Finite Domain and Cutting Plane Techniques in CLP($\mathcal{PB}$)

Peter Barth Alexander Bockmayr

MPI–I–94–261 November 1994

Authors' Addresses

Peter Barth, Alexander Bockmayr
Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany
{barth,bockmayr}@mpi-sb.mpg.de

Publication Notes

The present report has been submitted for publication elsewhere and will be copyrighted if accepted.

Acknowledgements

This work was supported by the German Ministry for Research and Technology (BMFT) (contract ITS 9103), the ESPRIT Basic Research Project ACCLAIM (contract EP 7195) and the ESPRIT Working Group CCL (contract EP 6028).

Abstract

Finite domain constraints are one of the most important constraint domains in constraint logic programming. Usually, they are solved by local consistency techniques combined with enumeration. We argue that, in many cases, the concept of local consistency is too weak for both theoretical and practical reasons. We show how to obtain more information from a given constraint set by computing cutting planes and how to use this information in constraint solving and constrained optimization. Focusing on the pseudo-Boolean case CLP($\mathcal{PB}$), where all domains are equal to the two-element set $\{0, 1\}$, we present specialized cutting plane techniques and illustrate them on a number of examples.

Contents

1 Introduction

Constraint Logic Programming (CLP) combines the declarativity of logic programming with the computational power of constraint solving over specific domains. Given a constraint solver over a computational domain $\mathcal{X}$, able to decide incrementally the consistency of a constraint set, the CLP($\mathcal{X}$)-scheme [16] provides a semantic base for the corresponding CLP-language. Solving combinatorial optimization problems is one of the main application areas of CLP. Here, the computational domain $\mathcal{X}$ is instantiated to finite domains $\mathcal{FD}$. Since solving finite domain constraints is, in general, an NP-complete problem, the "most important operation on constraints ... a test for consistency" [17] has been relaxed in most systems to local consistency [12, 1, 13, 20, 11, 6]. Global consistency has to be ensured by the programmer, using an extra enumeration predicate. A typical finite domain program has the form

```
problem(<Vars>) :- <state constraints over Vars>,
                   <enumerate domain of Vars>.
```

The search for a feasible solution of the collected constraints is performed by the PROLOG backtracking mechanism, while the finite domain solver is responsible for pruning the search tree.

In this paper, we argue that in many cases the inferences obtained by achieving local consistency are too weak for both theoretical and practical reasons. We propose to use a more powerful inference system, *cutting planes*, which originally have been developed in operations research, but which also have very interesting proof-theoretic properties. The organization of this paper is as follows. We start in Sect. 2 with some semantic problems related to local consistency. In Sect. 3, we introduce cutting planes in a very general way as a complete inference system for linear inequalities in integer variables. In Sect. 4, we present two concrete cutting plane approaches for the pseudo-Boolean case CLP($\mathcal{PB}$), where all variables are restricted to the domain $\{0, 1\}$. In Sect. 5, we explain how cutting planes can improve branch-and-bound algorithms for solving constrained optimization problems. Sect. 6 discusses some advantages of the cutting plane approach from the viewpoint of constraint programming. Finally, Sect. 7 illustrates the effect of cutting planes by a number of empirical results.

2 Local Consistency

We begin our discussion with some semantic problems related to local consistency. One of the most important tasks of any constraint solver is to compute a simplified and concise solved form of a given constraint set. Using only local consistency, this is in general not possible. Consider the query

`?-`$A + B \geq 1, A + (1 - B) \geq 1.$

and assume that all variables are restricted to the domain $\{0, 1\}$. The answer of a CLP-system based on local consistency techniques without extra enumeration predicate is, for example,

`Delayed goals:` $A + B \geq 1$, $A - B \geq 0$ `yes.`

This is correct, since the constraint set is satisfiable, but this is not a simplified and concise solved form.

If we add an extra enumeration predicate `enum`$([A, B])$, then we can enumerate via backtracking the complete solution set $A = 1, B = 0$ and $A = 1, B = 1$. In general, however, there may be an exponential number of solutions, so this enumeration can be done only for very small examples. What we would like to have, is a simple and concise description of the solution set. In this example, the variable A has to be 1 in all solutions, and the variable B is arbitrary. So a *good* answer of a CLP-system to the query `?-`$A + B \geq 1, A + (1 - B) \geq 1$ would be

$A = 1$ `yes.`

If the set of collected constraints is unsatisfiable, new problems arise. Assume again that the variables are restricted to the domain $\{0, 1\}$, and consider the query

`?-`$A + B \geq 1, A + (1 - B) \geq 1, (1 - A) + B \geq 1, (1 - A) + (1 - B) \geq 1.$

The answer of a CLP-system based on local consistency techniques is

`Delayed goals:` $A + B \geq 1, A - B \geq 0, B - A \geq 0, 1 - A - B \geq 0$ `yes`.

Especially the answer `yes` is misleading, since adding an extra enumeration predicate `enum`$([A, B])$ would yield the answer `no`. But, even adding an extra enumeration predicate cannot guarantee that we detect inconsistency in practice. Consider the schematic program

$$\begin{aligned}\texttt{f}([A, B])\ \texttt{:-}\ & A + B \geq 1, A + (1 - B) \geq 1, \\ & (1 - A) + B \geq 1, (1 - A) + (1 - B) \geq 1, \quad \texttt{f}([A, B]).\end{aligned}$$

The query `?-f(`$[A, B]$`),enum(`$[A, B]$`).` will go into an infinite loop. Note that a CLP($\mathcal{FD}$)-system in the sense of [16] would have to answer `no`, since it has to ensure global consistency.

3 Cutting Planes

Finite domain constraint solvers are based on local consistency and constraint propagation. Local consistency can be achieved very quickly. However, it is a rather weak principle. If we want to extract more information from a given constraint set, more powerful techniques are needed. To get an idea of what information can be hidden in a single constraint, consider the inequality

$$4x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 7, \tag{1}$$

in the 0-1 variables $x_1, \ldots, x_5 \in \{0, 1\}$. From (1) we can deduce for example the inequalities

$$\begin{gathered} x_1 + x_2 \geq 1, \quad x_1 + x_3 \geq 1, \quad x_1 + x_4 \geq 1, \\ x_2 + x_3 + x_4 \geq 1, \quad x_2 + x_3 + x_5 \geq 1, \quad x_2 + x_4 + x_5 \geq 1, \end{gathered} \tag{2}$$

but also

$$\begin{gathered} x_1 + x_2 \geq 1, \quad x_1 + x_3 \geq 1, \quad x_1 + x_4 \geq 1, \\ x_1 + x_2 + x_3 + x_4 \geq 2, \\ x_1 + x_2 + x_3 + x_5 \geq 2, \quad x_1 + x_2 + x_4 + x_5 \geq 2. \end{gathered} \tag{3}$$

Each of the systems (2) and (3) has the same set of 0-1 solutions as the original inequality (1). The inequalities in (2) correspond to *clauses* in propositional logic. For example, $x_2 + x_3 + x_4 \geq 1$

can be written as $x_2 \vee x_3 \vee x_4$. The inequalities in (3) are *extended clauses*, i.e. clauses with a right-hand side greater than 1. They contain more information. For example, $x_1+x_2+x_3+x_4 \geq 2$ states that at least 2 of the four variables $x_1, \ldots, x_4$ must take the value 1. Therefore, it implies $x_2 + x_3 + x_4 \geq 1$. The converse does not hold. The inequalities in (3) form the set Π of all *prime extended clauses* of (1), i.e. each extended clause implied by (1) is implied by some extended clause in (3) (see Sect. 4). Still another system, which corresponds to the set Γ of *facet-defining inequalities* for the 0-1 polytope generated by the solutions of (1) (see Sect. 4), is

$$\begin{gathered} x_1 + x_2 \geq 1, \quad x_1 + x_3 \geq 1, \quad x_1 + x_4 \geq 1, \\ 2x_1 + 2x_2 + x_3 + x_4 + x_5 \geq 4, \quad 2x_1 + x_2 + x_3 + x_4 \geq 3, \\ x_1 + x_2 + x_3 + x_5 \geq 2, \quad x_1 + x_2 + x_4 + x_5 \geq 2. \end{gathered} \tag{4}$$

How can we derive all these constraints from the original inequality (1)? Achieving local consistency is not sufficient. The only thing local consistency allows us to do is to fix some variables to the value 0 or 1. If for each variable x_i there exist one solution with $x_i = 0$ and another one with $x_i = 1$, like in the above example, nothing can be inferred, although there are many dependencies between the variables, as illustrated by the inequalities in (2), (3), and (4). To find out these dependencies we need a more powerful inference mechanism.

A general inference system to derive new inequalities from a given set of finite domain constraints, is the *Chvátal-Gomory rounding procedure* [19]. Consider a linear constraint system $Ax \geq b$, with an integer matrix $A \in \mathbb{Z}^{m \times n}$, and an integer vector $b \in \mathbb{Z}^m$. A new constraint $\lceil u^T A \rceil x \geq \lceil u^T b \rceil$ that is satisfied by all non-negative integer solutions of $Ax \geq b$ can be deduced in two steps:

Non-negative linear combination. $u^T Ax \geq u^T b$, with $u \in \mathbb{R}^m_+$.

Rounding. $\lceil u^T A \rceil x \geq \lceil u^T b \rceil$, where $\lceil z \rceil$ is obtained from z by rounding up each component z_i to the smallest integer number greater or equal z_i.

The first step is obvious. Even all real solutions of $Ax \geq b$ satisfy $u^T Ax \geq u^T b$. To justify the second step, note that for $x \geq 0$ rounding up the left-hand side of $u^T Ax \geq u^T b$ gives the weaker inequality $\lceil u^T A \rceil x \geq u^T b$. For all integer vectors $x \in \mathbb{Z}^n$, the left-hand side of this inequality is integer. Therefore, we can round up also the right-hand side and obtain $\lceil u^T A \rceil x \geq \lceil u^T b \rceil$, for all non-negative integer solutions of $Ax \geq b$.

The Chvátal-Gomory procedure has the following geometric interpretation. Each inequality $a^T x \geq \beta$ in the system $Ax \geq b$ defines a closed half space in n-dimensional real space. The system $Ax \geq b$ defines a finite intersection of such half spaces, which is called a *polyhedron* P. The inequality $u^T Ax \geq u^T b$ is *valid* for P, that is P is contained in the associated half space. If we assume that $u^T A$ consists of relatively prime integer numbers, then rounding up the right-hand side $u^T b$ means that we push the hyperplane $u^T Ax = u^T b$ into P until it meets an integer point, which however may lie outside P (see Fig. 1). Since $\lceil u^T A \rceil x \geq \lceil u^T b \rceil$ may cut off a part of the polyhedron P, it is called a *cutting plane*.

The Chvátal-Gomory Procedure is a complete inference systems for linear integer constraints. It was shown by V. Chvátal in 1973 for bounded integers and by A. Schrijver in 1980 for the general case that any linear inequality $c^T x \geq \delta$, with $c \in \mathbb{Z}^n, \delta \in \mathbb{Z}$, that is satisfied by all integer

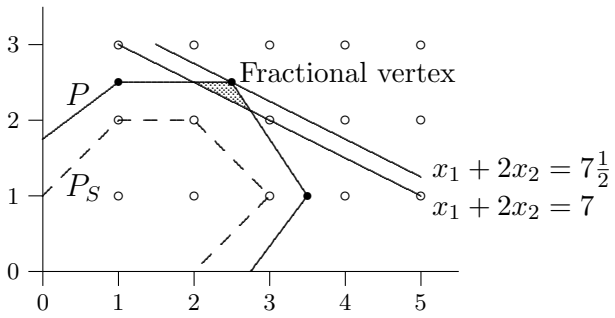


Figure 1: Chvátal-Gomory Cutting Plane

solutions of the system $Ax \geq b$, can be obtained by finitely many applications of the Chvátal-Gomory Procedure. If $Ax \geq b$ has no integer solutions, then we can derive a contradiction $0 \cdot x \geq 1$ [21].

In the given form, the Chvátal-Gomory Procedure is not effective. It is not clear how to choose the weights in the non-negative linear combination of the given constraint matrix. However, several well-known inference rules are subsumed by the Chvátal-Gomory principle. For example, this holds for *resolution* in propositional logic. It is easy to see that any classical resolvent of a set of propositional clauses of the form

$$x_1 \vee \ldots \vee x_m \vee \overline{y}_1 \vee \ldots \vee \overline{y}_k$$

can be obtained as a Chvátal-Gomory cutting plane from the corresponding *clausal inequalities*, which have the form

$$x_1 + \cdots + x_m - y_1 - \cdots\cdots - y_k \geq 1 - k \ ,$$

together with the bounds $0 \leq x_i, y_j \leq 1$. But, Chvátal-Gomory cutting planes are strictly more powerful. For example, the unsatisfiablity of the famous *pigeon-hole problem* (fit n pigeons into $n-1$ holes) can be proved by cutting planes in $O(n^3)$ steps, whereas any resolution proof requires exponentially many steps [10].

4 Cutting Planes in CLP($\mathcal{PB}$)

Using cutting planes, we can make explicit any dependency between the variables in the problem that is expressible by linear inequalities. This, however, is far too general to be practically useful. Therefore, we focus now on a special class of finite domain constraints, for which efficient cutting plane techniques are available. We consider *pseudo-Boolean* constraints, where all domains are equal to the two-element set $\{0, 1\}$. Formally, pseudo-Boolean constraints are defined as equations or inequalities between integer polynomials in 0-1 variables. They arise naturally in many practical applications, in particular in problems from combinatorial optimization. A constraint logic programming language CLP($\mathcal{PB}$) for logic programming with pseudo-Boolean constraints was introduced in [8].

Let C be a set of pseudo-Boolean constraints. Using linearization techniques [4], we can assume that C is given as a system of linear pseudo-Boolean inequalities of the form $C : Ax \geq b,\ x \in \{0,1\}^n$ with $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$. By $P = \{x \in \mathbb{R}^n \mid Ax \geq b, 0 \leq x \leq 1\}$ we denote the *linear relaxation* and by $S = P \cap \{0,1\}^n$ the *0-1 solution set* of C. We will present two cutting plane approaches for

solving C. In both cases, the basic idea is to solve C by approximating an *ideal solved form*, which is either

1. the set of facet-defining inequalities Γ [9] or
2. the set of prime extended clauses Π [4].

Let $P_S = conv(S) \subset \mathbb{R}^n$ denote the convex hull of the 0-1 set S (see Fig. 1). An inequality $c^T x \geq \delta$ is called *facet-defining* for P_S, if it is satisfied by all points in P_S and if moreover there are $\dim(P_S)$ affinely independent points in P_S, for which it is satisfied at equality. Geometrically, this means that $c^T x \geq \delta$ defines a face of maximal dimension of the 0-1 polytope P_S. The *ideal solved form* Γ is the set of all facet-defining inequalities for P_S.

An *extended clause* is an inequality of the form $L_1 + \cdots + L_m \geq d$ (shortly $L \geq d$), where $d \geq 1$ is a positive integer number and $L_i, i = 1, \ldots, m$, is either a positive literal x_i or a negative literal $\overline{x}_i = 1 - x_i$. Intuitively, the clause holds if at least d out of the m literals L_i are true. Classical clauses correspond to the case $d = 1$. An extended clause $L \geq d$ is *prime* for a set $S \subseteq \{0,1\}^n$ if it is satisfied by all points in S and if there is no other extended clause with this property that logically implies $L \geq d$. The *ideal solved form* Π is the set of all prime extended clauses for S.

The two solved forms Γ and Π fit into the requirements imposed in the context of CLP. First, Γ and Π have the same set of 0-1 solutions as the original constraint set C. Second, Γ and Π ease deciding entailment, which is particularly important in the context of concurrent constraint logic programming.

1. A linear pseudo-Boolean inequality $c^T x \geq \delta$ is entailed by C iff $\min\{c^T x \mid x \in S\} \geq \delta$. Since the minimum of $c^T x$ over the polytope $P_S = conv(S)$ is attained in an extreme point $x^* \in S$, we have the basic relationship

$$\min\{c^T x \mid x \in S\} = \min\{c^T x \mid x \in P_S\}. \tag{5}$$

   The first problem is a *discrete* linear optimization problem over the set $S \subseteq \{0,1\}^n$, while the second problem is a *continuous* linear optimization problem over the polytope $P_S \subseteq \mathbb{R}^n$. Given the linear inequality description Γ of P_S, deciding entailment can be reduced to a linear programming problem, which can be solved very efficiently.

2. An extended clause $L \geq d$ is entailed by C if and only if it is entailed by a single extended clause in Π. Deciding entailment between two extended clauses is easy. In fact, $L \geq d$ entails $L' \geq d'$ iff $|L \setminus L'| \leq d - d'$. Here, L and L' are viewed as sets of literals. Reformulation techniques transforming arbitrary linear pseudo-Boolean inequalities into an equivalent set of extended clauses are available [4].

The *ideal* solved forms Γ and Π may involve an exponential number of constraints. Therefore, in general, they cannot be computed in practice. Providing the ideal solved form, however, is not necessary. It suffices to approximate it until satisfiability or entailment can be decided easily. This approximation is done by computing *cutting planes*. At each step, one or several cutting planes are generated and added to the current constraint set, followed by simplification. The cutting planes are chosen in such a way that they lead closer to the ideal solved form. Consider the satisfiability problem for the constraint set C.

1. We start by solving the linear relaxation $P = \{x \in \mathbb{R}^n \mid Ax \geq b, 0 \leq x \leq 1\}$. If P is empty, then C is unsatisfiable. Otherwise, we obtain an extreme point x^* of the polyhedron P. If $x^* \in \{0,1\}^n$, then C is satisfiable. If $x^* \notin \{0,1\}^n$, we generate cutting planes that cut off x^*, but leave invariant the 0-1 solution set S. The set C' obtained from C by adding these cutting planes better approximates Γ, since the associated polyhedron P' is smaller than P. Finding strong cutting planes, ideally facets in Γ, is an active research area in polyhedral combinatorics. Recently developed strong cutting plane algorithms for general 0-1 problems can be embedded into CLP-systems (see Appendix A and [9]).

2. Each extended clause entailed by C is entailed by a single extended clause in Π. To better approximate Π, we look for an extended clause not yet entailed by a constraint in C. We start an implicit enumeration algorithm on C, and suppose that we obtain a failure after fixing all literals in a set L to the value 1. It can be shown that the clause $\overline{L} \geq 1$ is valid, but not yet entailed by any pseudo-Boolean constraint in C. Thus, adding $\overline{L} \geq 1$ to S gives a better approximation of Π, since more extended clauses are entailed. A subsequent simplification procedure ensures that also extended clauses with a right-hand side greater than 1 are generated. The choice of L focuses on the current entailment problem. If the implicit enumeration algorithm finds a feasible 0-1 solution, we know that C is satisfiable. If $0 \geq 1$ is entailed, then C is unsatisfiable (see Appendix B and [4]).

A valid constraint not yet entailed by some constraint in C is also called a *logic cut*. There exist many parallels between logic cuts and the more classical *polyhedral cuts* used in the first approach. Logic cuts approximate the set of prime extended clauses Π, polyhedral cuts the set of facet-defining inequalities Γ. A facet-defining inequality need not be prime, and a prime inequality need not be facet-defining. However, techniques for deriving logic cuts may carry over to the generation of polyhedral cuts and vice versa.

5 Pruning the Enumeration Tree

Many practical applications of finite domain techniques involve the optimization of some objective function subject to a given constraint set. Consider the maximization problem

$$\max\{c^T x \mid Ax \geq b, x \in \mathbb{Z}^n\}, \tag{6}$$

with $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m, c \in \mathbb{Z}^n$. The basic principle applied in finite domain constraint solvers to solve this problem is

- find a feasible solution x^* of the current constraint system
- add the inequality $c^T x \geq c^T x^* + 1$ as a new constraint

until the constraint system becomes infeasible. The last feasible solution that was found is an optimal solution of problem (6).

While feasible solutions yield *lower bounds* for the maximum value of the objective function, *upper bounds* are also very important. First, upper bounds give us information about the quality of a feasible solution even without knowing the optimal solution. If the objective function value

of a feasible solution is close to the upper bound, then it must also be close to the optimum. Often, good feasible solutions can be found very quickly, while finding an optimal solution may take a very long time. The search can be stopped, if we know that only a minor improvement is still possible. Second, upper bounds can be used to discard whole sets of feasible solutions from further consideration. Suppose the set of feasible solutions S has been divided into two subsets S^0 and S^1. If we know that $\max\{c^T x \mid x \in S^0\} \leq u$ and we have found a feasible solution x^* with $c^T x^* > u$, then the optimal solution of the problem must be contained in S^1. To apply this *branch-and-bound* principle, good upper bounds for the objective function are crucial. The bounds obtained by local consistency, however, are very weak. Consider, for example, the constraint $x_1 + \cdots + x_n \leq 1, x \in \{0,1\}^n$, and the objective function $cx_1 + \cdots + cx_n$, with $c \geq 0$. Using local consistency, we can derive only the trivial upper bound nc, although there is the much better bound c.

A more powerful principle to derive upper bounds, widely used in mathematical programming, is to consider the *linear relaxation*

$$\max\{c^T x \mid Ax \geq b, x \in \mathbb{R}^n\}, \tag{7}$$

of the original problem (6). This problem can be solved very efficiently by linear programming techniques like the Simplex algorithm. Obviously, we have $\max\{c^T x \mid Ax \geq b, x \in \mathbb{Z}^n\} \leq \max\{c^T x \mid Ax \geq b, x \in \mathbb{R}^n\}$. Cutting planes tighten the linear relaxation without changing the set of integer solutions. Therefore, they improve the upper bound obtained by solving the linear relaxation and thus reduce the search space of branch-and-bound. The combination of branch-and-bound with cutting plane generation, called *branch-and-cut*, has been extremely successful during the last years. For many hard problems, there is no hope to solve them by pure branch-and-bound. Only when cutting planes are generated, good feasible or even optimal solutions can be obtained. For the notoriously hard traveling salesman problem, instances up to 7397 cities, which corresponds to 27.354.106 0-1 variables, could be solved to optimality using branch-and-cut [2].

6 Global Consistency

Cutting planes can be used in a flexible way. Adding a few cutting planes already can yield a tighter description of the problem that significantly reduces the number of enumeration steps. Generating more cutting planes, we can decide consistency and provide a simplified and concise solved form of the constraint system. The advantages are as follows. From the theoretical point of view, ensuring global consistency gives a semantics in accordance with the original CLP($\mathcal{X}$)-scheme [16]. From the practical point of view of programming with constraints, global consistency allows us to write 'real' CLP-programs that can react to a failure precisely at the point where it occurs. Consider, for example, the CLP-program

```
problem(<Vars>) :- cA(<Vars>),cB(<Vars>),cC(<Vars>).
cA(<Vars>) :- <state constraint over Vars>.
cA(<Vars>) :- <react to failure>.
.... ,
```

where `cB` and `cC` are defined similar to `cA`. Suppose that `<state constraint over Vars>` in `cA` causes global, but not local inconsistency, whereas `<state constraint over Vars>` in `cB` and `cC`

cause no failure. If we ensure global consistency we will detect immediately the failure of the first clause of `cA`, and can try the next clause. In a CLP-system based on local consistency, however, failure will be detected only at the end in the extra enumeration predicate. By backtracking, we will then retry `cC`, which is not intended. The wrong choice will be detected again only at the end, during enumeration. Next `cB` will be retried, and only after one more iteration we will finally retry `cA`. This illustrates how global consistency can enhance the efficiency of the program, and enables the user to program explicitly with the failure of constraints. Another important point is that as long as global consistency is ensured, the execution of a CLP-program does not depend on the underlying constraint solver. Thus, existing applications need not be rewritten when the constraint solving techniques of a CLP-system are enhanced.

It has been argued that ensuring global consistency for finite domain constraints in the context of CLP is not achievable, since this is in general an NP-complete problem. As a consequence, constraint solving has been delayed, and only local consistency is ensured. The advantage of delaying constraint solving is that if all constraints are known, these can be used for pruning the search space. The argument is that ensuring global consistency at each computation step might be very complex, while deciding consistency of the complete constraint set at the end might be trivial. However, we feel that for a large fraction of practical problems this is not the case. In particular, this is true for optimization problems, meanwhile a standard functionality of CLP-systems [24, 17], where we collect a set of constraints and then determine the optimal value of an objective function. Typically, the constraint set is easily shown to be consistent. For many applications the constraints that may cause inconsistency, or that make the problem hard, are known in advance. By reordering the collection of constraints such that these hard constraints are added at the end, we will spend effort at the same point where the extra enumeration predicate would. Promising experience in this direction has been made with incremental enumeration algorithms for clausal satisfiability problems that ensure global consistency [15]. Furthermore, almost all typical CLP-programs in systems with incomplete solvers employ a determinate collection of the constraints, since inconsistency of the current constraint set cannot be used to guide the computation. Typically, indeterminacy is only employed in the extralogical enumeration predicate. A specialized CLP-compiler can extract these determinate constraint collections and the whole set of accumulated constraints can be given at once to the underlying complete solver. Hence, for typical CLP-programs written for incomplete constraint solvers, there is no loss in efficiency due to hard intermediate problems. On the other hand, the full power of a complete solver is available whenever a 'real' CLP-program has to be executed.

7 Computational Experience

7.1 Integer Programming

The utility of cutting plane generation has been widely recognized in the mathematical programming community. For example, cutting planes are used in the mixed integer programming systems MINTO [18], MIPO [3], MOPS [22] as well as in the commercial systems OSL and CPLEX 3.0. For the new release CPLEX 3.0, we compared pure branch-and-bound with branch-and-cut. As a set of benchmarks, we used the pure 0-1 problems in the mixed integer programming library MIPLIB [7], except the very difficult problems air04, air05, p0548, p2756, and p6000. All experiments were

Table 1: **MIPLIB: Branch-and-bound vs. branch-and-cut**

CPLEX 3.0			Branch & Bound		Branch & Cut		
Name	#C	#V	Nodes	Time	Nodes	Cuts	Time
bm23	20	27	452	1.88	590	0/0/59	5.80
l152lav	97	1989	24732	2415.20	7226	0/0/70	976.32
lp4l	85	1086	69	4.27	297	66/0/2	37.68
lseu	28	89	15889	82.05	1932	17/0/31	14.95
misc01	54	83	721	6.97	439	10/7/16	5.95
misc02	39	59	60	0.60	44	8/3/0	0.55
misc03	96	160	527	16.77	428	13/6/0	15.40
misc07	212	260	22832	1450.48	11854	17/14/0	858.70
mod010	146	2655	1065	109.75	292	58/0/0	42.33
p0033	16	33	964	2.23	82	5/0/21	0.47
p0040	23	40	54	0.17	0	0/0/1	0.07
p0201	133	201	1023	25.83	986	6/0/62	45.25
p0282	241	282	> 100000	> 2000.00	749	269/40/122	39.65
p0291[a]	252	291	> 100000	> 2000.00	40	413/14/40	5.02
pipex	25	48	3452	12.08	337	0/0/62	3.02
sentoy	30	60	723	4.92	1129	0/0/65	19.97

[a] No presolving

done on a SPARC 10/31. The integer optimizer of CPLEX 3.0 generates only two special classes of cutting planes, clique cuts and cover cuts, which are not applicable to all problems. Other, more sophisticated cutting plane techniques, like those of MIPO (see Appendix A), may lead to further improvements. Table 1 contains those problems for which there exists a significant difference between pure branch-and-bound and branch-and-cut with CPLEX 3.0. The columns #C and #V indicate the number of constraints and 0-1 variables in the problem. The next two columns give the number of nodes and the running time (in seconds) of the pure branch-and-bound variant of CPLEX 3.0 (option: covers = cliques = –1), which already is known to be very efficient. The last three columns give the number of nodes, the number of generated clique cuts/applied clique cuts/applied cover cuts, and the running time (in seconds) of the branch-and-cut variant of CPLEX 3.0 (option: covers = cliques = 1). In most cases, branch-and-cut is better than branch-and-bound, in particular for problems with a large number of nodes. In some of the smaller examples, the number of nodes increases when cutting planes are generated, due to the branching heuristics of CPLEX 3.0.

7.2 Propositional Satisfiability

The clausal satisfiability problem of propositional logic (SAT) is fundamental to a large number of practical problems. It has become a standard for the evaluation of constraint solvers. We show on two examples how cutting plane methods can be applied successfully to notoriously hard instances of the SAT-problem.

First, we consider the well-known pigeon-hole problems. By a restricted application of the di-

Table 2: **Combinatorial SAT-problems**

Name	Vars	Clauses	CPLEX	Simp	SCPLEX
ulmbc060	60	610	3.22	5.40	0.35
ulmbc084	84	994	1.62	12.90	0.40
ulmbp096*	96	244	364.23	1.28	0.05
ulmbp126*	126	351	>1000	2.30	0.07
ulmbp160*	160	485	>1000	4.18	0.10
ulmbp198*	198	649	>1000	7.21	0.12
ulmbs084	84	189	6.70	0.78	0.03
ulmbs112	112	280	89.70	1.50	0.07
hole6*	42	133	25.02	0.38	0.02
hole7*	56	204	306.90	0.87	0.03
hole8*	72	297	>1000	1.71	0.02
hole10*	110	561	>1000	5.86	0.08
hole20*	420	4221	>1000	291.30	0.10
nod5col4*	40	130	2.20	0.70	0.22
nod6col4*	60	255	12.68	2.21	0.05
nod7col4*	84	441	86.98	6.25	0.12
nod7col6*	126	651	>1000	11.91	797.42

agonal sum rule (see Appendix B), we are able to construct *automatically* a cutting plane proof similar to the one found by Cook et al. [10]. This specialized cut generation technique has been incorporated into a logic cut based constraint solver [4]. In combination with a linear programming system, we obtain a constraint solver that detects unsatisfiability of the pigeon hole problems in polynomial time [5]. In Table 2, we report computational results for solving some satisfiability problems with CPLEX (Branch & Bound) and for solving the simplified problem after cut generation. In the column 'CPLEX' we report the running time (in seconds) for solving the original problem with CPLEX. In column 'Simp' we report the time used by a simple Prolog-prototype for the logic cut generation and in 'SCPLEX' the running time of CPLEX for solving the preprocessed problem. We also included some other examples in order to demonstrate the wider applicability of the approach.

Second, we consider some of the "aim..." problems, artificially generated instances of the SAT problem with at most 3 literals per clause, found in the DIMACS-collection [23]. The names give more information about the problem, e.g. aim-100-1_6-no-1 means that the problem is unsatisfiable and that it contains 100 variables and 160 clauses. We solved the problems with a prototype implementation of a logic cut based pseudo-Boolean constraint solver that generates logic cuts, as presented on page 8, and additionally includes simplifications steps based on the deductive system "generalized resolution" (see Appendix B), which further strengthen the derived logic cuts [5, 4]. We report the number of logic cuts generated with "generalized resolution" (GRCuts), the number of logic cuts generated by enumeration (ECuts) and the running time (Time) needed to solve the problem. We compare the results with SATO [25], a state-of-the-art implementation of the implicit enumeration Davis-Putnam procedure, where we abort the search after the first solution of the problem has been found or all branches have been explored. We report the number of evaluated

Table 3: **"aim..." Problems**

Name	DPNodes	DPTime	ECuts	GRCuts	Time
aim-100-1_6-no-1	521776459	56812.13	70	131	39.43
aim-100-1_6-no-2	324665079	31308.94	15	33	7.50
aim-100-1_6-no-3	212014823	35305.32	168	65	113.18
aim-100-1_6-no-4	531054239	55406.34	88	122	55.70
aim-100-1_6-yes1-1	16459005	1782.93	0	22	0.55
aim-100-1_6-yes1-2	53	0.0	0	20	0.53
aim-100-1_6-yes1-3	42288874	4860.4	41	12	15.51
aim-100-1_6-yes1-4	53778746	5677.95	0	10	2.80
aim-100-2_0-no-1	92442475	12880.38	6	13	7.80
aim-100-2_0-no-2	53797042	6675.17	7	18	8.48
aim-100-2_0-no-3	24070150	2786.52	32	38	27.61
aim-100-2_0-no-4	268969301	31555.01	64	26	53.10
aim-100-2_0-yes1-1	271346	24.8	75	108	51.98
aim-100-2_0-yes1-2	32300	3.66	3	80	7.51
aim-100-2_0-yes1-3	24876	2.86	39	26	17.75
aim-100-2_0-yes1-4	166373	24.99	1	96	7.16
aim-100-3_4-yes1-1	10	0.01	7	67	24.30
aim-100-3_4-yes1-2	77	0.02	7	122	43.10
aim-100-3_4-yes1-3	121	0.03	3	131	25.41
aim-100-3_4-yes1-4	915	0.17	5	97	26.58

nodes (DPNodes) and the running time (DPTime).

We selected only some of the problems in order to demonstrate the applicability of the presented logic cut method. The smaller problem instances (50 variables) are solved in about the same time by both methods, but almost always fewer cuts are generated than nodes visited. Most hard larger instances (200 variables) cannot be solved by SATO in reasonable time. The logic cut method was able to solve many of the hard larger instances within 1000 seconds running time. The final state of the logic cut based method for the satisfiable instances consists of the single extended clause $L \geq |L|$, where L is the set of all literals that need to be fixed to 1. Thus, the information that the satisfiable instances have a unique solution is determined by the solver. An implicit enumeration procedure, like SATO, would have to enumerate completely the remaining search space for detecting this information.

References

[1] A. Aggoun, D. Chan, P. Dufresne, E. Falvey, H. Grant, A. Herold, G. Macartney, M. Meier, D. Miller, B. Perez, E. van Rossum, J. Schimpf, P. A. Tsahageas, and D. H. de Villeneuve. ECLIPSE 3.4, ECRC Common Logic Programming System. Technical report, ECRC, Munich, July 1994.

[2] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding cuts in the TSP (A preliminary report). Distributed at the Mathematical Programming Symposium, Ann Arbor, August 1994.

[3] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58:295 – 324, 1993.

[4] P. Barth. *Logic-based 0-1 constraint solving in constraint logic programming.* PhD thesis, Fachbereich Informatik, Univ. des Saarlandes, 1994. Forthcoming.

[5] P. Barth. Simplifying clausal satisfiability problems. In *Constraints in Computational Logic CCL'94, Munich*, pages 19–33. Springer, LNCS 845, 1994.

[6] F. Benhamou, D. McAllester, and P. van Hentenryck. CLP(Intervals) revisited. Technical Report CS-94-18, Brown Univ., April 1994.

[7] R. E. Bixby, E. A. Boyd, and R. Indovina. MIPLIB: A test set for mixed integer programming problems. *SIAM News 25*, page 16, 1992. ftp: softlib.rice.edu.

[8] A. Bockmayr. Logic programming with pseudo-Boolean constraints. In F. Benhamou and A. Colmerauer, editors, *Constraint Logic Programming. Selected Research*, chapter 18, pages 327 – 350. MIT Press, 1993.

[9] A. Bockmayr. Cutting planes in constraint logic programming. Technical Report MPI-I-94-207, Max-Planck-Institut für Informatik, Saarbrücken, February 1994.

[10] W. Cook, C. R. Coullard, and Gy. Turán. On the complexity of cutting plane proofs. *Discrete Applied Mathematics*, 18:25 – 38, 1987.

[11] D. Diaz and P. Codognet. A minimal extension of the WAM for clp(FD). In *Proc. 10th Intern. Conf. Logic Programming, Budapest*, 1993.

[12] M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, and T. Graf. The constraint logic programming language CHIP. In *Fifth Generation Computer Systems, Tokyo, 1988.* Springer, 1988.

[13] W. S. Havens, S. Sidebottom, G. Sidebottom, J. Jones, and R. Ovans. Echidna: a constraint logic programming shell. In *Pacific Rim Int. Conf. Artificial Intelligence, Seoul, Korea*, pages 165 – 171, 1992.

[14] J. N. Hooker. Generalized resolution and cutting planes. *Annals of Operations Research*, 12:217 – 239, 1988.

[15] J. N. Hooker. Solving the incremental satisfiability problem. *Journal of Logic Programming*, 15:177 – 186, 1993.

[16] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proc. 14th ACM Symp. Principles of Programming Languages*, Munich, 1987.

[17] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 1994.

[18] G. Nemhauser, M. W. P. Savelsbergh, and G. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15:47 – 58, 1994.

[19] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley, 1988.

[20] W. Older and F. Benhamou. Programming in CLP(BNR). In *Principles and Practice of Constraint Programming PPCP'93, Newport, RI*, 1993.

[21] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley, 1986.

[22] U. Suhl. MOPS - Mathematical OPtimization System. *Europ. J. Oper. Res.*, 72:312 – 322, 1994.

[23] M. Trick. The second DIMACS international algorithm implementation challenge: Clique, graph coloring, and satisfiability, 1993. ftp: dimacs.rutgers.edu.

[24] P. van Hentenryck. *Constraint satisfaction in logic programming.* MIT Press, 1989.

[25] H. Zhang. SATO: A decision procedure for propositional logic. *Association of Automated Reasoning Newsletters*, 22:1–3, March 1993.

A Lift-and-Project Cutting Planes

A complete polyhedral cutting plane method for linear pseudo-Boolean constraints is the *lift-and-project* approach [3]. Consider a polyhedron $P = \{x \in \mathbb{R}^n \mid Ax \geq b, 0 \leq x \leq 1\} = \{x \in \mathbb{R}^n \mid \tilde{A}x \geq \tilde{b}\}$, with $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^n$, and a fractional vertex x^* of P. For any $j \in \{1, \ldots, n\}$ with $0 < x_j^* < 1$ a *lift-and-project cutting plane* $\alpha x \geq \beta$ that cuts off x^* can be obtained by solving a linear program of the form

$$\max\{\beta - x^*\alpha \mid (\alpha, \beta) \in R_j^*(P) \cap T\}.$$

Here, $R_j^*(P)$ is the polyhedral cone consisting of those $(\alpha, \beta) \in \mathbb{R}^{n+1}$ for which there exist vectors $u, v \in \mathbb{R}^{m+2n}$ and $u_0, v_0 \in \mathbb{R}$ satisfying

$$\begin{array}{rrrrrl} \alpha & -u\tilde{A} & -u_0 e_j & & & = 0 \\ \alpha & & & -v\tilde{A} & -v_0 e_j & = 0 \\ & u\tilde{b} & & & & = \beta \\ & & & v\tilde{b} & +v_0 & = \beta \\ & & & & u, v & \geq 0, \end{array}$$

and e_j is the j-th unit vector in $\mathbb{R}^n$. The set T is a normalization that truncates the cone $R_j^*(P)$, for example $T = \{(\alpha, \beta) \mid \sum_{i=1}^n |\alpha_i| \leq 1\}$.

Given a system of linear pseudo-Boolean constraints $Ax \geq b, x \in \{0,1\}^n$, the lift-and-project method yields in finitely many steps either a 0-1 solution or detects that the problem is unsatisfiable [9].

B Generalized Resolution

A complete inference system for deriving the set of prime extended clauses for a set of extended clauses E is *generalized resolution* [14, 5], which is based on the following rules.

Resolution: $E \vdash E \cup \{R\}$ if there exist classical clauses C_1 and C_2 each implied by some extended clause in E such that R is a classical resolvent of C_1 and C_2 and R is not implied by some extended clause in E.

Diagonal Sum: $E \vdash E \cup \{DS\}$ if there exist $m - d + 1$ out of the m extended clauses

$$\begin{array}{cccccc} \square & L_2 & +L_3 & +\cdots & +L_{m-1} & +L_m & \geq d \\ L_1 & \square & +L_3 & +\cdots & +L_{m-1} & +L_m & \geq d \\ \vdots & & & & & & \\ L_1 & +L_2 & +L_3 & +\cdots & +L_{m-1} & \square & \geq d \end{array}$$

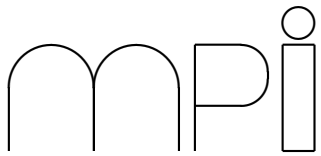
that are implied by some extended clause in E such that DS is the extended clause

$$L_1 \quad +L_2 \quad +L_3 \quad +\cdots \quad +L_{m-1} \quad +L_m \quad \geq d+1$$

and DS is not implied by some extended clause in E.
Here, the symbol $\square$ is used to indicate a missing literal.

Simplification: $E \uplus \{C\} \vdash E$ iff C is implied by some extended clause in E.

Given a set of extended clauses E, applying these rules as long as possible, yields the set of prime extended clauses for the 0-1 solution set S of E [14]. Note that both, resolvents and diagonal sums, are special Chvátal-Gomory cutting planes.



Below you find a list of the most recent technical reports of the research group *Logic of Programming* at the Max-Planck-Institut für Informatik. They are available by anonymous ftp from our ftp server `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Regina Kraemer
Im Stadtwald
D-66123 Saarbrücken
GERMANY
e-mail: `kraemer@mpi-sb.mpg.de`

MPI-I-94-246	M. Hanus	On Extra Variables in (Equational) Logic Programming
MPI-I-94-241	J. Hopf	Genetic Algorithms within the Framework of Evolutionary Computation: Proceedings of the KI-94 Workshop
MPI-I-94-240	P. Madden	Recursive Program Optimization Through Inductive Synthesis Proof Transformation
MPI-I-94-239	P. Madden, I. Green	A General Technique for Automatically Optimizing Programs Through the Use of Proof Plans
MPI-I-94-238	P. Madden	Formal Methods for Automated Program Improvement
MPI-I-94-235	D. A. Plaisted	Ordered Semantic Hyper-Linking
MPI-I-94-234	S. Matthews, A. K. Simpson	Reflection using the derivability conditions
MPI-I-94-233	D. A. Plaisted	The Search Efficiency of Theorem Proving Strategies: An Analytical Comparison
MPI-I-94-232	D. A. Plaisted	An Abstract Program Generation Logic
MPI-I-94-230	H. J. Ohlbach	Temporal Logic: Proceedings of the ICTL Workshop
MPI-I-94-229	Y. Dimopoulos	Classical Methods in Nonmonotonic Reasoning
MPI-I-94-228	H. J. Ohlbach	Computer Support for the Development and Investigation of Logics
MPI-I-94-226	H. J. Ohlbach, D. Gabbay, D. Plaisted	Killer Transformations
MPI-I-94-225	H. J. Ohlbach	Synthesizing Semantics for Extensions of Propositional Logic
MPI-I-94-224	H. Aït-Kaci, M. Hanus, J. J. M. Navarro	Integration of Declarative Paradigms: Proceedings of the ICLP'94 Post-Conference Workshop Santa Margherita Ligure, Italy
MPI-I-94-223	D. M. Gabbay	LDS – Labelled Deductive Systems: Volume 1 — Foundations
MPI-I-94-218	D. A. Basin	Logic Frameworks for Logic Programs
MPI-I-94-216	P. Barth	Linear 0-1 Inequalities and Extended Clauses
MPI-I-94-209	D. A. Basin, T. Walsh	Termination Orderings for Rippling