actual facet of the projection and does not require further processing. This process is repeated until all constraints are found to be terminal.

For instance, we consider again the first example in the previous section where the variables of the projection space are $\{x_3, x_4\}$. Maximizing and minimizing x_3 give the two extreme points:

$$\{x_3 = \tfrac{1}{2},\ x_4 = 1\} \quad \text{and} \quad \{x_3 = -\tfrac{1}{2},\ x_4 = -\tfrac{1}{2}\}.$$

The hyperplane (unique here) passing through these two points is:

$$-6x_3 + 4x_4 = 1.$$

Maximizing $-6x_3 + 4x_4$ gives another extreme point:

$$\{x_3 = -\tfrac{3}{7},\ x_4 = -\tfrac{2}{7}\}.$$

As $d = 2$, there are now enough points for the initial convex hull which is:

$$\mathbf{P}_0'\colon \begin{cases} 6x_3 - 4x_4 \leq -1 \\ -\tfrac{9}{2}x_3 + \tfrac{13}{4}x_4 \leq 1 \\ -3x_3 + x_4 \leq 1 \end{cases}$$

The left-hand side of the first constraint is maximized giving the new extreme point:

$$\{x_3 = \tfrac{5}{13},\ x_4 = \tfrac{4}{13}\}.$$

The constraint is therefore discarded and the convex hull augmented by:

$$3x_3 - \tfrac{1}{2}x_4 \leq 1 \quad \text{and} \quad 21x_3 - 23x_4 \leq 1,$$

generated with the new point. Repeating this process, $-\tfrac{9}{2}x_3 + \tfrac{13}{4}x_4 \leq 1$, $-3x_3 + x_4 \leq 1$ and $3x_3 - \tfrac{1}{2}x_4 \leq 1$ are found to be terminal. Maximizing $21x_3 - 23x_4$ does give the new extreme point:

$$\{x_3 = \tfrac{1}{9},\ x_4 = 0\}.$$

$21x_3 - 23x_4 \leq 1$ is removed and

$$9x_3 - 11x_4 \leq 1 \quad \text{and} \quad 9x_3 - 8x_4 \leq 1$$

are added which are found to be terminal. Consequently, the projection is:

$$\mathbf{P}'_{\{x_3,x_4\}}: \begin{cases} -18x_3+13x_4 \le 4 \\ -3x_3 \quad +x_4 \le 1 \\ 6x_3 \quad -x_4 \le 2 \\ 9x_3-11x_4 \le 1 \\ 9x_3 \quad -8x_4 \le 1 \end{cases}$$

which contains no redundancy and has five extreme points.

When the projection is not bounded, CHM works on the dual formulation used for EPM. The combined set of constraints defining $\boldsymbol{\Phi}$ and $\boldsymbol{\Delta}$ which is bounded by definition (see previous section) forms the input set. CHM can now be applied to that set as in the bounded case, the lambda's being the variables to eliminate. As we saw above, CHM computes the constraints defining the projection but *in the process it also computes the extreme points of the projection.* In the unbounded case, as we work in a dual space, it is those extreme points that determine the constraints defining the projection as illustrated by the following example. Let

$$\mathbf{S}: \begin{cases} -x_1 \quad +x_2 \quad -x_3+x_4-x_5 \le 1 \\ -x_1-2x_2-2x_3+x_4-x_5 \le 1 \\ 2x_1 \quad +x_2+2x_3-x_4+x_5 \le 1 \\ 2x_1 \quad +x_2 \quad -x_3+x_4-x_5 \le 1 \\ -x_1 \quad -x_2+2x_3+x_4-x_5 \le 1 \end{cases}$$

be the input set and $\mathbf{V}=\{x_1, x_2\}$ the set of variables to eliminate from **S**. **S** represents an unbounded polygon and its projection on $\{x_3, x_4, x_5\}$ is also unbounded. The corresponding EPM formulation is:

$$\boldsymbol{\Phi}: \begin{cases} -\lambda_1-2\lambda_2+2\lambda_3-\lambda_4+2\lambda_5=\alpha \\ +\lambda_1 \quad +\lambda_2 \quad -\lambda_3+\lambda_4 \quad +\lambda_5=\beta \\ -\lambda_1 \quad -\lambda_2 \quad +\lambda_3-\lambda_4 \quad -\lambda_5=\gamma \\ \lambda_1 \quad +\lambda_2 \quad +\lambda_3+\lambda_4 \quad +\lambda_5=b \end{cases}$$

$$\boldsymbol{\Delta}: \begin{cases} -\lambda_1 \quad -\lambda_2+2\lambda_3+2\lambda_4-\lambda_5 \quad 0 \\ \lambda_1-2\lambda_2 \quad +\lambda_3 \quad +\lambda_4-\lambda_5=0 \\ \lambda_1 \quad +\lambda_2 \quad +\lambda_3 \quad +\lambda_4+\lambda_5=1 \\ \lambda_1 \ge 0, \quad \lambda_2 \ge 0, \quad \lambda_3 \ge 0, \quad \lambda_4 \ge 0, \quad \lambda_5 \ge 0. \end{cases}$$

CHM actually computes the projection of the system consisting of the constraints of $\boldsymbol{\Phi}$ and $\boldsymbol{\Delta}$ on the space of $\{\alpha, \beta, \gamma, b\}$. This, in fact, corresponds to

computing the image of Δ by Φ see [14]. The extreme points computed by CHM are:

$$\left\{\tfrac{3}{2}, \tfrac{1}{3}, -\tfrac{1}{3}, 1\right\} \quad \left\{-\tfrac{1}{2}, 1, -1, 1\right\} \quad \left\{-\tfrac{4}{3}, 1, -1, 1\right\} \quad \left\{-\tfrac{1}{3}, \tfrac{1}{3}, -\tfrac{1}{3}, 1\right\}$$

from which the constraints:

$$\begin{cases} \tfrac{3}{2}x_3 + \tfrac{1}{3}x_4 - \tfrac{1}{3}x_5 \le 1 \\ -\tfrac{1}{2}x_3 + x_4 - x_5 \le 1 \\ -\tfrac{4}{3}x_3 + x_4 - x_5 \le 1 \\ -\tfrac{1}{3}x_3 + \tfrac{1}{3}x_4 - \tfrac{1}{3}x_5 \le 1 \end{cases}$$

which define the projection are derived.

In practice, CHM works well when the dimension of the projection space is small and the projection is bounded. As the dimension of the projection space increases, the cost of constructing the intermediate convex hulls increases as well. In particular, when the projection contains many extreme points the situation becomes worse. For example, if $d = 7$ and the facet giving a new point already has 30 extreme points, building all the new facets requires consideration of $\binom{30}{6} = 593775$ possible combinations!

Still, a main advantage of CHM is that it can be used to generate an approximation of the projection if the number of facets becomes unmanageable. Furthermore, if the projection is bounded, by storing the facets of the intermediate convex hulls in a queue, the approximation can be made to grow evenly in all directions. In the unbounded case, however, we have no control over how the approximation develops.

6. Redundancy removal via subsumption cone

Redundancy is a major problem in the computation of projections particularly in FV and EPM. It can be so severe that the projection becomes unmanageable due to the overwhelming number of redundant constraints generated during the process. A naive way to remove redundancy is to test each constraint with a linear program. If the set of constraints is large, however, this approach is impractical. Also, because these tests must wait until all the constraints are generated, memory may be exhausted beforehand. Here we introduce another technique to remove redundancy on the fly, which utilizes the concept of the *subsumption cone* described in [14]. The subsumption cone is essentially the polar cone [20] but defined in a constructive manner (differences between the two definitions appear in cases such as unsolvability which are not relevant here). The cost of this technique depends mainly on the cost of generating the subsumption cone. The theoretical limitations of this method are far more severe than for the naive method, because it is a projection problem rather than

a linear programming problem. When the subsumption cone can be efficiently generated, however, the problem of removing redundancy becomes far simpler.

Let $\mathbf{S} = \{\mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ where $\mathbf{A}$ is an $m \times n$ matrix. The subsumption cone of $SC(\mathbf{P})$ of the polyhedral set $\mathbf{P}$ associated with $\mathbf{S}$, is obtained by eliminating all λ_i's and the variable q from the system:

$$\begin{cases} \boldsymbol{\alpha} = \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{A} \\ \beta = \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{b} + q \\ \boldsymbol{\lambda} \geq \mathbf{0} \\ q \geq 0 \end{cases}$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\lambda}$ are column vectors of dimension n and m respectively and β is a variable.

To compute $SC(\mathbf{P})$, we first apply Gauss–Jordan reduction to the system of homogeneous equations:

$$\begin{cases} \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{A} - \boldsymbol{\alpha} = \mathbf{0} \\ \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{b} + q - \beta = 0 \end{cases}$$

to derive the row-reduced echelon form with λ_i's and q being the fixed variables (a fixed variable has the first non-zero coefficient in a row after Gauss–Jordan reduction). Because q appears only in the last equation, we can assume that it is always a fixed variable. As for the λ_i's, some may stay free after the reduction depending on the matrix $\mathbf{A}$. To eliminate the fixed variables we substitute them into the inequalities $\boldsymbol{\lambda} \geq \mathbf{0}$ and $q \geq 0$ and then discard the equalities where they are fixed. If any λ_i's stay free, we have to eliminate them from the subsequent system of inequalities via projection. The resulting system contains a set of relations between the α_i's and β which characterize all the constraints implied by $\mathbf{P}$.

For example, the subsumption cone of the polyhedral set $\mathbf{P}$ given as the first example in section 4 is:

$$SC(\mathrm{P}): \begin{cases} -\frac{1}{3}\alpha_1 & -\frac{5}{6}\alpha_3 - \frac{2}{3}\alpha_4 + \frac{13}{6}\beta \geq 0 \\ -\frac{1}{9}\alpha_1 & -\frac{1}{9}\alpha_3 + \beta \geq 0 \\ & -\frac{1}{3}\alpha_3 - \frac{2}{3}\alpha_4 + \frac{2}{3}\beta \geq 0 \\ & \frac{1}{6}\alpha_3 + \frac{1}{9}\alpha_4 + \frac{7}{18}\beta \geq 0 \\ & \frac{1}{3}\alpha_3 + \frac{1}{3}\alpha_4 + \frac{2}{3}\beta \geq 0 \\ -\alpha_2 & + \beta \geq 0 \\ & \frac{1}{18}\alpha_3 + \frac{1}{2}\beta \geq 0 \end{cases}$$

To use the subsumption cone to remove redundancy from the description of a projection $\mathbf{P}'$ of $\mathbf{P}$, we need to generate $SC(\mathbf{P}')$. The formulation to compute

$SC(\mathbf{P}')$ is:

$$\begin{cases} \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{A}_{.i} = 0 & \forall i \text{ s.t. } x_i \in \mathbf{V} \\ \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{A}_{.j} - \alpha_j = 0 & \forall j \text{ s.t. } x_j \notin \mathbf{V} \\ \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{b} + q - \beta = 0 & \\ \boldsymbol{\lambda} \geq \mathbf{0} & \\ q \geq 0 & \end{cases}$$

where $\mathbf{V}$ is the set of variables to be eliminated. If $SC(\mathbf{P})$ has already been computed however, $SC(\mathbf{P}')$ can be obtained directly using the following result that:

> The subsumption cone $SC(\mathbf{P}')$ is derived from the subsumption cone $SC(\mathbf{P})$ of $\mathbf{P}$ by setting to zero, in the constraints of $SC(\mathbf{P})$, the coefficient α_i corresponding to each variable x_i to be eliminated.

This result, whose proof is straightforward from the construction of $SC(\mathbf{P})$, is very useful when $\mathbf{P}$ is projected on various subspaces. For instance, from the subsumption cone given above we can derive immediately the subsumption cone of the projection on the $\{x_3, x_4\}$ plane as

$$SC(\mathbf{P}')_{\{x_3,x_4\}}: \begin{cases} -\frac{5}{6}\alpha_3 - \frac{2}{3}\alpha_4 + \frac{13}{6}\beta \geq 0 \\ -\frac{1}{9}\alpha_3 + \beta \geq 0 \\ -\frac{1}{3}\alpha_3 - \frac{2}{3}\alpha_4 + \frac{2}{3}\beta \geq 0 \\ \frac{1}{6}\alpha_3 + \frac{1}{9}\alpha_4 + \frac{7}{18}\beta \geq 0 \\ \frac{1}{3}\alpha_3 + \frac{1}{3}\alpha_4 + \frac{2}{3}\beta \geq 0 \\ \beta \geq 0 \\ \frac{1}{18}\alpha_3 + \frac{1}{2}\beta \geq 0 \end{cases}$$

We know that if $\mathbf{P}$ is full-dimensional then each extreme ray of $SC(\mathbf{P})$ is associated with a constraint which defines a facet of $\mathbf{P}$. Therefore, to test whether a constraint in the description of a projection $\mathbf{P}'$ is redundant, one simply tests if that constraint can be derived from an extreme ray of $SC(\mathbf{P}')$. If $\mathbf{P}$ is not full-dimensional then $SC(\mathbf{P})$ is not pointed and is not the convex closure of its extreme rays. For example, if we are given:

$$\mathbf{P}: \begin{cases} x - y \leq 0 \\ -x + y \leq 0 \\ x \leq 1 \\ -x \leq 0 \end{cases}$$

then the subsumption cone of **P** is:

$$SC(\mathbf{P}): \begin{cases} \alpha_1 + \alpha_2 - \beta \leq 0 \\ -\beta \leq 0 \end{cases}$$

which is an unbounded wedge. This case, which we will not address here, requires more complex rules in order to use the subsumption cone to detect redundancy. We find it easier to use the parametric representation.

Although the constraints defining a subsumption cone may consist of both equalities and inequalities, the equalities are ignored in the course of testing redundancy because they are satisfied by all points. If the dimension of a subsumption cone is d, an extreme ray is defined by $d-1$ inequalities of rank $d-1$. This can result in an expensive operation if it is applied systematically. Instead, we use here a simpler and far cheaper test at the expense of completeness. It simply counts the number of inequalities in the subsumption cone which are satisfied as equalities by the coefficients and the right-hand-side constant of the tested constraint. It is based on the remark that a constraint is redundant if its coefficients and right-hand-side constant do not satisfy at least $d-1$ of these inequalities as equalities. For instance, in the first example given in section 4,

$$\Phi(\lambda_F) = \left\{\alpha_3 = \tfrac{9}{10},\ \alpha_4 = -\tfrac{9}{10},\ \beta = \tfrac{3}{10}\right\}$$

does not belong to any facet of $SC(\mathbf{P}'_{\{x_3,x_4\}})$ so it is redundant.

Note that in principle some points may satisfy this test and still correspond to redundant constraints, even though it occurs only rarely in practice (see next paragraph). Another main advantage of this technique is that redundancy can be tested on the fly while the constraints are being generated. This reduces significantly the memory usage as we will see later.

As a final remark, one could use the above result to improve Fourier's algorithm. Once we have the subsumption cone for **P**, we have all the subsumption cones for the intermediate projections generated by Fourier's algorithm. Therefore, we can use these subsumption cones to remove redundancy at each intermediate step. Even though there are cases where this method is very efficient, in general it does not remove enough redundancy to avoid combinatorial explosions during the Fourier's steps. This is due to the type of redundancies generated by Fourier's algorithm.

7. Performance evaluation

We now compare the performance of the three methods discussed in this paper. We have implemented all three algorithms in C. The computations are carried out on an IBM RS/6000 model 530 operating under AIX version 3. The measurements are given in virtual CPU seconds. Each timing is the total time of preprocessing and computing the projection. Note that, because of the nature of

the input (randomly generated sets of constraints) both FV and EPM give the same set of constraints as output. To construct examples where FV generates more constraints than EPM, one only needs to have FV generate a combination of $d+1$ (d is the number of variables to be eliminated) constraints which corresponds to a non-trivially singular submatrix in the system of $\boldsymbol{\Delta}$.

The first test deals with sparse systems; that is, the matrix of the coefficients of the variables to be eliminated is sparse. For this purpose we use the interesting geometry application of constructing the convex hull of a given set of points. The formulation of the problem comes form the observation that a point with coordinates $(x_1, x_2, \ldots, x_n)$ is in the convex hull of a set of m given points

$$\{(\alpha_{1,1}, \alpha_{1,2}, \ldots, \alpha_{1,n}), (\alpha_{2,1}, \alpha_{2,2}, \ldots, \alpha_{2,n}), \ldots, (\alpha_{m,1}, \alpha_{m,2}, \ldots, \alpha_{m,n})\}$$

iff there exist λ_i's such that the system

$$\begin{cases} x_1 = \sum_{i=1}^{m} \alpha_{1,i}\lambda_i \\ x_2 = \sum_{i=1}^{m} \alpha_{2,i}\lambda_i \\ \vdots \\ x_n = \sum_{i=1}^{m} \alpha_{n,i}\lambda_i \\ \sum_{i=1}^{m} \lambda_i = 1 \\ \lambda_i \geq 0 \quad \forall i \end{cases}$$

is satisfied. The convex hull of the given points is an equivalent set of relations solely between the x_i's. It can be obtained by eliminating all the λ_i's in the above system. Hence, if m points are given then m variables are to be eliminated. We can view this problem as computing the projection of the polyhedral set represented by the above system on the $\{x_1, x_2, \ldots, x_n\}$-space. Clearly from the formulation of the problem, if m is large the matrix of the coefficients of the constraints becomes very sparse.

Table 1 summarizes the results of running these algorithms to compute the convex hull of 12 samples of randomly generated points in two, three, and four dimensions. The first row shows the number of given points. The second row shows the number of constraints (facets) in the convex hull generated by all three methods. Usually the outputs of both FV and EPM contain redundancy, but due to the special structure of the matrix in this specific application they both generate only non-redundant constraints. The fourth row gives the number of vertices of the convex hull (this information is obtained as a by-product of CHM only). The last three rows record the execution times for each method.

Table 1
Results of the convex hull problem

	2 dimensions				3 dimensions				4 dimensions			
Points	100	200	300	400	100	200	300	400	100	200	300	400
Facets	12	14	18	18	52	88	94	108	218	391	489	592
Vertices	12	14	18	18	28	46	49	56	47	79	94	115
FV	1.9	14.5	44.5	99.8	4.3	30.9	70.8	173.4	19.0	153.0	437.3	874.9
EPM	1.1	5.1	16.2	29.1	2.6	26.9	65.7	131.4	7.4	106.8	288.6	609.4
CHM	1.2	8.8	30.0	54.0	2.9	31.8	85.1	189.7	5.8	82.8	216.9	454.7

For sets of low dimension (2 or 3), EPM is the most efficient. This is because the matrix associated with Δ (as defined in section 4) is nearly square: the number of rows in the matrix of Δ is equal to the number of points less the dimension of the points space, and the number of columns is the number of points; therefore, when the dimension is small, the matrix is nearly square. As the dimension increases the number of columns also increases with respect to the number of rows. Hence, the number of submatrices in Δ grows accordingly. This, together with the problem of degeneracy, slows down EPM when compared to CHM as shown in the last four columns of the table (4 dimensions).

In the next test, we consider a set of constraints for which the associated matrix is dense. This set is randomly generated and consists of 20 constraints over 7 variables, and all the coefficients in the constraints are non-zero. It is bounded and consequently so are its projections. Table 2 summarizes the results of eliminating 2 to 5 variables by all three methods. The first row shows the number of variables eliminated in each test. The second and third rows give respectively the number of facets and the number of vertices in each projection. The fourth row gives the number of constraints generated by CHM. Since the projections are bounded, the output of CHM does not contain any redundancy, which explains why the numbers in this row and in row 2 are identical. The fifth row gives the number of constraints generated by both FV and EPM, which

Table 2
Results of projection where the input contains 20 constraints and 7 variables

Variables eliminated	2	3	4	5
Facets	170	158	99	26
Vertices	352	223	109	26
Constraints generated by CHM	170	158	99	26
Constraints generated by FV/EPM	300	669	1188	1462
FV	0.6	1.3	3.0	5.3
EPM	0.8	1.6	3.1	3.8
CHM	43.4	4.2	0.5	0.1

obviously contain many redundant constraints. The last three rows give the execution times.

These results show that when only 2 or 3 variables are eliminated, FV and EPM perform better than CHM and FV is the most efficient one. As mentioned in section 5, it is very costly to construct convex hulls when the dimension of the space is high. In this case, the dimension of the projection space is 5 (or 4) when 2 (or 3) variables are eliminated. Hence, CHM takes a long time to construct the convex hull. Although CHM has the advantage of not generating any redundant constraints, FV and EPM can achieve the same result in less time using the subsumption cone, as will be shown next. As more variables are eliminated and the dimension of the projection space decreases, CHM performance improves. This is evident in the last two columns of the table. For instance, when 5 variables are eliminated CHM is 38 times faster than EPM and 53 times faster than FV. CHM is at its best when the dimension of the projection is small; this is well illustrated in [8] by examples of spatial reasoning where one is particularly interested in projecting onto the 3-dimensional space of the coordinates.

Finally, we test the feasibility of using the subsumption cone to remove redundancy in the projections. We use the polyhedral set of the previous example as input and we run EPM with and without the subsumption cone to generate the projections. The subsumption cone of the input set has 404 inequalities. Table 3 records the results. The first row shows the number of variables eliminated. The second row shows the number of constraints with redundancy generated by EPM. The third row shows the number of constraints remaining after the subsumption cone was used to remove the redundant ones. The fourth row gives the timings of EPM alone. The last row gives the total timings of EPM plus generating the subsumption cone and using it to remove redundancy. From the table one can see that a substantial amount of redundancy is removed with the subsumption cone. In fact, here, all the redundant constraints generated are removed. When compared with table 2, the first two columns indicate that EPM can achieve the same results as CHM but in lesser time even with the extra computation required for the subsumption cone. This would be true for FV as well if we use the subsumption cone to remove redundancy at the end. As an interesting remark, it is not true that removing redundant constraints always costs more time than ignoring them. In fact

Table 3
Results of EPM with and without subsumption cone

Variables eliminated	2	3	4	5
Constraints generated with redundancy	300	669	1188	1462
Constraints generated without redundancy	170	158	99	26
EPM	0.8	1.6	3.1	3.8
EPM + subsumption cone	1.6	2.8	4.9	6.0

sometimes it is even faster. For example, we have one case where EPM alone takes 17.2 seconds to generate 3155 constraints in the projection, whereas EPM plus subsumption cone generates the same projection without redundancy (reduced to 50 constraints) in only 17 seconds. This is because the subsumption cone detects redundant constraints as soon as they are generated and hence, eliminates the time to allocate memories for storing them.

References

[1] R.J. Duffin, On Fourier's analysis of linear inequality systems, Math. Progr. Study 1 (1974) 71–95.

[2] M.E. Dyer and L.G. Proll, An algorithm for determining all extreme points of a convex polytope, Math. Progr. 12 (1977) 81–96.

[3] H. Edelsbrunner, *Algorithms in Computational Geometry* (Springer, 1987).

[4] J.B.J. Fourier, reported in: Analyse des travaux de l'Académie Royale des Sciences, pendant l'année 1824, Partie mathématique, *Histoire de l'Académie Royale des Sciences de l'Institut de France 7* (1827) pp. 47–55. (Partial English translation in: D.A. Kohler, Translation of a Report by Fourier on his work on Linear Inequalities, Opsearch 10 (1973) 38–42.

[5] T. Gal, On the structure of the set bases of a degenerate point, J. Optim. Theory Appl. 45 (1985) 577–589.

[6] I. Golan, Direct polyhedron projection algorithm, IBM Research Report, T.J. Watson Research Center, RC 16969 (1991).

[7] T. Huynh and J.-L. Lassez, Extreme point enumeration applied to projection of polyhedral sets, in preparation.

[8] T. Huynh, L. Joskowicz, C. Lassez and J.-L. Lassez, Reasoning about linear constraints using parametric queries, in: *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Sciences vol. 472 (Springer, 1990).

[9] J. Jaffar, M.J. Maher, P.J. Stuckey and R.H.C. Yap, Output in CLP(R), to appear, *FGCS'92*.

[10] J. Jaffar, S. Michaylov, P.J. Stuckey and R.H.C. Yap, The CLP(R) language and system, to appear, *TOPLAS'92*.

[11] Special issue on: Algorithms in Real Algebraic Geometry, J. Symbol. Comput. 5 (1988).

[12] D.A. Kohler, Projections of convex polyhedral sets, Operational Research Center Report, ORC 67-29, University of California, Berkeley (1967).

[13] H.-J. Kruse, *Degeneracy Graphs and the Neighborhood Problem*, Lecture Notes in Economics and Mathematical Systems No. 260 (Springer, 1986).

[14] J.-L. Lassez, Querying constraints, *Proc. ACM Conf. on Principles of Database Systems*, Nashville (1990).

[15] J.-L. Lassez, T. Huynh and K. McAloon, Simplification and elimination of redundant arithmetic constraints, *Logic Programming: Proc. North American Conf.* (MIT Press, 1989).

[16] C. Lassez and J.-L. Lassez, Quantifier elimination for conjunctions of linear constraints via a convex hull algorithm, IBM Research Report, T.J. Watson Research Center, RC 16779 (1991), also in: *Symbolic and Numerical Computation – Towards Integration*, eds. Kapur and Mundy (Academic Press).

[17] J.-L. Lassez and M.J. Maher, On Fourier's algorithm for linear arithmetic constraints, IBM Research Report, T.J. Watson Research Center, RC 14114 (1988), to appear in J. Autom. Reasoning.

[18] J.-L. Lassez and K. McAloon, A canonical form for generalized linear constraints, IBM Research Report, T.J. Watson Research Center, RC 15004 (1989), to appear in J. Symbol. Comput.

[19] T.H. Matheiss and D.S. Rubin, A survey and comparison of methods for finding all vertices of convex polyhedral sets, Math. Oper. Res. 5 (1980) 167–185.
[20] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization* (Wiley, 1988).
[21] R.H. Taylor and V.T. Rajan, The efficient computation of uncertainty spaces for sensor-based robot planning IBM Research Report, T.J. Watson Research Center, RC 13998 (1988).