Solving Parametric Linear Systems: an Experiment with Constraint Algebraic Programming

Clemens Ballarin*
Institut für Informatik
Technische Universität München
80290 München
Germany
ballarin@in.tum.de

Manuel Kauers
Fakultät für Informatik
Universität Karlsruhe (TH)
76128 Karlsruhe
Germany
manuel@kauers.de

Abstract

Algorithms in computer algebra are usually designed for a fixed set of domains. For example, algorithms over the domain of polynomials are not applicable to parameters because the inherent assumption that the indeterminate X bears no algebraic relation to other objects is violated.

We propose to use a technique known as *constraint programming* to gain more flexibility, and we show how it can be applied to the Gaussian algorithm to be used for parametric systems. Our experiments suggest that in practice this leads to results comparable to the algorithm for parametric linear systems by Sit (1992) — at least if the parameters are sparse.

1 Introduction

A common feature of computer algebra systems is that the variable symbols they manipulate bear no algebraic or logical relation to other objects. Their semantics is that of *indeterminates*, not of *logical variables* or *parameters*. This semantics is common to almost all algorithms in computer algebra.

The choice of the semantics of symbols makes an important difference. Linear equation systems are the focus of this paper. The rank of a matrix over a polynomial domain $k[X]$ may become smaller if values are substituted for the indeterminate X, and hence the solution space of the corresponding equation system differs. As a consequence, the *generic* solution obtained over $k[X]$ is not necessarily correct under substitutions over k. Even worse, special solutions can not necessarily be obtained from the generic solution or from the original matrix. The following example, taken from the textbook by Noble and Daniel (1988) illustrates this.

*Corresponding author. Work carried out while at the Universität Karlsruhe.

Example 1 *The augmented matrix*

$$\left(\begin{array}{ccc|c} 1 & -2 & 3 & 1 \\ 2 & x & 6 & 6 \\ -1 & 3 & x-3 & 0 \end{array}\right)$$

has the reduced row echelon form

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & \frac{x+9}{x+4} \\ 0 & 1 & 0 & \frac{4}{x+4} \\ 0 & 0 & 1 & \frac{1}{x+4} \end{array}\right).$$

It is immediate that $x = -4$ is a special case. There is another special case $x = 0$, which is not obvious from the result. For $x = 0$ and $x = -4$ the reduced row echelon forms are

$$\left(\begin{array}{ccc|c} 1 & 0 & 3 & 3 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array}\right) \quad \text{and} \quad \left(\begin{array}{ccc|c} 1 & 0 & -5 & 0 \\ 0 & 1 & -4 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right)$$

respectively, and the sets of solutions differ from the general case.

The effect illustrated in the example is known as *specialisation problem.*

There are some occasions where computer algebra systems deal with the specialisation problem. The integrator of Macsyma addresses the problem by asking the user if, for example, knowledge about the sign of a parameter is neccessary for further computations (Davenport *et al.*, 1993). Maple provides an assume facility, which maintains a global context of assumptions on symbolic objects. Assumptions are asserted by the user and can be queried during a computation (Weibel and Gonnet, 1993).

Asking the user is not practical if many queries have to be resolved. An unnerved user likely aborts the computation after the tenth query or so. More importantly, if the symbol was introduced during the computation itself, the user is in no position of answering such queries. This can conceivably happen if the procedure issuing the queries is called as a subroutine by another algorithm that introduced the symbol.

In the present paper, we propose a general approach to the specialisation problem that is based on ideas from *constraint logic programming*, and that allows for the exchange of constraints between sub-programs in a natural way. The approach is illustrated with the probably simplest algorithm where the specialisation problem occurs: Gaussian elimination. The approach is feasible, and the capability of solving linear systems can be improved substantially by combining it with suitable strategies.

2 Constraint Logic Programming

The notion of *constraint store* plays a central role in constraint logic programming. A constraint store is a data-structure that maintains knowledge about objects that appear in the current computation. The knowledge may have been asserted as part of the problem specification, or it may have been discovered during the computation. This knowledge is also known as *context.* The purpose

of this section is to introduce the notion of constraint store. We do this in the context of logic programming.

The programming language Prolog is based on the resolution principle (Robinson, 1965), which is combined with a fixed control strategy. A Prolog program is a set of *clauses*. A clause is a set of negated and non-negated predicates (called *literals*) and represents their disjunction. During the execution of a Prolog program, the resolution rule is repeatedly applied to pairs of clauses, and new clauses are derived. The resolution rule has the following form.

$$\frac{R \cup \{p(s_1, \dots, s_n)\} \qquad S \cup \{\neg p(t_1, \dots, t_n)\}}{R\sigma \cup S\sigma} \tag{1}$$

Here σ denotes the *most general unifier (mgu)* of $p(s_1, \dots, s_n)$ and $p(t_1, \dots, t_n)$. The sets of variables occurring in the clauses must be disjoint, and the rule can only be applied if the mgu exists. The resolution process terminates when an empty clause is derived. Unifiers are accumulated along the computation and provide an *answer substitution.*

Constraint logic programming is based on the observation that matching two literals and unifying their terms can be disentangled (Frühwirth and Abdennadher, 1997). Unification decides if $s_1 = t_1 \wedge \dots \wedge s_n = t_n$ is consistent with syntactic equality of terms built of function and constant symbols. This is also known as *Clark's equation theory.*

In ordinary logic programming, this theory is fixed. The generalised resolution rule used in constraint logic programming makes it possible to extend Prolog to other equation theories T_e where satisfiability can be decided.

$$\frac{R \cup \{p(x_1, \dots, x_n)\}, C \qquad S \cup \{\neg p(y_1, \dots, y_n)\}, D}{R \cup S, C \wedge D} \tag{2}$$

The rule can only be applied if $c \wedge d$ is satisfiable. Note that literals now only may contain variables, not arbitrary terms, and that clauses are enriched by constraints.

From an abstract point of view, the decision procedure for T_e provides a service that can be encapsulated in a separate module, and that can also be used in applications other than logic programming. Often, for reasons of efficiency, it is required that the decision procedure is incremental. This leads to the concept of a *reasoning specialist* for a theory T_e with the following interface functionality.

- $\mathsf{cs\text{-}init}()$: returns a T_e-valid constraint store — for example, the empty constraint store.
- $\mathsf{cs\text{-}unsat}(C)$: true only if C is T_e-unsatisfiable.
- $\mathsf{cs\text{-}simp}(C, D)$: the main functionality of the reasoning specialist. This function adds the assertions in D to the constraint store C, obtaining a new constraint store. For soundness it is required that if $\mathsf{cs\text{-}simp}(C, D) = C'$ then $C, D \models_{T_e} C'$.

The interface functionality of the reasoning specialist is sometimes extended by other functions — for example, by a function $\mathsf{cs\text{-}normal}(C, t)$ that simplifies term t with respect to the facts in constraint store C. It is not required that cs-normal computes a normal form, even though the name of the function suggests that.

3 Gaussian Elimination for Parametric Systems

Our extension of the Gaussian algorithm to parametric systems is based on constraint programming. Assumptions on parameters are maintained in constraint stores, and a suitable reasoning specialist is used to decide problems in the parameter domain.

The common Gaussian algorithm works as follows. In a given matrix A a non-zero entry a_{ij}, the *pivot*, is selected. Then, using suitable row transformations, all other entries in column j are reduced to zero. Finally, the algorithm is applied recursively to the submatrix obtained by deleting row i and column j, until the remaining submatrix is a 1×1-matrix, or contains only zero-entries.

Our constraint-variant of Gaussian elimination takes both a matrix A with parametric entries and a constraint store C as arguments. The critical modification concerns pivot selection. Any entry p that is not the zero-term is a suitable candidate. If $p \neq 0$ is inconsistent with C then $C \models p = 0$ and p is not suitable as a pivot. If $p = 0$ is inconsistent with C then $C \models p \neq 0$ and p is a suitable pivot. Otherwise, that is both $p = 0$ and $p \neq 0$ are consistent with C, the algorithm branches. On one branch, $p = 0$ is added to the constraint store, and this relation may be used to simpify matrix entries. On the other branch, the assumption $p \neq 0$ is added.

New matrix entries are computed during elimination as sums of products of other entries and are thus polynomial expressions over the entries of the original matrix. The smallest suitable theory for the reasoning specialist is thus the theory of polynomial equations and inequations (if an elimination scheme is used that does not introduce fractions). This theory can be decided with the radical membership test, which is based on Hilbert's Nullstellensatz and Buchberger's algorithm (Cox *et al.*, 1992). This leads to the following implementation of the reasoning specialist, where the constraint store represents a set of polynomials.

- cs-init(): the empty set of polynomials.
- cs-simp$(C, p = 0)$: returns $C \cup \{p\}$.

  cs-simp$(C, p \neq 0)$: returns $C \cup \{py - 1\}$ for a new variable y.
- cs-unsat(C): returns true if and only if 1 is element of the radical ideal generated by C.
- cs-normal(C, t): returns the ideal reduction of t modulo the ideal $\langle C \rangle$ generated by C.

Note that cs-normal could be extended to return a normal form of t with respect to C but this would require to compute the radical ideal generated by C, a fairly expensive operation, which is not necessary for the radical membership test.

4 Strategies for Sparse Parametric Systems

A subset of the parameter space for which a uniform solution to the equation system has been computed is called a *regime*. Sit (1992) has pointed out that the number of regimes computed by Gaussian elimination and branching is exponential to the size of the matrix. Any optimisation therefore must aim at

Class of pivot	$f^{\mathbf{1}}_{ij}$	$f^{\boldsymbol{x}}_{ij}$	$f^{\boldsymbol{x}}_{ij}$
$\mathbf{1}$	$(r^{\mathbf{1}}_i - 1)(c^{\mathbf{1}}_j - 1)$	0	$(m - r^{\mathbf{0}}_i - 1)(n - c^{\mathbf{0}}_j - 1)$ $-(r^{\mathbf{1}}_i - 1)(c^{\mathbf{1}}_j - 1)$
$\boldsymbol{x}$	0	$r^{\mathbf{0}}_i(n - c^{\mathbf{0}}_j - 1)$	$(m - r^{\mathbf{0}}_i - 1)(n - c^{\mathbf{0}}_j - 1)$
$\boldsymbol{X}$	0	0	$(m - 1)(n - c^{\mathbf{0}}_j - 1)$

Table 1: Estimated fill-in depending on the pivot.

reducing the number of regimes. The minimum number of regimes needed to cover the parameter space is small if the number of symbolic matrix entries is small. Again, the number of regimes needed to cover the parameter space may grow exponentially with the number of symbolic entries. Therefore the fundamental assumption for the following optimisations is that the matrix is *symbolically sparse* — that is, only a small number of its entries is symbolic. This assumption is important also because of the expression swell in symbolic Gaussian elimination: because the entries in the matrix tend to become larger, solving large symbolic systems is a hard problem, even if they are not parametric.

4.1 The Markowitz Criterion goes Symbolic

A good strategy to prevent expression swell in a sparse matrix is to select a pivot such that the number of entries that stay zero during elimination is maximal, or, phrased the other way round, that the *fill-in* created by the operation is minimal. A good heuristic, used in numerics for non-symbolic matrices, is to choose a pivot, say a_{ij}, where the number of non-zero entries r_i in the row and the number of non-zero entries c_j in the column are minimal. More precisely, the so-called *Markowitz Criterion* is to choose a non-zero entry for which $(r_i - 1)(c_j - 1)$ is minimal (Duff *et al.*, 1986).

For symbolic matrices the criterion needs to be changed. Our aim is to keep the symbolic fill-in small in order to reduce the likelihood of branching.

Although analogous to Markowitz's criterion, the criterion that we propose is more complex. Four classes of matrix entries are distinguished:

- $\mathbf{0}$: This class consists of the zero element only.
- $\mathbf{1}$: The class of constant, non-zero polynomials.
- $\boldsymbol{x}$: Polynomials that are not constant but known to be constant relative to the current constraint store.
- $\boldsymbol{X}$: All other polynomials.[1]

The estimated fill-in for class c and pivot a_{ij} is denoted by f^c_{ij} and given in Table 1. Here r^c_i denotes the number of entries of class c in row i, and c^c_j denotes the number of entries of class c in row j. The pivot is chosen from an $m \times n$-submatrix.

[1] The definition of classes $\boldsymbol{x}$ and $\boldsymbol{X}$ takes into account that constantness wrt. to the current constraint store may not be decidable or not decidable efficiently.

A pivot a_{ij} is chosen, such that $(f_{ij}^{\boldsymbol{X}}, f_{ij}^{\boldsymbol{x}}, f_{ij}^{\boldsymbol{1}})$ is minimal wrt. lexicographic order. If this does not lead to a unique choice then the product of total degree and number of monomials are compared and the pivot with the smaller value is chosen.

4.2 Deviating from the Gaussian Elimination Scheme

The Gaussian elimination scheme sometimes introduces unnecessary case splits of regimes that cannot be avoided even by a clever pivot selection strategy. Consider the matrix

$$\begin{pmatrix} x & 2-x \\ -1-x & -1+x \end{pmatrix}.$$

All entries are symbolic and any choice of pivot leads to branching. On the other hand, the following sequence of row transformations yields a matrix with no symbolic entries, hence there is a uniform solution of the corresponding equation system for the entire parameter space.

$$\begin{pmatrix} x & 2-x \\ -1-x & -1+x \end{pmatrix} \rightsquigarrow \begin{pmatrix} x & 2-x \\ -1 & 1 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 0 & 2 \\ -1 & 1 \end{pmatrix}$$

The observation that sometimes an arbitrary sequence of row transformations is superior to the Gaussian elimination scheme is exploited by an algorithm we call *Column Simplification* and that is invoked whenever the symbolic Markowitz Criterion fails to select a pivot that is constant, or at least constant with respect to the current context. The algorithm focuses on one column of the matrix — hence its name — and applies a sequence of row transformations that reduce the degrees of the entries in this column.

The basic idea is to successively eliminate leading monomials until no more simplification of the column elements is possible. This is similar to Buchberger's algorithm for computing Gröbner bases. For efficiency first only a sequence of row operations is determined. In a second phase, these transformations are applied to the entire matrix.

4.3 Further Optimisations

Two optimisations concern the simplification of matrix entries after row transformations.

It is possible to divide all entries of a row by their gcd g. If $g \neq 0$ is not entailed by the current context, then g is factored and the reduction is restricted to the factors that can be shown to be non-zero. An alternative strategy is to apply the cs-normal operation provided by the reasoning specialist to all matrix entries.

It is fairly unclear in which situations cs-normal is better than dividing by row gcds and vice versa. In our implementation, division by the row gcd is performed after each row operation whereas cs-normal is only applied immediately after a branch. The latter will be referred to as *Simplify-after-Branch*.

5 Experimental Results

A first step in the evaluation of the proposed method is to measure its performance on equation systems that appear in practice. The algorithm was tested with an implementation in the MuPAD computer algebra system. Despite some effort by the authors to use a state-of-the-art package for Gröbner bases computation, none of these packages could be readily used within MuPAD 2.0.0 on our hardware platform. Instead, MuPAD's own implementation of Gröbner bases was used. Recomputation of Gröbner bases was avoided where possible. All experiments were conducted on a Sparc Ultra 5 with 192 MBytes of main memory.

5.1 The Corpus

Although parametric linear systems arise in the solution of differential equations — for example, when computing the characteristic solutions (Dautray and Lions, 1988) — and also in coloured Petri nets (Jensen, 1996) it turned out that a collection of test data was not available. We were only able to obtain a few symbolic matrices, and their number was insufficient to assess the effectiveness of our method. Instead, a corpus of randomly generated matrices was used.

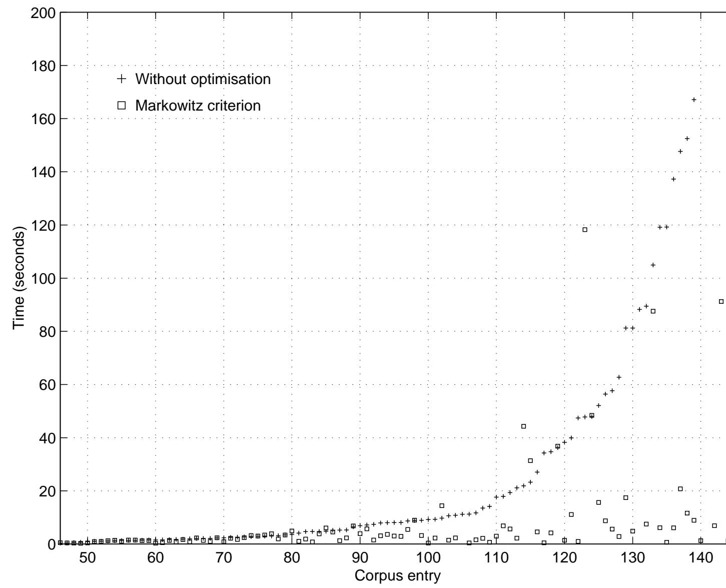
For the experiments a corpus of 540 matrices was randomly generated and then fixed for all experiments. These matrices vary in size from 4×4 to 6×6, in number of parameters from 0 to 3, in the maximum total degree of symbolic entries from 2 to 5, in number of symbolic entries from 0 to 12 and in number of zero entries from 0 to 12. Coefficients and constant entries are evenly distributed from the set $\{-9, -8, \ldots, -1, 1, \ldots, 9\}$. Polynomials of degree k were generated by adding k randomly generated monomials of maximum degree k. More details on the composition of the corpus can be found in Table 3. The entire corpus is available at http://www4.in.tum.de/~ballarin/ in various formats.

5.2 Effectiveness of the Strategies

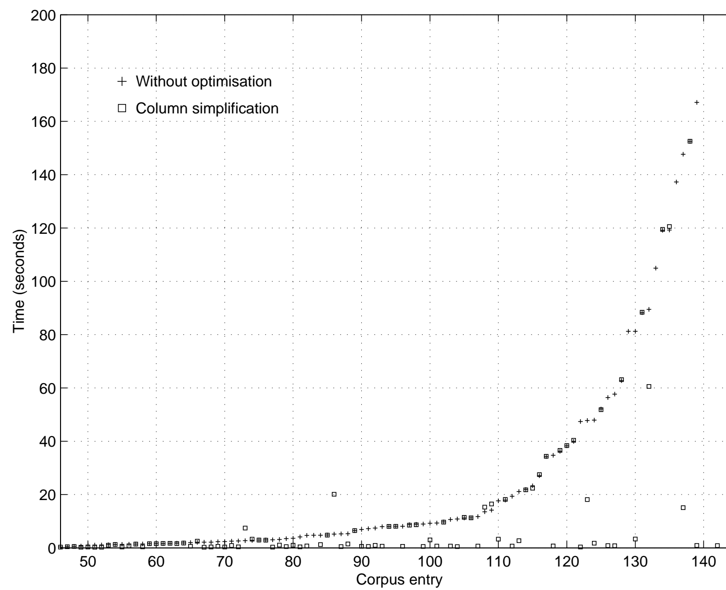
Within a time limit of 450 seconds, 145 matrices of the corpus could be solved without optimisations. This increased to 224 when the combination of all three strategies was used. See Table 3 for the distribution on the corpus' classes. The success rate was even higher when the Markowitz criterion was used as the only optimisation, but then the number of computed regimes is usually higher.

Figures 1 and 2 give more insight in the effect contributed by the strategies. These experiments are based on the 145 corpus elements that could be solved within 450 seconds without use of strategies. Computation of row gcds was always turned on. In the diagrams the 45 matrices without parametric entries are omitted. Corpus elements are sorted by runtime without use of strategies.

The symbolic version of the Markowitz Criterion alone leads to a considerable reduction of computation time. Column Simplification can also lead to dramatic reduction of runtime. Figure 1(b) shows that Column Simplification does either have almost no effect or, when it can be applied, is very effective. It turns out that Column Simplification is most effective in the univariate case because then the likelihood of finding suitable row transformations rather high. Simplify-after-branch itself only rarely leads to a reduction in runtime or regimes. Its main benefit is that it reduces the degree of polynomials describing the regimes.

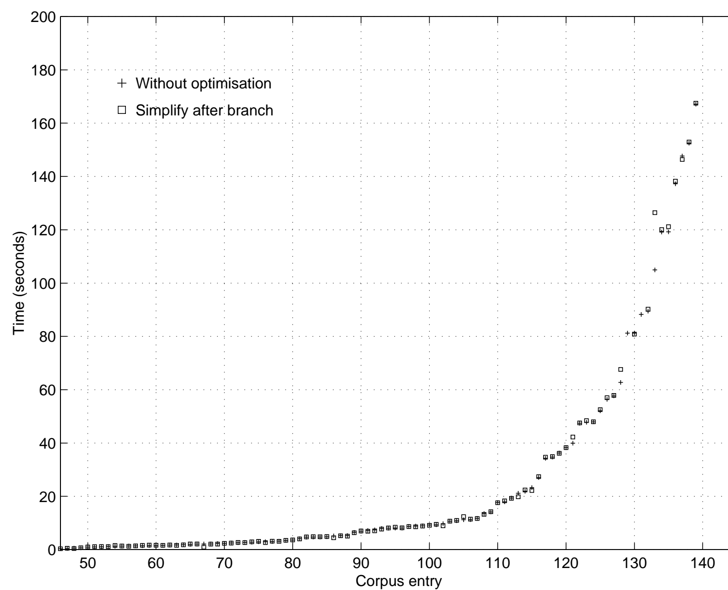


(a) Markowitz Criterion

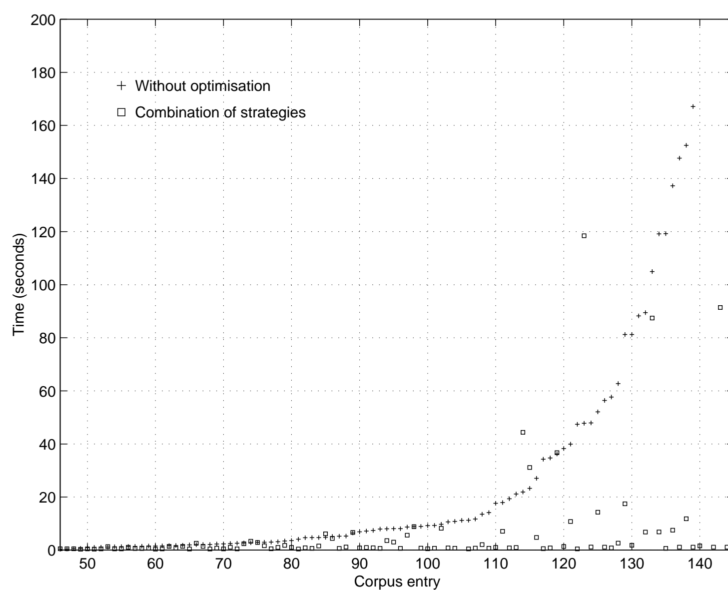


(b) Column Simplification

Figure 1: Speed-up achieved by the strategies. The diagrams show timings for parametric corpus entries that could be solved (without use of strategies) within 450 seconds. Corpus elements are sorted by runtime without use of strategies.

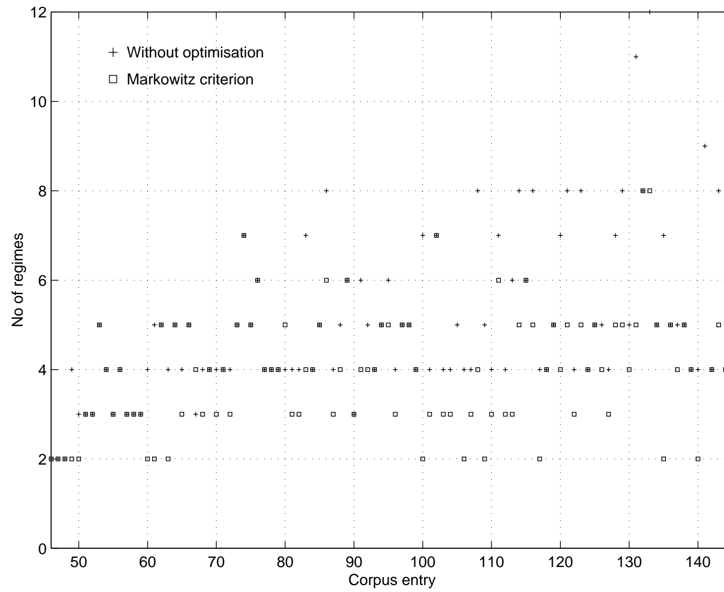


(c) Simplify-after-Branch

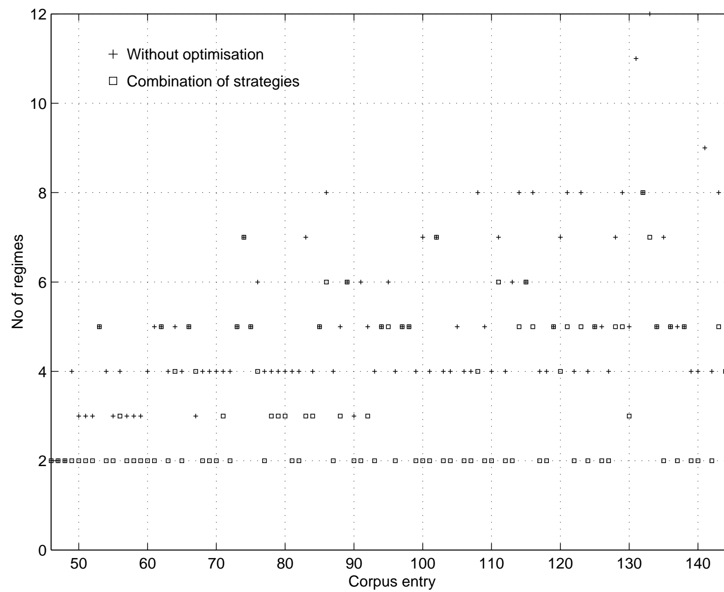


(d) Combination of all strategies

Figure 1 continued: Speed-up achieved by the strategies. The diagrams show timings for parametric corpus entries that could be solved (without use of strategies) within 450 seconds. Corpus elements are sorted by runtime without use of strategies.



(a) Markowitz Criterion



(b) Combination of all strategies

Figure 2: Reduction of regimes achieved by the strategies. Timeout 450 seconds. Corpus elements are sorted by runtime without use of strategies.

	1	2	3	4	5	6	7	8	9
9	·	·	·	·	·	·	·	·	1
8	·	·	·	·	·	·	·	1	·
7	·	·	·	·	1	1	2	·	·
6	·	·	·	1	4	1	1	·	·
5	·	·	3	23	8	1	·	·	·
4	·	·	2	2	·	2	·	·	·
3	·	7	7	2	·	·	·	·	·
2	·	49	8	1	·	·	·	·	·
1	45	·	·	·	·	·	·	·	·

Table 2: Comparision of the number of regimes computed by Sit's algorithm (horizontal axis) and ours (vertical axis). Entries denote number of solved matrices from the corpus.

A close inspection of the data reveals that most gain in runtime is linked to the reduction of regimes achieved by the strategies, and the connected savings in Gröbner basis computations.

5.3 Experimental Comparison with Sit's Algorithm

Sit (1992) has presented an algorithm for parametric linear systems that is based on a different approach. A direct comparison of Sit's and our algorithm does not seem sensible, because the underlying software architectures of MuPAD and Axiom differ greatly. Instead of timings we compare the number of regimes needed to cover the parameter space. The number of regimes is a measure of how adequate the analysis of the parameter space is: the algorithm that returns fewer regimes provides the better analysis. Note that while our algorithm always returns disjoint regimes, Sit's algorithm is not restricted to that.

Table 2 shows the comparision of our algorithm (where all strategies are used) with Sit's algorithm. The comparision is for the 173 matrices of the corpus that could be solved with both our and Sit's algorithm in 450 seconds. The table shows that our algorithm (with all strategies being effective) usually only generates slightly more regimes than Sit's. On the other hand, there are also matrices where our algorithm needs less regimes to cover the parameter space. This underlines the effectiveness of the strategies.

6 Conclusions

Constraint algebraic programming can be applied to parametric linear equation systems successfully. The constraint algebraic programming version of the Gaussian algorithm computes a cover of the parameter space and — for each regime — a solution. The algorithm is also able to compute a cover for only part of the parameter space, if a suitable contraint store is supplied with the matrix.

A feature of contraint algebraic programming is that the base algorithm can be combined with suitable strategies. In the present example, strategies,

Matrix properties					Total	Solved with strategies				
Size	Vars	Deg	Symb	Zero	number	none	m	c	s	all
4	1	2	8	4	20	20	20	20	20	20
5	1	2	10	7	20	17	20	20	17	20
6	1	2	12	12	20	9	19	20	9	20
4	2	2	8	4	20	7	20	7	7	19
5	2	2	10	7	20	2	6	1	1	5
6	2	2	12	12	20	0	4	0	0	3
4	3	2	8	4	20	2	9	3	2	9
5	3	2	10	7	20	0	1	0	0	1
6	3	2	12	12	20	0	0	0	0	0
4	2	2	8	4	15	7	10	5	7	9
5	2	2	10	7	15	2	6	2	2	5
6	2	2	12	12	15	0	2	0	0	2
4	2	3	8	4	15	2	6	2	2	6
5	2	3	10	7	15	0	3	0	0	3
6	2	3	12	12	15	0	0	0	0	0
4	2	4	8	4	15	1	3	1	1	3
5	2	4	10	7	15	0	0	0	0	0
6	2	4	12	12	15	0	0	0	0	0
4	2	5	8	4	15	0	0	0	0	0
5	2	5	10	7	15	0	0	0	0	0
6	2	5	12	12	15	0	0	0	0	0
4	2	2	0	0	15	15	15	15	15	15
5	2	2	0	0	15	15	15	15	15	15
6	2	2	0	0	15	15	15	15	15	15
4	2	2	8	0	15	3	10	2	3	8
5	2	2	10	0	15	0	0	0	0	0
6	2	2	12	0	15	0	0	0	0	0
4	2	2	8	4	15	8	12	8	9	12
5	2	2	10	7	15	0	7	0	0	7
6	2	2	12	12	15	1	1	1	1	1
4	2	2	4	4	15	14	15	13	14	15
5	2	2	7	7	15	5	11	5	4	10
6	2	2	12	12	15	0	3	0	0	1

Table 3: Corpus details. Degree denotes the maximum degree and simultaneously maximal number of monomials of the polynomial entries. Strategies are abbreviated as follows: m Markowitz criterion, c Column Simplification, s Simplify-after-Branch. The corpus is available at http://www4.in.tum.de/~ballarin/.

namely Markowitz Criterion and Column Simplification increase the performance greatly and lead to a favourable comparision to Sit's algorithm — at least, for the corpus used in our experiments.

To the surprise of the authors, it was hard to obtain symbolic matrices that could be used as test data. Although parametric linear equation systems occur naturally in many problems, a collection of such matrices was not available. We chose not to refine our strategies further, because fine tuning only makes sense in the context of "natural" problems. We make our own corpus of randomly generated matrices publicly available in the Internet, but we also would like to encourage others to contribute parametric linear matrices arising in applications. Matrices can be submitted to the first author and will be made publicly available, too.

The greatest advantage of constraint algebraic programming is its flexibility. The above algorithm is not restricted to systems where the parameter domain is polynomials. Any domain for which a reasoning specialist exists is suitable. An extension to parameters involving trigonometric functions over the reals could, for example, not only exploit the relation $\sin^2 x + \cos^2 x = 1$ but also $-1 \leq \sin x, \cos x \leq 1$. The latter would allow chosing $3 - \sin x$ as a pivot without branching.

Acknowledgement

We would like to thank William Sit for providing the source code of his algorithm.

References

Cox, D., Little, J., and O'Shea, D. (1992). *Ideals, Varieties, and Algorithms*. Springer.

Dautray, R. and Lions, J.-L. (1988). *Mathematical Analysis and Numerical Methods for Science and Technology*. Springer.

Davenport, J. H., Siret, Y., and Tournier, E. (1993). *Computer Algebra: Systems and algorithms for algebraic computation*. Academic Press, second edition.

Duff, I., Erisman, A., and Reid, J. (1986). *Direct Methods for Sparse Matrices*. Clarendon Press.

Frühwirth, T. and Abdennadher, S. (1997). *Constraint-Programmierung: Grundlagen und Anwendungen*. Springer-Verlag.

Jensen, K. (1996). *Coloured Petri Nets*. Springer.

Noble, B. and Daniel, J. W. (1988). *Applied linear algebra*. Prentice-Hall, 3rd edition.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM*, **12**(1), 23–41.

Sit, W. (1992). An algorithm for solving parametric linear systems. *Journal of Symbolic Computation*, pages 353–394.

Weibel, T. and Gonnet, G. H. (1993). An assume facility for CAS, with a sample implementation for Maple. In J. Fitch, editor, *Design and implementation of symbolic computation systems: International Symposium, DISCO '92, Bath, U.K., April 13–15 1992: proceedings*, number 721 in LNCS, pages 95–103. Springer-Verlag.