Cutting Planes in Constraint Logic Programming

Alexander Bockmayr

MPI–I–94–207 February 1994

Author's Address

Alexander Bockmayr, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, 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 benefited from discussions with P. Barth and S. Ceria. It 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

In this paper, we show how recently developed techniques from combinatorial optimization can be embedded into constraint logic programming. We develop a constraint solver for the constraint logic programming language CLP($\mathcal{PB}$) for logic programming with pseudo-Boolean constraints. Our approach is based on the generation of polyhedral cutting planes and the concept of branch-and-cut. In the case of 0-1 constraints, this can improve or replace the finite domain techniques used in existing constraint logic programming systems.

Contents

1 Introduction

Constraint logic programming has been one of the major developments in declarative programming in recent years [JM94]. The idea is to combine logic programming with constraint solving over some domain of computation such as linear arithmetic, Boolean algebra, finite domains or lists. Various constraint logic programming systems have been developed so far, for example Prolog III [Col87], CLP($\mathcal{R}$) [JL87], CHIP [DvHS$^+$88], and CAL [ASS$^+$88]. Constraint logic programming has been very successful in many practical applications, in particular in problems from artificial intelligence and operations research. The combination of logic programming and constraint solving has benefits for both sides. From the viewpoint of logic programming, constraint solving techniques enhance the expressive power and the efficiency of logic programs. Efficient algorithms from mathematics, artificial intelligence or operations research can be embedded directly into the logic programming language and need not be coded on the level of unification and resolution using primitive system predicates. From the viewpoint of constraint solving, a high-level declarative programming language is available in addition to a pure constraint solver. This means that the problem to be solved can be represented in a natural and declarative way. The logic language not only can be used to generate the constraint formulas. It also allows us to handle those problem features that do not fit into the given constraint solving framework.

The general idea of constraint solving is to compute a solved form which represents all the solutions of a given constraint set. In many practical applications, however, one is interested in a solution which is optimal with respect to some objective function. Although optimization is not part of the original constraint logic programming framework, it is present in many existing constraint logic programming systems. In this paper, we show how recently developed techniques from combinatorial optimization can be embedded into constraint logic programming. We develop a constraint solver for our constraint logic programming language CLP($\mathcal{PB}$) for logic programming with pseudo-Boolean constraints. Originally, pseudo-Boolean constraints were seen as a generalization of Boolean constraints that combines Boolean algebra with a restricted form of arithmetic [Boc93]. Accordingly, our first approach to handle pseudo-Boolean constraints was to extend Boolean unification to the pseudo-Boolean case. In this paper, we describe a new constraint solver for linear pseudo-Boolean constraints which is based on cutting plane techniques from polyhedral combinatorics. Concerning 0-1 constraints, the generation of cutting planes and the concept of branch-and-cut can improve or replace the finite domain techniques and the branch-and-bound approach adopted in existing constraint logic programming systems.

The organization of the paper is as follows. In Section 2, we recall briefly some basic concepts of constraint logic programming and the language CLP($\mathcal{PB}$) for logic programming with pseudo-Boolean constraints. We also present the main problems which have to be solved by a constraint solver for this language. In Section 3, we discuss how a solved form for pseudo-Boolean constraints should look like. We define a solved form which is based on the notion of strong valid inequalities for the convex hull of the 0-1 solution set. A constraint set C is simplified by computing polyhedral cutting planes for the linear relaxation of C. In Section 4, we consider the question how this solved form can be computed in practice. To generate strong cutting planes, we propose to use the new lift-and-project method [BCC93b, BCC93a] recently developed in the context of mixed 0-1 optimization. On the one hand, this method has very nice theoretical properties. In particular, we can prove a correctness and completeness theorem for the constraint solver. On the other hand, the lift-and-project algorithm has also produced a number of impressive results on hard practical problems. In Section 5, we show how the cutting plane algorithm can be embedded into the more efficient branch-and-cut framework, without loosing the basic properties required in the context of constraint solving. Finally, in Section 6, we discuss briefly a possible extension of this work to the case of mixed 0-1 constraints.

2 Constraint Logic Programming with 0-1 Constraints

In its classical form [JL87], a constraint logic program over an algebra $\mathcal{A}$ consists of a finite set of rules of the form

$$p_0(\vec{t_0}) :- c_1(\vec{u}_1), \ldots, c_m(\vec{u}_m), p_1(\vec{t_1}), \ldots, p_n(\vec{t_n})$$

where p_i are predicate symbols different from the relation symbols in $\mathcal{A}$, c_j are symbols denoting relations in $\mathcal{A}$, and $\vec{t_i}, \vec{u}_j$ are tuples of terms of $\mathcal{A}$ with additional undefined function symbols. The declarative semantics of such a rule is that the head $p_0(\vec{t_0})$ is logically implied by the conjunction $c_1(\vec{u}_1) \wedge \ldots \wedge c_m(\vec{u}_m) \wedge p_1(\vec{t_1}) \wedge \ldots \wedge p_n(\vec{t_n})$ of all *constraints* $c_j(\vec{u}_j)$ and atoms $p_i(\vec{t_i})$ in the body.

Given a constraint logic program one has to choose a goal, which has the same form as the body of a rule. The operational semantics is based on two components: a constraint solver for $\mathcal{A}$-relations and an adaptation of the resolution method from classical logic programming. For example, from the goal $?- \ c(t_1), p(t_2)$ and the rule $p(t_3) :- d(t_4), q(t_5)$ we may derive the new goal $?- \ \mathcal{S}(c(t_1), t_2 \doteq t_3, d(t_4)), q(t_5)$ provided that the constraint $c(t_1), t_2 \doteq t_3, d(t_4)$ is solvable in $\mathcal{A}$ and $\mathcal{S}(c(t_1), t_2 \doteq t_3, d(t_4))$ is its *solved form.* A derivation sequence consists of goals with solvable constraints. It is successful when the last goal contains only constraints. These are called answer constraints and constitute the output of the program. A derivation sequence is finitely failed if the last goal cannot be expanded.

In [Boc93] we introduced a new constraint logic programming language CLP($\mathcal{PB}$) for logic programming with pseudo-Boolean constraints. The language CLP($\mathcal{PB}$) is an instance of the constraint logic programming language scheme CLP($\mathcal{X}$) [JL87]. *Pseudo-Boolean constraints* or *0-1 constraints* combine Boolean algebra with arithmetic. A *pseudo-Boolean function* is an integer-valued function $f : \{0,1\}^n \to \mathbb{Z}$ of 0-1 variables [HR68] and thus generalizes the notion of a Boolean function $f : \{0,1\}^n \to \{0,1\}$. Any such function can be represented as a multilinear polynomial in its variables. A *pseudo-Boolean constraint* is an equation or inequality between pseudo-Boolean functions. A simple example is the inequality $3x_1 + 2x_2 + x_3 + x_4 \leq 4$ which has the solutions (0,0,0,0), (0,0,0,1), (0,0,1,0), (0,0,1,1), (0,1,0,0), (0,1,0,1), (0,1,1,0), (0,1,1,1), (1,0,0,0), (1,0,0,1), (1,0,1,0). Any Boolean constraint can be translated into a pseudo-Boolean constraint using the identities

$$\neg x = 1 - x, \ x \wedge y = x * y, \ x \vee y = x + y - x * y.$$

The following example illustrates some typical aspects of logic programming with pseudo-Boolean constraints.

Example 2.1 Suppose we want to compute the Hamming distance of two 0-1 vectors $x, y \in \{0,1\}^n$. A naive program for solving this problem is

```
    hamming1([], [], 0).
```

```
hamming1([X|Xs],[Y|Ys], N) :-
      X = Y, hamming1(Xs,Ys,N).
hamming1([X|Xs],[Y|Ys], N + 1) :-
      X = 1 - Y, hamming1(Xs,Ys,N).
```

Here N is a real or integer variable and X, Y are 0-1 variables. If we ask the query `?- hamming1([A,B,C],[0,1,0],2)`, backtracking will generate the answers

```
A = 0, B = 0, C = 1;
A = 1, B = 1, C = 1;
A = 1, B = 0, C = 0;
No.
```

To avoid enumerating a possibly exponential number of 0-1 tuples we may use the program

```
hamming2([], [], 0).
hamming2([X|Xs], [Y|Ys], N + (X + Y - 2*X*Y)) :-
      hamming2(Xs,Ys,N).
```

Note that $X + Y - 2 * X * Y$ evaluates to 0 if $X = Y$ and to 1 if $X \neq Y$. Linearization of the nonlinear term $X * Y$ using a new 0-1 variable Z yields

```
hamming3([], [], 0).
hamming3([X|Xs], [Y|Ys], N + (X + Y - 2*Z)) :-
      Z <= X, Z <= Y, X + Y <= 1 + Z,
      hamming3(Xs,Ys,N).
```

Here, the pseudo-Boolean constraints $Z \leq X, Z \leq Y, X + Y \leq 1 + Z$ express that $Z = X * Y$. Now consider again the query `?- hamming3([A,B,C],[0,1,0],2)`. A successful derivation is

```
?- hamming3([A,B,C],[0,1,0],2).
?- X = A, Xs = [B,C], Y = 0, Ys = [1,0], N + (X + Y - 2*Z) = 2,
   Z <= X, Z <= Y, X + Y <= 1 + Z, hamming3([B,C],[1,0],N).
?- X = A, Xs = [B,C], Y = 0, Ys = [1,0], N + (X + Y - 2*Z) = 2,
   Z <= X, Z <= Y, X + Y <= 1 + Z,
   X1 = B, X1s = [C], Y1 = 1, Y1s = [0], N1 + (X1 + Y1 - 2*Z1) = N,
   Z1 <= X1, Z1 <= Y1, X1 + Y1 <= 1 + Z1, hamming3([C],[0],N1).
?- X = A, Xs = [B,C], Y = 0, Ys = [1,0], N + (X + Y - 2*Z) = 2,
   Z <= X, Z <= Y, X + Y <= 1 + Z,
   X1 = B, X1s = [C], Y1 = 1, Y1s = [0], N1 + (X1 + Y1 - 2*Z1) = N,
   Z1 <= X1, Z1 <= Y1, X1 + Y1 <= 1 + Z1,
   X2 = C, X2s = [], Y2 = 0, Y2s = [], N2 + (X2 + Y2 - 2*Z2) = N1,
   Z2 <= X2, Z2 <= Y2, X2 + Y2 <= 1 + Z2, hamming3([],[],N2).
?- X = A, Xs = [B,C], Y = 0, Ys = [1,0], N + (X + Y - 2*Z) = 2,
   Z <= X, Z <= Y, X + Y <= 1 + Z,
   X1 = B, X1s = [C], Y1 = 1, Y1s = [0], N1 + (X1 + Y1 - 2*Z1) = N,
   Z1 <= X1, Z1 <= Y1, X1 + Y1 <= 1 + Z1,
   X2 = C, X2s = [], Y2 = 0, Y2s = [], N2 + (X2 + Y2 - 2*Z2) = N1,
   Z2 <= X2, Z2 <= Y2, X2 + Y2 <= 1 + Z2, N2 = 0.
```

which can be simplified to `A = 1 + B - C`. This example shows how new variables and constraints are generated during program execution and how symbolic answers to a given query can be obtained.

One of the most important features of the language CLP($\mathcal{PB}$) is that it allows us to handle combinatorial optimization problems within the constraint logic programming framework [Boc93].

Example 2.2 Consider for example a knapsack problem. Suppose there is a vessel with capacity w and goods g_i with weight w_i and value v_i for $i = 1, \ldots, n$. We introduce 0-1 variables X_i which indicate whether or not g_i is loaded on the vessel. The possible cargos not exceeding the capacity can be determined by a constraint rule involving a pseudo-Boolean constraint.

```
cargo($X_1, \ldots, X_n$) :-
        $w_1 * X_1 + \cdots + w_n * X_n \leq w$.
```

If we want to find a most valuable cargo, this can be done using a meta-predicate `max` that maximizes a pseudo-Boolean function subject to some pseudo-Boolean constraints.

```
most-valuable-cargo($X_1, \ldots, X_n$) :-
        max($v_1 * X_1 + \cdots + v_n * X_n$, cargo($X_1, \ldots, X_n$)).
```

Other applications of CLP($\mathcal{PB}$), in particular in artificial intelligence, are described in [BB93].

2.1 Main problems

In [Boc93] we showed how Boolean unification [MN89] which is a standard approach for solving Boolean constraints can be generalized to the pseudo-Boolean case. In particular, we gave variable elimination algorithms for pseudo-Boolean unification and unconstrained pseudo-Boolean optimization. Both algorithms generalize the well-known Boolean unification algorithm of [BS87].

The aim of this paper is to present a new constraint solver for *linear* pseudo-Boolean constraints of the form $a_1x_1 + \cdots + a_nx_n \leq b$, with $a_1, \ldots, a_n, b \in \mathbb{Z}$ and 0-1 variables $x_1, \ldots, x_n$.

The constraint solver has to provide solutions to the following typical problems in constraint logic programming:

Solvability: Decide whether a constraint set C has a solution.

Simplification: Simplify a constraint set C to a *solved form* $\mathcal{S}(C)$.

Entailment: Decide whether a constraint c is implied by a constraint set C.

Since in every resolution step new constraints are added to the current constraint set, the constraint solver has to be *incremental*.

In many applications, as is illustrated for example by the knapsack problem given before, it is not sufficient to compute an arbitrary solution of the constraint set C but one wishes to find a solution which is *optimal* with respect to some pseudo-Boolean function f [Boc93, BB93]. Therefore we have also the problem

Optimization: Optimize a pseudo-Boolean function f subject to a constraint set C.

The problem of optimization has recently received a lot of attention within the constraint logic programming community (see e.g. [Fag93, MT93]).

2.2 Pseudo-Boolean and finite domain constraints

Pseudo-Boolean constraints generalize Boolean constraints. At the same time, they are a special form of finite domain constraints (see for example [Mac92]). In principle, we could apply finite domain techniques to solve pseudo-Boolean constraints. Finite domain constraints have been used very successfully in constraint logic programming [vH89]. However, these methods, which are based on local consistency techniques for constraint satisfaction, cannot exploit the special structure of 0-1 problems. Another problem is that standard finite domain constraint solvers are not complete. A set of constraints may be locally consistent although it does not admit a global solution. In order to achieve completeness the values in the domains have to be enumerated by a backtracking mechanism (`labeling` procedure).

Our approach to solve 0-1 constraints is based on polyhedral cutting plane techniques from operations research. In general, polyhedral cutting planes are used in combination with traditional branch-and-bound, which is then called *branch-and-cut*. The idea, however, is to avoid branching as much as possible. The generation of polyhedral cutting planes may reduce the search space for branch-and-bound algorithms in a dramatic way. There exist non-trivial examples which could be solved even without any branching [Boy93]. For many hard combinatorial optimization problems, branch-and-cut algorithms yield the best methods currently available in order to find optimal or at least provably good solutions. The most prominent example is the traveling salesman problem [PR91, JRT92] where instances with up to 3000 cities, which roughly corresponds to 4500000 0-1 variables, could be solved in a reasonable time.

3 Solving 0-1 Constraints

When executing a constraint logic program, the main problem is to simplify a given constraint set C and to compute a *solved form* $\mathcal{S}(C)$. Our first question is how this solved form should look like. There are the following basic requirements:

- $\mathcal{S}(C)$ should be equivalent to C, i.e. C and $\mathcal{S}(C)$ should have the same set of 0-1 solutions.
- $\mathcal{S}(C)$ should be "simpler" than C.
- $\mathcal{S}(C)$ should be *false* if C is unsolvable.

3.1 Equality descriptions of the solution set

The most natural approach to solve a set of 0-1 constraints is to compute *families of solutions* like

X_1	X_2	X_3	X_4	X_5
0	0	–	–	–
0	1	1	–	–
0	1	0	1	–
1	0	1	–	–

Logically, this corresponds to a *disjunction* of conjunctions of equalities. In logic programming, which is based on Horn logic, disjunction is normally provided by the possibility of defining a predicate by more than one rule. Backtracking is used to apply these rules sequentially top-down

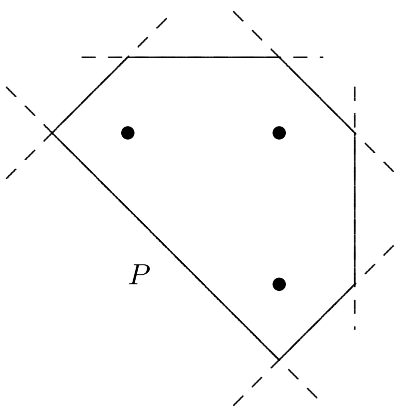

Figure 1: Polyhedron P defined by six linear inequalities containing three 0-1 points

from left-to-right in order to find a solution to the given goal. Disjunction in the solved form of a constraint set would mean that we introduce an additional level of indeterminism during program execution. Backtracking would have to occur also within the constraint solving procedure. This leads to an enormous blow-up of the search space which we want to avoid.

A second approach is therefore to compute a parametric solution or a *most general pseudo-Boolean unifier* [Boc93]

$$\begin{array}{c} X_1 \doteq t_1[Y_1, \ldots, Y_k] \\ \vdots \\ X_n \doteq t_n[Y_1, \ldots, Y_k], \end{array} \tag{1}$$

where $t_1, \ldots, t_n$ are pseudo-Boolean terms containing new variables $Y_1, \ldots, Y_k$. This corresponds to one of the most common approaches in Boolean constraint solving [BS87]. The advantage of this approach is that we capture all the solutions of the constraint set by a conjunctive formula which in addition can be seen as an idempotent substitution. However, practical experience already in the Boolean case has shown that these most general unifiers involve very complex pseudo-Boolean terms. Therefore, they seem to be useful only for special problem classes, for example in circuit verification [SND88].

3.2 Inequality descriptions

Instead of computing an explicit representation of the solution set using equalities we may also compute an implicit representation based on *inequalities.* As we will see, this is particularly useful in the context of optimization because it allows us to use powerful methods from operations research.

From a mathematical point of view, a set C of linear 0-1 constraints is the same as a system of linear inequalities in 0-1 variables

$$\begin{array}{cccc} a_{11}x_1 & + \cdots + & a_{1n}x_n & \leq b_1 \\ \vdots & & \vdots & \vdots \\ a_{m1}x_1 & + \cdots + & a_{mn}x_n & \leq b_m \end{array} \Leftrightarrow Ax \leq b, \tag{2}$$

with $A \in \mathbb{Z}^{m \times n}$, $x \in \{0,1\}^n$ and $b \in \mathbb{Z}^m$. Such a system defines a possibly empty set S of 0-1 points in $\mathbb{R}^n$. If we drop the condition $x \in \{0,1\}^n$ then the system of linear 0-1 constraints (2) becomes a system of *linear arithmetic constraints* over the real numbers, which defines a *polyhedron*

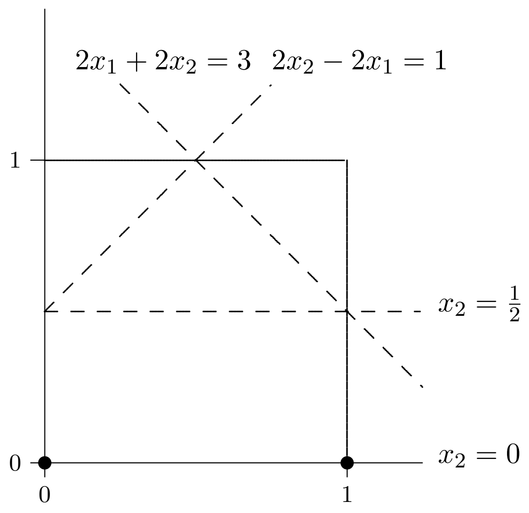

Figure 2: Linear inequality descriptions of $S = \{(0,0),(1,0)\}$

P in $\mathbb{R}^n$ (see Fig. 1). The set of 0-1 solutions $S = P \cap \{0,1\}^n$ corresponds to the set of 0-1 points lying within P.

There is an important difference between linear inequality descriptions of the real polyhedron P and the 0-1 set S. Let us call a system $Ax \le b$ over $\mathbb{R}^n$ (resp. $\{0,1\}^n$) *irredundant* if no constraint $\alpha x \le \beta$ can be removed without changing the solution set P (resp. S). For a full-dimensional polyhedron P there is up to multiplication of inequalities by a positive real number a unique representation of P by an irredundant system of inequalities $Ax \le b$. The inequalities in $Ax \le b$ are in a 1-1 correspondence with the facets of P [Sch86]. Remember that an inequality defines a *facet* of P if it is satisfied by all points in P and if moreover there are n affinely independent points in P for which it is satisfied at equality.

For a 0-1 set S, however, there are many different representations by irredundant systems $Ax \le b$. For example, the irredundant constraint sets $\{x_2 \le 0\}$, $\{2x_2 \le 1\}$, and $\{2x_1 + 2x_2 \le 3, 2x_2 - 2x_1 \le 1\}$ all define the same 0-1 set $S = \{(0,0),(1,0)\} \subseteq \mathbb{R}^2$ (see Fig. 2).

If we use inequalities then the solved form of a constraint system $C : Ax \le b$ is again a system of inequalities $\bar{C} : \bar{A}x \le \bar{b}$. Since we want to simplify our problem, $\bar{C}$ should be "simpler" than C. However, it is not clear what this should mean. There are many possibilities to compare linear inequality descriptions $Ax \le b$ of a 0-1 set S. In particular we may consider

- the size of the system $Ax \le b$ depending on the number of inequalities and the size of their coefficients.
- the strength of the inequalities in the system $Ax \le b$.

3.3 Size of the representation

Given a linear inequality in 0-1 variables, there are many other inequalities having the same set S of 0-1 solutions. The set S^* of all $(\alpha,\beta) \in \mathbb{R}^{n+1}$ such that $\alpha x \le \beta$ has the 0-1 solution set S defines a polyhedron in $\mathbb{R}^{n+1}$. To find an equivalent inequality with minimal coefficients one can solve a linear optimization problem as described in [BHW74].

Concerning the size of the coefficients, a recent result of Schmitt [Sch92] (improving a classical theorem from threshold logic [Mur71]) says that any linear 0-1 inequality $\alpha x \le \beta, (\alpha, \beta) \in \mathbb{R}^{n+1}$, is equivalent to a 0-1 inequality $\gamma_1 x_1 + \cdots + \gamma_n x_n \le \delta$ with integer coefficients $\gamma_1, \ldots, \gamma_n, \delta$ such that

$$|\gamma_i| \le 2^{-n}(n+1)^{(n+1)/2} \quad \text{and} \quad |\delta| \le 2^{-(n+1)}(n+1)^{(n+3)/2} + 1/2.$$

By a result of Jeroslow [Jer75], at most 2^{n-1} linear 0-1 inequalities are needed to define a 0-1 set $S \subseteq \{0,1\}^n$. For any k in the range $1 \le k \le 2^{n-1}$, there exists $S \subseteq \{0,1\}^n$, such that at least k linear 0-1 inequalities are needed to define S. Lipkin [Lip87] showed that in order to define a set $S \subseteq \{0,1\}^n$ of cardinality m, at most m linear 0-1 inequalities are needed. The problem of deciding whether an arbitrary 0-1 set S can be defined by a single linear 0-1 inequality is NP-complete [PS85].

3.4 Strength of the representation

Linear inequality descriptions $Ax \le b$ of a 0-1 set S can also be compared with respect to the strength of the inequalities that are used. When comparing the strength of linear inequalities we have to specify in which kind of solutions we are interested.

Definition 3.1 Let $Q, R \subseteq \mathbb{R}^n$. An inequality $\alpha x \le \beta$ is *valid* for Q iff $\alpha x_0 \le \beta$, for all $x_0 \in Q$. $\alpha x \le \beta$ is *stronger than* $\gamma x \le \delta$ *with respect to* R iff $\{x \in R \mid \alpha x \le \beta\} \subseteq \{x \in R \mid \gamma x \le \delta\}$. The inequality is *strictly stronger* iff the inclusion is strict.

If $\{x \in \mathbb{R}^n_+ \mid \alpha x \le \beta\} \ne \emptyset$, then $\alpha x \le \beta$ is stronger than $\gamma x \le \delta$ with respect to $\mathbb{R}^n_+$ iff there exists $\lambda \ge 0$ such that $\lambda \alpha_i \ge \gamma_i$, for $i = 1, \ldots, n$ and $\lambda \beta \le \delta$.

A facet-defining inequality of a polyhedron $P \subseteq \mathbb{R}^n$ is a strongest valid inequality for P with respect to $\mathbb{R}^n$. A strongest valid inequality for a 0-1 set S with respect to $\{0,1\}^n$ is called a *prime inequality* [Hoo92]. Note that these two notions of strongest valid inequality are not equivalent. A facet-defining inequality of the convex hull $conv(S)$ of S need not be a prime inequality for S and vice versa.

Example 3.2 Let $S = \{(1,0,0), (1,0,1), (0,1,1), (1,1,0), (1,1,1)\}$. The inequality $x_1 + x_2 \ge 1$ defines a facet of $conv(S)$ but it is not prime. The inequality $2x_1 + x_2 + x_3 \ge 2$, which is valid for S, is stronger than $x_1 + x_2 \ge 1$ with respect to $\{0,1\}^3$ because it is not satisfied by the point $(0,1,0)$. However, $2x_1 + x_2 + x_3 \ge 2$ does not define a facet of $conv(S)$.

The notion of prime inequality can also be defined relative to a given class T of linear 0-1 inequalities. A *prime inequality for S relative to T* is a strongest valid inequality for S in the class T with respect to $\{0,1\}^n$.

3.5 Defining a solved form

Unfortunately, the various possibilities that we have considered so far to compare two constraint sets C_1 and C_2 are not compatible.

Example 3.3 The constraint sets

$$C_1 = \{3x_1 + 2x_2 + x_3 + x_4 + x_5 \geq 5\} \tag{3}$$

and

$$\begin{aligned} C_2 = \{ \quad & x_1 + x_2 \geq 1, \quad x_1 + x_3 \geq 1, \quad x_1 + x_4 \geq 1, \quad x_1 + x_5 \geq 1, \\ & x_1 + x_2 + x_3 + x_4 \geq 2, \quad x_1 + x_2 + x_3 + x_5 \geq 2, \quad x_1 + x_2 + x_4 + x_5 \geq 2 \quad \} \end{aligned} \tag{4}$$

have the same 0-1 solution set S.

The first description contains only one inequality, but some coefficients of the left-hand side are different from 0 and 1. The second description consists of seven *extended clauses*, that is inequalities of the form $L_1 + \cdots + L_m \geq k$, which express that at least k out of m literals $L_1, \ldots, L_m$ have to be true [Hoo92].

In [Bar93a, Bar93b], a constraint solver for 0-1 constraints is presented that computes for a given linear 0-1 inequality an equivalent set of *prime extended clauses*. Given a set of prime extended clauses $\pi(S)$ for a 0-1 set S, testing solvability and entailment becomes very easy. The set $\pi(S)$, however, may be very large. The members of $\pi(S)$ are strongest valid inequalities for S with respect to $\{0,1\}^n$ (relative to the class T of extended clauses), but they need not be strongest valid inequalities for the convex hull $conv(S)$ with respect to $\mathbb{R}^n$.

Example 3.4 The extended clause $\{y_1 + y_2 + y_3 + y_4 \geq 2\}$ is equivalent to the set of classical clauses $\{y_1 + y_2 + y_3 \geq 1, y_1 + y_2 + y_4 \geq 1, y_1 + y_3 + y_4 \geq 1, y_2 + y_3 + y_4 \geq 1\}$. Therefore another description of S by classical clauses would be

$$\begin{aligned} C_3 = \{ \quad & x_1 + x_2 \geq 1, \quad x_1 + x_3 \geq 1, \quad x_1 + x_4 \geq 1, \quad x_1 + x_5 \geq 1, \\ & x_2 + x_3 + x_4 \geq 1, \quad x_2 + x_3 + x_5 \geq 1, \quad x_2 + x_4 + x_5 \geq 1 \quad \}. \end{aligned} \tag{5}$$

While all these representations define the same 0-1 set S, their strength is quite different. The set of all nontrivial facets of $conv(S)$ is

$$\begin{aligned} C_4 = \{ \quad & x_1 + x_2 \geq 1, \quad x_1 + x_3 \geq 1, \quad x_1 + x_4 \geq 1, \quad x_1 + x_5 \geq 1, \\ & 2x_1 + x_2 + x_3 + x_4 \geq 3, \quad 2x_1 + x_2 + x_3 + x_5 \geq 3, \quad 2x_1 + x_2 + x_4 + x_5 \geq 3, \\ & 3x_1 + 2x_2 + x_3 + x_4 + x_5 \geq 5 \quad \}. \end{aligned} \tag{6}$$

For example, the facet-defining inequality $2x_1 + x_2 + x_3 + x_4 \geq 3$ is strictly stronger than the prime extended clause $x_1 + x_2 + x_3 + x_4 \geq 2$, which again is strictly stronger than than the classical clause $x_2 + x_3 + x_4 \geq 1$ (with respect to both $\{0,1\}^n$ and $[0,1]^n$). The inequality $3x_1 + 2x_2 + x_3 + x_4 + x_5 \geq 5$ is strictly stronger than $2x_1 + x_2 + x_3 + x_4 \geq 3$ with respect to $\{0,1\}^n$, but not with respect to $[0,1]^n$. Note that a set C of facet-defining inequalities for $conv(S)$ need not contain all facets of $conv(S)$ in order to define the 0-1 set S, consider for example $C = \{3x_1 + 2x_2 + x_3 + x_4 + x_5 \geq 5\}$.

Given all these possibilities, how should we define the solved form? In order to answer this question we have to take into account one more point that we have not considered so far. Since we are interested not only in constraint solving but also in constrained optimization we need a solved form that supports optimization.

One of the most powerful techniques to obtain provably good solutions of combinatorial optimization problems is *linear optimization* combined with the generation of *polyhedral cutting planes* [HP85]. A discrete optimization problem over the 0-1 set S

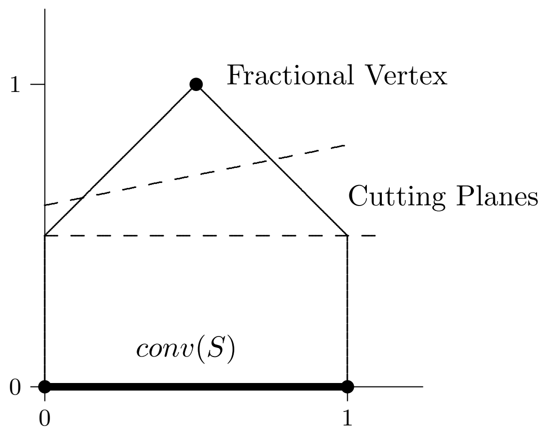

Figure 3: Approximating the convex hull of S by cutting planes

$$\text{Maximize } z = cx \text{ subject to } x \in S = \{x \in \{0,1\}^n \mid Ax \leq b\}$$

is equivalent to the linear optimization problem

$$\text{Maximize } z = cx \text{ subject to } x \in conv(S) \subseteq \mathbb{R}^n$$

over the convex hull $conv(S)$ of S. This means that 0-1 optimization problems can be solved by linear optimization techniques provided that a linear inequality description of $conv(S)$ is known.

First we should point out that it is not necessary to have a complete description of $conv(S)$ which might be very complex [KP82]. It is enough to have a sufficiently good approximation of $conv(S)$, which can be computed by the generation of *cutting planes*. A first approximation P of $conv(S)$ is obtained by the *linear relaxation* $Ax \leq b, 0 \leq x \leq 1$ of the original problem. Then new inequalities are added which are satisfied by all points in S but which cut off at least one fractional vertex of P. These are called *cutting planes*. This is repeated until a 0-1 vertex is obtained, which then automatically belongs to S (see Fig. 3).

It is crucial for this approach that the cutting planes which are generated are strong with respect to $\mathbb{R}^n$ (cf. Definition 3.1). In the best case, they should define facets of $conv(S)$ or at least faces of sufficiently high dimension. Traditional *Chvàtal-Gomory-Cuts* [Gom58] do not have this property. They converge much too slowly in order to be practically useful. In order to compute strong cutting planes with a reasonable effort, advanced techniques from *polyhedral combinatorics* are necessary, which will be described in the next section. Our paradigm for solving 0-1 constraints is therefore the following:

> A constraint set C is solved by computing polyhedral cutting planes for the convex hull $conv(S)$, where S denotes the set of 0-1 solutions of C.

This is also reflected in the next definition.

Definition 3.5 Given two constraint sets $C : Ax \leq b$ and $C' : A'x \leq b'$ with the same set of 0-1 solutions S, we say that C is *simpler* than C' iff $\{x \in \mathbb{R}^n \mid Ax \leq b\} \subseteq \{x \in \mathbb{R}^n \mid A'x \leq b'\}$.

The *ideal solved form* of C is a constraint set $\bar{C} : \bar{A}x \leq \bar{b}$ with $conv(S) = \{x \in \mathbb{R}^n \mid \bar{A}x \leq \bar{b}\}$ such that there is a 1-1 correspondence between the inequalities in $\bar{A}x \leq \bar{b}$ and the facets of $conv(S)$.

In practice, this ideal solved form cannot be computed, because an exponential number of inequalities may be needed. However, the idea is not to compute the ideal solved form. On the contrary, we want to avoid the exponential blow-up as much as possible. Therefore constraint solving is done in a *lazy* way. We compute only as many cutting planes as are needed in order to answer the basic questions presented in Section 2.1. The solved form that is actually computed is defined implicitly by the cut generation process.

4 Computing the Solved Form

In the previous section, we have defined a solved form for a given set of 0-1 constraints. The main question now is how to compute this solved form. In other words the problem is to find strong cutting planes for a given set of 0-1 constraints. For special problems (knapsack, set covering, traveling salesman etc.) strong valid inequalities have been found by analyzing the specific problem structure. For general 0-1 problems, strong valid inequalities are very hard to obtain. An early strong cutting plane approach for general 0-1 problems was presented in [CJP83]. The cutting planes used there are facet-defining inequalities for the knapsack polytope generated by an individual 0-1 constraint. This approach has been particularly successful in the case of large-scale 0-1 problems with a sparse coefficient matrix and with no apparent special structure. In this case, the constraints do not interact very much, so that one can expect that strong valid inequalities for the knapsack polytope of an individual 0-1 constraint will also be strong for the 0-1 polytope of the full problem. The mixed integer optimizer MINTO [SN93] is based on these ideas. Another interesting recent approach for solving hard 0-1 programs, which uses knapsack and enumeration cutting planes combined with preprocessing, is described in [Boy93].

In our context, however, the most promising way to find strong cutting planes for general 0-1 problems, seems to be the new *lift-and-project* method for mixed 0-1 optimization due to [BCC93b, BCC93a]. This method not only has very nice theoretical properties, such as finite convergence. It has also been applied very successfully in practice. The approach was able to solve several previously unsolved problems and in many cases seems to obtain better results than other procedures available like CPLEXMIP 2.0 or OSL 2.0 [Cer93]. In several cases, the procedure even outperformed specialized algorithms on the class of problems for which these algorithms were designed. Extensive empirical results documenting this can be found in [BCC93b, BCC93a, Cer93].

The starting point of the lift-and-project method is a *sequential convexification* theorem. Suppose

$$P = \{x \in \mathbb{R}^n \mid Ax \geq b, 0 \leq x \leq 1\} \overset{\text{def}}{=} \{x \in \mathbb{R}^n \mid \tilde{A}x \geq \tilde{b}\}, \tag{7}$$

with $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$, is a polyhedron in $\mathbb{R}^n$. If

$$R_j(P) = conv(P \cap \{x \in \mathbb{R}^n \mid x_j \in \{0,1\}\}), \qquad \text{for } j = 1, \ldots, n, \tag{8}$$

is the convex hull of those points in P for which $x_j = 0$ or $x_j = 1$, then

$$R_1(R_2(\ldots R_n(P))) = conv(S), \tag{9}$$

where as usual $S = P \cap \{0,1\}^n$.

The idea of [BCC93b, BCC93a] is to use facets of $R_j(P)$ as strong cutting planes for S. By lifting the problem into a higher-dimensional space one can show that

$$R_j(P) = \{x \in \mathbb{R}^n \mid \alpha x \geq \beta \text{ for all } (\alpha, \beta) \in R_j^*(P)\}$$

where $R_j^*(P)$ is the set 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

$$\begin{array}{cccccr} \alpha & -u\tilde{A} & -u_0 e_j & & & = 0 \\ \alpha & & & -v\tilde{A} & -v_0 e_j & = 0 \\ & u\tilde{b} & & & & = \beta \\ & & & v\tilde{b} & +v_0 & = \beta \\ & & & & u, v & \geq 0, \end{array} \tag{10}$$

where e_j is the j-th unit vector in $\mathbb{R}^n$.

If P is full-dimensional, $P \cap \{x \mid x_j = 0\} \neq \emptyset$ and $P \cap \{x \mid x_j = 1\} \neq \emptyset$, then for any constant $\beta \neq 0$, $\alpha x \geq \beta$ defines a facet of $R_j(P)$ iff (α, β) is an extreme ray of $R_j^*(P)$. To compute such extreme rays one can solve a linear program of the form

$$max\{a\alpha + b\beta \mid (\alpha, \beta) \in R_j^*(P) \cap T\} \tag{11}$$

where $(a, b) \in \mathbb{R}^{n+1}$ is a vector that determines the direction of the cut and T a normalization set that truncates the cone $R_j^*(P)$. This linear program has roughly twice the size of the current problem. Several choices of (a, b) and T are possible. Usually, one uses $(a, b) = (-x, 1)$, where x is the fractional vertex of P which should be cut off, and $T = \{(\alpha, \beta) \mid -1 \leq \alpha_i \leq 1, i = 1, \ldots, n\}$ or $T = \{(\alpha, \beta) \mid \sum_{i=1}^{n} |\alpha_i| \leq 1\}$. In some cases, one can also choose the truncation $\beta = 1$ or $\beta = -1$.

Example 4.1 Consider the set of 0-1 constraints

$$C = \{-2x_1 - 2x_2 \geq -3,\ 2x_1 - 2x_2 \geq -1\}.$$

The linear relaxation P is given by the system of linear inequalities $\tilde{A}x \geq \tilde{b}$, where

$$\tilde{A}^T = \begin{pmatrix} -2 & 2 & 1 & 0 & -1 & 0 \\ -2 & -2 & 0 & 1 & 0 & -1 \end{pmatrix}$$

and

$$\tilde{b}^T = \begin{pmatrix} -3 & -1 & 0 & 0 & -1 & -1 \end{pmatrix}.$$

To cut off the fractional vertex $(\frac{1}{2}, 1)$ of P we solve the linear program

Maximize $-\frac{1}{2}\alpha_1 - \alpha_2 + \beta$ subject to

$$\begin{array}{cccccccccr} \alpha_1 & & +2u_1 & -2u_2 & -u_3 & & +u_5 & & -u_0 & = 0 \\ & \alpha_2 & +2u_1 & +2u_2 & & -u_4 & & +u_6 & & = 0 \\ \alpha_1 & & +2v_1 & -2v_2 & -v_3 & & +v_5 & & -v_0 & = 0 \\ & \alpha_2 & +2v_1 & +2v_2 & & -v_4 & & +v_6 & & = 0 \\ & & -3u_1 & -u_2 & & & -u_5 & -u_6 & & = \beta \\ & & -3v_1 & -v_2 & & & -v_5 & -v_6 & +v_0 & = \beta \end{array}$$

$$-1 \leq \alpha_1, \alpha_2 \leq 1$$
$$u_1, u_2, u_3, u_4, u_5, u_6, v_1, v_2, v_3, v_4, v_5, v_6 \geq 0.$$

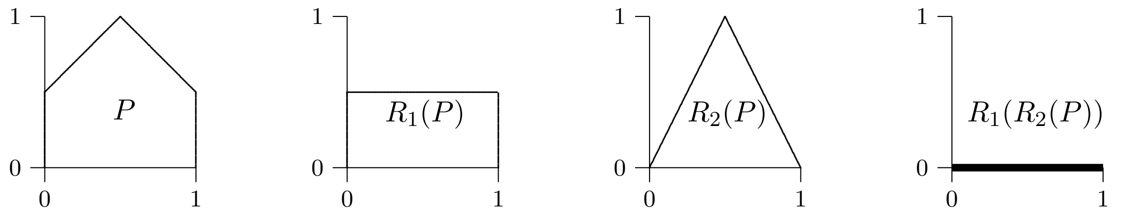

Figure 4: Sequential convexification for $P = \{x \in [0,1]^2 \mid -2x_1 - 2x_2 \geq -3,\ 2x_1 - 2x_2 \geq -1\}$

We get the optimal solution $\alpha_1 = 0, \alpha_2 = -1, \beta = -\frac{1}{2}$, which corresponds to the facet $x_2 \leq \frac{1}{2}$ of $R_1(P)$. In the same way, we can compute the facets $x_1 + \frac{1}{2}x_2 \leq 1$ and $x_1 - \frac{1}{2}x_2 \geq 0$ of $R_2(P)$, which cut off the vertices $(1, \frac{1}{2})$ and $(0, \frac{1}{2})$ respectively (see Fig. 4). In this example, we really compute facets of $R_j(P)$. Note however that, due to the truncation T, this need not always be the case.

We give now our strong cutting plane algorithm for solving 0-1 constraints. Implicitly, it computes a solved form of a constraint set C.

Strong Cutting Plane Algorithm for Solving 0-1 Constraints

```
Initialization: Let $t := 1$; $C^1 := C = \{Ax \geq b, 0 \leq x \leq 1\}$,
Iteration t: 1. Solution of the linear relaxation: Optimize a linear function $g^t$
        over the relaxation $P^t = \{x \in \mathbb{R}^n \mid x$ is a solution of $C^t\}$.
    2. Infeasibility test: If $P^t = \emptyset$, stop, $C$ is unsolvable.
        Otherwise let the vertex $x^t \in P^t$ be an optimal solution.
    3. Feasible 0-1 solution: If $x^t_j \in \{0,1\}$, for $j = 1, \ldots, n$, stop,
        $x^t$ is a feasible 0-1 solution of $C$,
        $C^t$ is the solved form of $C$.
    4. Cut generation: For $j \in \{1, \ldots, n\}$ with $0 < x^t_j < 1$ generate a j-cut $\alpha^j x \geq \beta^j$ by solving
            $max\{\beta - \alpha x^t \mid (\alpha, \beta) \in R^*_j(P^t) \cap T\}$.
        Define $C^{t+1}$ by adding the j-cuts $\alpha^j x \geq \beta^j$ to the constraint set $C^t$ and
        simplify $C^t$ considered as a set of linear arithmetic constraints.
    5. Let $t := t + 1$ and goto 1.
```

Example 4.2 Consider again the linear 0-1 inequality

$$3x_1 + 2x_2 + x_3 + x_4 + x_5 \geq 5.$$

For most objective functions, linear optimization over the corresponding linear relaxation yields immediately a 0-1 solution. Minimizing the objective function $x_1 + x_4 + x_5$ yields the fractional solution $x_1 = 2/3, x_2 = x_3 = 1, x_4 = x_5 = 0$. Now, the cutting plane procedure generates the 1-cut $x_1 + x_4 + x_5 \geq 1$. Adding this cut to the linear relaxation and re-optimizing yields the optimal 0-1 solution $x_1 = x_2 = x_3 = 1, x_4 = x_5 = 0$.

For a specialized version of this algorithm we can prove the following correctness and completeness theorem (see Appendix A).

Theorem 4.3 *Given a constraint set $C : \{Ax \geq b, x \in \{0,1\}\}$ the specialized strong cutting plane algorithm finds a feasible 0-1 solution if one exists or detects that C is unsolvable.*

The same algorithm can also be used for *optimization* and *entailment*. If we want to compute an optimal solution to the linear 0-1 optimization problem

Minimize $g(x)$ subject to the constraints $Ax \geq b, x \in \{0,1\}^n$.

we choose $g^t = g$, for all $t = 1, 2, \ldots$. If the algorithm stops with a solution $x^* \in \{0,1\}^n$, then this solution is optimal. In order to check whether a constraint $cx \geq d$ is entailed by a constraint set $Ax \geq b$, we can minimize the objective function cx subject to $Ax \geq b, x \in \{0,1\}^n$. As soon as we find an optimal solution x^t for the linear relaxation P^t such that $cx^t \geq d$, the constraint is entailed. If $cx^* < d$, for an optimal solution x^* of the 0-1 problem, the constraint is not entailed.

We close this section with a remark on incrementality. Suppose we have computed a solved form for the constraint set C and a new constraint has to be added. If we have used the simplex algorithm to solve the linear relaxation P then there is a straightforward way to use the current optimal solution to solve the new problem. If the new constraint is satisfied by the current optimal solution, nothing has to be done. If the constraint is not satisfied, it can be converted to an equality by the addition of a nonnegative slack variable and added to the constraint set. The optimal basis for the original problem and the new slack variable provide a basis for the expanded problem. This new basis is dual feasible and primal feasible in all but the last row. In order to reoptimize, we can therefore use the *dual simplex algorithm* which was designed to deal with just this kind of situation. Since the current solution is "nearly" primal feasible, it is likely that only a few iterations will be required [NW88].

5 Branch and Cut

It is a general experience with cutting plane algorithms that the effect of the cutting planes on the objective function value is more significant at the beginning than at the end [Cer93]. In order to move away from the current optimal solution when the cuts become shallow, the cutting plane algorithm can be combined with classical branch-and bound. The idea of *branch-and-bound* is to divide the set of feasible solutions into subsets, to compute bounds for the objective function on these subsets, and to use these bounds to discard some of the subsets from further consideration. In the case of 0-1 optimization, the splitting of the set of feasible solutions is usually done by fixing a variable to the values 0 and 1 (for more details see for example [NW88]).

Example 5.1 Consider the linear 0-1 optimization problem

Maximize $z = x_1 + x_2 + x_3$ subject to

$$x \in S = \{x \in \{0,1\}^3 \mid x_1 - x_2 \geq 0,\ x_1 - x_3 \geq 0,\ x_1 + x_2 + x_3 \geq 1\}.$$

The branch-and-bound tree is given in Fig. 5. It contains only 7 instead of possibly 15 nodes.

The linear relaxation of S^0 is infeasible. The sets S^{110} resp. S^{111} contain only one point with value 2 resp. 3. For S^{10} we get the upper bound 2 which is smaller than 3. Therefore this node need not be expanded and $(1,1,1)$ is the optimal solution with optimal value 3.

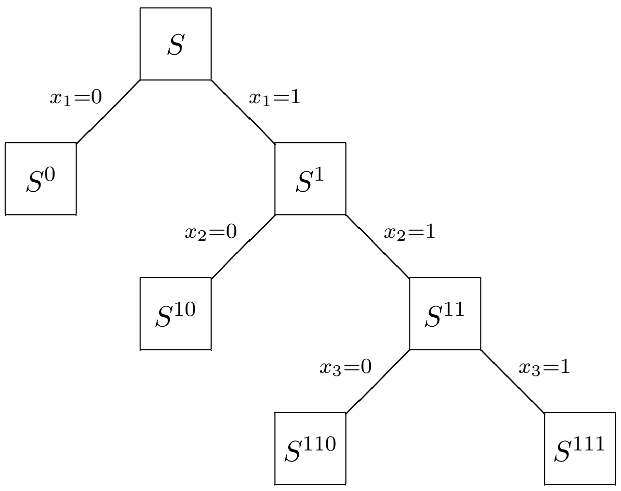

Figure 5: Branch-and-bound tree

Branch-and-bound is the standard technique used in finite domain constraint solvers in case an optimal solution of the given constraint set has to be found. It was also very popular in operations research until it became clear that bounds computed by solving the linear relaxation could be considerably improved by generating suitable cutting planes. Better bounds in turn reduce the size of the branch-and-bound search tree. In a first period, cutting planes were generated only at the root node of the branch-and-bound tree in order to improve the initial formulation given by the linear relaxation.

This had already remarkable effects as is illustrated for example by the standard benchmark p0033 of the mixed integer programming library MIPLIB [BBI92]. This is a linear 0-1 minimization problem consisting of 33 0-1 variables and 15 constraints. If standard branch-and-bound is applied without adding cuts, the best solution found after 1000 nodes has value 3095, and the tree still contains 163 active nodes. If branch-and-bound is applied after generating 20 cuts (in 6 iterations), then a solution of value 3095 is found at node 17, and an optimal solution of value 3089 is found at node 65. Optimality is proved at node 77 [NW88].

In a second step, cutting planes were used not only at the root node of the branch-and-bound tree but also for the rest of the tree [PR91]. The crucial point here is that the cutting planes that are generated are not only valid for the descendants of the current node but for the whole tree. This means that one has to compute coefficients for the variables that have been fixed before. Padberg and Rinaldi called their method, which they developed in the context of solving large instances of the traveling-salesman-problem, *branch-and-cut.*

The cutting planes generated by the lift-and-project method have this crucial property. They can be *lifted* from one node in the branch-and-bound tree to all other nodes (see Appendix B). Therefore, our strong cutting plane algorithm for 0-1 constraint solving can be embedded into a branch-and-cut procedure, which in many cases converges much faster.

Here is a sketch of a full branch-and-cut procedure [BCC93b, PR91, JRT92].
For $F_0, F_1 \subseteq \{1, \ldots, n\}, F_0 \cap F_1 = \emptyset$ let $LP(z, C, F_0, F_1)$ denote the linear optimization problem

$$\text{Minimize } z = cx \text{ subject to } Ax \geq b, 0 \leq x \leq 1,\ x_i = 0, \text{for } i \in F_0,\ x_i = 1, \text{for } i \in F_1.$$

Branch-and-Cut Procedure

1. **Initialization:** Let $\mathcal{L} = \{(\emptyset, \emptyset)\}$, $C = \{Ax \geq b, 0 \leq x \leq 1\}$, $UB = \infty$.
2. **Node Selection:** If $\mathcal{L} = \emptyset$, then stop.
   Otherwise choose an ordered pair (F_0, F_1) and remove it from $\mathcal{L}$.
3. **Lower Bound:** Solve the linear optimization problem $LP(z, C, F_0, F_1)$.
   If the problem is infeasible, goto Step 2, otherwise let $\bar{x}$ denote its optimal solution.
   If $c\bar{x} \geq UB$ goto Step 2.
   If $\bar{x}$ is integer let $x^* = \bar{x}, UB = c\bar{x}$ and goto Step 2.
4. **Branching-Cutting-Decision:** Should cutting planes being generated?
   If yes, goto Step 5, else goto Step 6.
5. **Cut Generation:** Generate lift-and-project cutting planes $\alpha x \geq \beta$ violated by $\bar{x}$.
   Lift the cuts so that they are valid for the whole branch-and-cut tree.
   Add the resulting cuts to C and goto Step 3.
6. **Branching:** Pick an index $j \in \{1, \ldots, n\}$ such that $0 < \bar{x}_j < 1$.
   Generate the subproblems corresponding to $(F_0 \cup \{j\}, F_1)$ and $(F_0, F_1 \cup \{j\})$ and add them to $\mathcal{L}$. Goto Step 2.

When the algorithm stops, the problem is either infeasible or x^* is an optimal solution. C is the solved form of the constraint set $Ax \geq b$. Note that in spite of branching the solved form does not involve disjunction. This important property is obtained by lifting the cuts.

Several other improvements are possible, which we mention only briefly. First of all, we should apply the various preprocessing techniques for 0-1 programs which are described in the literature [HP91]. There is a computationally inexpensive method of *strengthening* the cuts computed by the lift-and-project method which also improves convergence. Finally, in an actual implementation of the branch-and-cut procedure various choices have to be made. For example, we have to decide which normalization T should be chosen, how many cuts should be generated in one iteration, when a branching step should be performed etc.

6 Conclusion and Further Research

In this paper we have presented a new approach for 0-1 constraint solving based on cutting plane techniques from combinatorial optimization. The generation of strong cutting planes may dramatically reduce the search tree of classical branch-and-bound algorithms. It thus provides an interesting alternative to the finite domain techniques used in existing constraint logic programming systems.

When solving linear 0-1 constraints by cutting plane techniques, we have to use linear arithmetic constraints and linear optimization. A natural extension of our approach would be to use linear arithmetic already from the beginning. This leads to a new kind of constraints, *mixed 0-1 constraints*

$$\sum_{i \in I} a_i x_i + \sum_{j \in J} b_j y_j \leq c,$$

with real variables x_i and 0-1 variables y_j.

Mixed 0-1 constraints are particularly interesting because they integrate two of the most important domains in constraint logic programming, Boolean algebra and linear arithmetic. The strong cutting plane method presented in this paper works also for the mixed 0-1 case. Therefore this approach could also provide the basis for a constraint logic programming language for mixed 0-1 constraints.

References

[ASS+88] A. Aiba, K. Sakai, Y. Sato, D.J. Kawley, and R. Hasegawa. Constraint logic programming language CAL. In *Fifth Generation Computer Systems, Tokyo, 1988.* Springer, 1988.

[Bar93a] P. Barth. A complete symbolic 0-1 constraint solver. In *3rd Workshop on Constraint Logic Programming, WCLP'93, Marseille*, March 1993.

[Bar93b] P. Barth. Linear 0-1 inequalities and extended clauses. In *Logic Programming and Automated Reasoning, LPAR'93, St. Petersburg*, Springer, LNCS 698, 1993.

[BB93] P. Barth and A. Bockmayr. Solving 0-1 problems in CLP($\mathcal{PB}$). In *Proc. 9th Conf. Artificial Intelligence for Applications CAIA, Orlando, Florida*, 263 - 269. IEEE, 1993.

[BBI92] R. E. Bixby, E. A. Boyd, and R. Indovina. MIPLIB: A test set for mixed integer programming problems. *SIAM News 25*, page 16, 1992.

[BCC93a] 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.

[BCC93b] E. Balas, S. Ceria, and G. Cornuéjols. Solving mixed 0-1 programs by a lift-and-project method. In *Symposium on Discrete Algorithms SODA, Austin, Texas.* ACM - SIAM, 1993.

[BHW74] G. H. Bradley, P. L. Hammer, and L. Wolsey. Coefficient reduction for inequalities in 0-1 variables. *Mathematical Programming*, 7:263 –282, 1974.

[Bla80] C. E. Blair. Facial disjunctive programs and sequences of cutting planes. *Discrete Applied Mathematics*, 2:173 – 179, 1980.

[Boc93] 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.

[Boy93] E. A. Boyd. Solving integer programs with Fenchel cutting planes and preprocessing. In G. Rinaldi and L. Wolsey, editors, *Integer Programming and Combinatorial Optimization IPCO'93, Erice*, 1993.

[BS87] W. Büttner and H. Simonis. Embedding Boolean expressions in logic programming. *Journal of Symbolic Computation*, 4(2):191–205, 1987.

[Cer93] S. Ceria. *Lift-and-project methods for mixed 0-1 programs.* PhD thesis, GSIA, Carnegie Mellon University, 1993.

[CJP83] H. Crowder, E. J. Johnson, and M. Padberg. Solving large-scale 0-1 linear programming problems. *Operations Research*, 31(5):803–834, 1983.

[Col87] A. Colmerauer. Introduction to PROLOG III. In *4th Annual ESPRIT Conference, Bruxelles.* North Holland, 1987.

[DvHS$^+$88] 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.

[Fag93] F. Fages. On the semantics of optimization predicates in CLP languages. In *Principles and Practice of Constraint Programming PPCP'93, Newport, RI*, 1993.

[Gom58] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bull. AMS*, 64:275 – 278, 1958.

[Hoo92] J. N. Hooker. Generalized resolution for 0-1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 6:271–286, 1992.

[HP85] K. Hoffman and M. Padberg. LP-based combinatorial problem solving. *Annals of Operations Research*, 4:145 – 194, 1985.

[HP91] K. Hoffman and M. Padberg. Improving LP-representations of zero-one linear programs for branch-and-cut. *ORSA Journal on Computing*, 3(2):121 – 134, 1991.

[HR68] P.L. Hammer and S. Rudeanu. *Boolean Methods in Operations Research and Related Areas.* Springer, 1968.

[Jer75] R. G. Jeroslow. On defining sets of vertices of the hypercube by linear inequalities. *Discrete Mathematics*, 11:119 – 124, 1975.

[Jer80] R. G. Jeroslow. A cutting-plane game for facial disjunctive programs. *SIAM J. Control and Optimization*, 18(3):264 – 281, 1980.

[JL87] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proc. 14th ACM Symp. Principles of Programming Languages*, Munich, 1987.

[JM94] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 1994. (to appear).

[JRT92] M. Jünger, G. Reinelt, and S. Thienel. Provably good solutions for the traveling salesman problem. Preprint 92-31, Interdisziplinäres Zentrum für wissenschaftliches Rechnen der Univ. Heidelberg, 1992.

[KP82] R. M. Karp and C. H. Papadimitriou. On linear characterizations of combinatorial optimization problems. *SIAM J. Comput.*, 11(4):620 – 632, 1982.

[Lip87] L. I. Lipkin. Representation of boolean functions with a predetermined number of zeros by systems of linear inequalities. *U.S.S.R. Comput. Maths. Math. Phys.*, 27(3):204 – 206, 1987.

[Mac92] A. K. Mackworth. Constraint satisfaction. In S.C. Shapiro, editor, *Encyclopedia of artificial intelligence*, volume 1, pages 285–293. Wiley, 1992.

[MN89] U. Martin and T. Nipkow. Boolean unification - the story so far. *Journal of Symbolic Computation*, 7:275–293, 1989.

[MT93] K. McAloon and C. Tretkoff. 2lp: Linear programming and logic programming. In *Principles and Practice of Constraint Programming PPCP'93, Newport, RI*, 1993.

[Mur71] S. Muroga. *Threshold Logic and Its Applications.* Wiley-Interscience, 1971.

[NW88] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley, 1988.

[PR91] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60 –100, 1991.

[PS85] U. N. Peled and B. Simeone. Polynomial-time algorithms for regular set-covering and threshold synthesis. *Discrete Applied Mathematics*, 12:57 – 69, 1985.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley, 1986.

[Sch92] M. Schmitt. A slightly improved upper bound on the size of weights sufficient to represent any linearly separable Boolean function. Ulmer-Informatik-Bericht 92-10, Univ. Ulm, November 1992.

[SN93] M. W. P. Savelsbergh and G. L. Nemhauser. Functional description of MINTO, a mixed integer optimizer, version 1.5. Technical report, Computational Optimization Center, Georgia Institute of Technology, November 1993.

[SND88] H. Simonis, H. N. Nguyen, and M. Dincbas. Verification of digital circuits using CHIP. In *Proc. IFIP WG 10.2 Intern. Conf. Fusion of Hardware Design and Verification, Glasgow*, 1988.

[vH89] P. van Hentenryck. *Constraint satisfaction in logic programming.* MIT Press, 1989.

A Correctness and Completeness of the Specialized Cutting Plane Algorithm

In our strong cutting plane algorithm the current polyhedron P^t is defined by the original constraint set $C = \{Ax \geq b, 0 \leq x \leq 1\}$ and a set of cuts $C^t = \bigcup_{j=1}^{n} C_j^t$, where C_j^t is the set of j-cuts which have been generated so far. If P_j^t is the polyhedron defined by $C \cup \bigcup_{i=1}^{j} C_i^t$, then $P_0^t = P, P_n^t = P^t$ and $P_j^t \subseteq P_i^t$, if $i \leq j$.

The special version of the strong cutting plane algorithm is obtained by replacing the polyhedron P^t for which a j-cut is generated by the relaxation $P_{j-1}^t \supset P^t$ and by computing only one cut per iteration.

Remark A.1 For all $t = 1, 2, \ldots$ and for all $j = 1, \ldots, n$: $R_j(P_{j-1}^t) \subseteq P_j^t$.

Proof: P_j^t is obtained from P_{j-1}^t by adding some j-cuts. Any of these j-cuts is valid for $R_j(P_{j-1}^s)$, for some $s < t$. Since $R_j(P_{j-1}^s) \supseteq R_j(P_{j-1}^t)$, these cuts are also valid for $R_j(P_{j-1}^t)$. □

Specialized Strong Cutting Plane Algorithm

Initialization: Let $t := 1$; $C^1 := C = \{Ax \geq b, 0 \leq x \leq 1\}$,

Iteration t: 1. Solution of the linear relaxation: Optimize a linear function g^t over the relaxation $P^t = \{x \in \mathbb{R}^n \mid x \text{ is a solution of } C^t\}$.

2. Infeasibility test: If $P^t = \emptyset$, stop, C is unsolvable.
Otherwise let the vertex $x^t \in P^t$ be an optimal solution.

3. Feasible 0-1 solution: If $x_j^t \in \{0,1\}$, for $j = 1, \ldots, n$, stop,
x^t is a feasible 0-1 solution of C,
C^t is the solved form of C.

4. Cut generation: Let $j \in \{1, \ldots, n\}$ be the largest index with $0 < x_j^t < 1$.
Generate a j-cut $\alpha^j x \geq \beta^j$ by solving

$$max\{\beta - \alpha x^t \mid (\alpha, \beta) \in R_j^*(P_{j-1}^t) \cap T\}.$$

Define C^{t+1} by adding the j-cut $\alpha^j x \geq \beta^j$ to the constraint set C^t.

5. Let $t := t + 1$ and goto 1.

Theorem A.2 *Given a constraint set $C : \{Ax \geq b, x \in \{0,1\}\}$ the specialized strong cutting plane algorithm either finds a feasible 0-1 solution or detects that C is unsolvable.*

Proof: a) First we prove that the inequality $\alpha^j x \geq \beta^j$ generated in Step 4 cuts off x^t.

To prove this, we show by induction that x^t is a vertex of P_k^t, for $k = n, \ldots, j$. The case $k = n$ is trivial, because x^t is a vertex of $P^t = P_n^t$. Suppose that x^t is a vertex of $P_k^t \subseteq P_{k-1}^t$, for some $k \in \{n, \ldots, j+1\}$. Since $x_k^t \in \{0,1\}$, we get by Remark A.1 that $x^t \in P_{k-1}^t \cap \{x \in \mathbb{R}^n \mid x_k = x_k^t\} \subseteq R_k(P_{k-1}^t) \subseteq P_k^t$. This implies that x^t is a vertex of $P_{k-1}^t \cap \{x \in \mathbb{R}^n \mid x_k = x_k^t\}$. Since this is a face of P_{k-1}^t, x^t is also a vertex of P_{k-1}^t.

Next we show that $x^t \notin R_j(P_{j-1}^t)$. Since $R_j(P_{j-1}^t) \subseteq P_j^t$ by Remark A.1, the assumption $x^t \in R_j(P_{j-1}^t)$ would imply that x^t is a vertex of $R_j(P_{j-1}^t)$. But all vertices of $R_j(P_{j-1}^t)$ have a j-th component equal to 0 or 1 in contradiction to $0 < x_j^t < 1$.

Since $x^t \notin R_j(P_{j-1}^t)$ there exists a valid inequality $\alpha x \geq \beta$ for $R_j(P_{j-1}^t)$ that is violated by x^t, i.e. $\beta - \alpha x^t > 0$. We may assume that $(\alpha, \beta) \in R_j^*(P_{j-1}^t) \cap T$. Therefore $\beta^j - \alpha^j x^t = \max\{\beta - \alpha x^t \mid (\alpha, \beta) \in R_j^*(P_{j-1}^t) \cap T\} > 0$, i.e. $\alpha^j x \geq \beta^j$ cuts off x^t.

b) Next we prove by induction that for $j = 1, \ldots, n$ the set C_j of all j-cuts generated by the algorithm is finite.

For $j = 1$, this holds because every 1-cut corresponds to a vertex of $R_1^*(P_0^t) \cap T = R_1^*(P) \cap T$, of which there are only finitely many. Furthermore, every 1-cut cuts off some x^t that satisfies all 1-cuts generated earlier.

Suppose that C_i is finite for all $i = 1, \ldots, j-1$ and consider the case $i = j$. Since P_{j-1}^t is defined by $C \cup \bigcup_{i=1}^{j-1} C_i^t$ and $C_i^t \subseteq C_i$, there exist only finitely many different polyhedra $P_{j-1}^t, t = 1, 2, \ldots$. Every j-cut corresponds to some vertex of some $R_j^*(P_{j-1}^t) \cap T$, of which there are only finitely many, and it cuts off some x^t which satisfies all j-cuts generated earlier. Therefore the set C_j of j-cuts is finite, too.

c) Since only finitely many cutting planes can be generated by the algorithm, for some $t \geq 1$ either $P^t = \emptyset$ or $x^t \in \{0,1\}^n$. $\square$

We used the proof technique of [BCC93a], which in turn is based on the convergence proof in [Jer80]. Note that cutting plane algorithms have the surprising property that choosing the wrong facets or generating deeper cuts than those specified may destroy finite convergence [Bla80].

B Cut Lifting

Given a fractional vertex x^t of the linear relaxation P^t, let $F = \{i \in \{1,\dots,n\} \mid 0 < x_i^t < 1\}$ denote the index set of the fractional components of x^t and let $|F| = r$. By complementing variables if necessary, we can assume that $x_i^t = 0$, for all $i \notin F$.

We now show how a lift-and-project cutting plane separating x^t can also be obtained by working on the subspace defined by F. This means that the size of the linear optimization problem that has to be solved in order to find a cut can be substantially reduced. Let $\tilde{A}^F$ denote the $(m+2r)\times r$-matrix obtained from the $(m+2n)\times n$-matrix $\tilde{A}$ by removing column i and rows $m+i, m+n+i$, for all $i \notin F$. Similarly, denote by $\alpha^F, (x^t)^F, e_j^F \in \mathbb{R}^r$ resp. $\tilde{b}^F, u^F, v^F \in \mathbb{R}^{m+2r}$ the vectors obtained from $\alpha, x^t, e_j \in \mathbb{R}^n$ resp. $\tilde{b}, u, v \in \mathbb{R}^{m+2n}$ by removing the components i resp. $m+i, m+n+i$, for all $i \notin F$. Note that $m+i, m+n+i$ correspond to the constraints $x_i \geq 0, -x_i \geq -1$.

Consider the linear optimization problem

Maximize $\beta - (x^t)^F\alpha^F$ subject to

$$\begin{array}{llllll}
\alpha^F & -u^F\tilde{A}^F & -u_0e_j^F & & & = 0 \\
\alpha^F & & & -v^F\tilde{A}^F & -v_0e_j^F & = 0 \\
 & u^F\tilde{b}^F & & & & = \beta \\
 & & & v^F\tilde{b}^F & +v_0 & = \beta \\
 & & & & u^F, v^F & \geq 0 \\
 & & & & \sum_{i\in F}|\alpha_i| & \leq 1
\end{array} \tag{12}$$

If $(\alpha^F, \beta, u^F, u_0, v^F, v_0)$ is an optimal solution to (12), then $(\alpha, \beta, u, u_0, v, v_0)$, with

$$\begin{array}{lcl}
\alpha_i & = & \alpha_i^F, \text{ for } i \in F \\
\alpha_i & = & \left\{\begin{array}{ll} v^F\tilde{A}_i^F, & \text{if } v^F\tilde{A}_i^F \geq u^F\tilde{A}_i^F \\ u^F\tilde{A}_i^F, & \text{if } u^F\tilde{A}_i^F \geq v^F\tilde{A}_i^F \end{array}\right. \quad \text{for } i \notin F, \\
 & & \\
u_i & = & u_i^F, \text{ for } i = 1,\dots,m \\
v_i & = & v_i^F, \text{ for } i = 1,\dots,m \\
u_{m+i} & = & \left\{\begin{array}{ll} (v^F - u^F)\tilde{A}_i^F, & \text{if } v^F\tilde{A}_i^F > u^F\tilde{A}_i^F \\ 0, & \text{otherwise} \end{array}\right. \quad \text{for } i \notin F, \\
v_{m+i} & = & \left\{\begin{array}{ll} (u^F - v^F)\tilde{A}_i^F, & \text{if } u^F\tilde{A}_i^F > v^F\tilde{A}_i^F \\ 0, & \text{otherwise} \end{array}\right. \quad \text{for } i \notin F, \\
u_{m+n+i} & = & 0, \text{ for } i \notin F \\
v_{m+n+i} & = & 0, \text{ for } i \notin F
\end{array} \tag{13}$$

is a basic feasible optimal solution to the linear optimization problem

$$max\{\beta - \alpha x^t \mid (\alpha, \beta) \in R_j^*(P^t) \cap T\}, \tag{14}$$

with the truncation $T = \{(\alpha, \beta) \mid \sum_{i \in F} |\alpha_i| \leq 1\}$. Here $\tilde{A}_i$ denotes the i-th column of $\tilde{A}$.

The proof of a similar result for the truncation $\beta = 1$ or $\beta = -1$ can be found in [BCC93a].

Example B.1 Consider again the 0-1 constraint set

$$C = \{-2x_1 - 2x_2 \geq -3,\ 2x_1 - 2x_2 \geq -1\}.$$

The linear relaxation P was given by the system of linear inequalities $\tilde{A}x \geq \tilde{b}$, where

$$\tilde{A}^T = \begin{pmatrix} -2 & 2 & 1 & 0 & -1 & 0 \\ -2 & -2 & 0 & 1 & 0 & -1 \end{pmatrix}$$

and

$$\tilde{b}^T = \begin{pmatrix} -3 & -1 & 0 & 0 & -1 & -1 \end{pmatrix}.$$

Suppose we want to generate a cutting plane for the fractional vertex $(0, \frac{1}{2})$ of P. This means that that $F = \{2\}$ and $r = 1$. We remove from $\tilde{A}$ the column 1 and the rows corresponding to $x_1 \geq 0$ and $x_1 \leq 1$. This yields

$$(\tilde{A}^F)^T = \begin{pmatrix} -2 & -2 & . & 1 & . & -1 \end{pmatrix}$$

and

$$(\tilde{b}^F)^T = \begin{pmatrix} -3 & -1 & . & 0 & . & -1 \end{pmatrix}.$$

The corresponding linear optimization problem is

Maximize $-\frac{1}{2}\alpha_2 + \beta$ subject to

$$\begin{array}{cccccccc} \alpha_2 & +2u_1 & +2u_2 & & -u_4 & +u_6 & -u_0 & = 0 \\ \alpha_2 & +2v_1 & +2v_2 & & -v_4 & +v_6 & -v_0 & = 0 \\ & -3u_1 & -u_2 & & & -u_6 & & = \beta \\ & -3v_1 & -v_2 & & & -v_6 & +v_0 & = \beta \end{array}$$

$$\begin{array}{l} -1 \leq \alpha_2 \leq 1 \\ u_1, u_2, u_4, u_6, v_1, v_2, v_4, v_6 \geq 0, \end{array}$$

which has the optimal solution $\alpha_2 = -1, u_0 = -1, v_0 = 1, v_2 = 1$ and all other variables are 0. Lifting gives us the values $u_3 = 2, v_3 = 0, u_5 = v_5 = 0$, and $\alpha_1 = 2$. Altogether, we have computed again the cutting plane $x_1 - \frac{1}{2}x_2 \geq 0$ of $R_2(P)$.