Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

TOBIAS ACHTERBERG[1]
TIMO BERTHOLD[2]
STEFAN HEINZ[2]
THORSTEN KOCH[2]
KATI WOLTER[2]

Constraint Integer Programming: Techniques and Applications

[1] ILOG Deutschland, Ober-Eschbacher Str. 109, 61352 Bad Homburg, Germany, tachterberg@ilog.de
[2] Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, {berthold,heinz,koch,wolter}@zib.de

ZIB-Report 08-43 (October 2008)

Constraint Integer Programming: Techniques and Applications

Tobias Achterberg Timo Berthold* Stefan Heinz*
Thorsten Koch Kati Wolter**

October 31, 2008

Abstract

This article introduces *constraint integer programming* (CIP), which is a novel way to combine constraint programming (CP) and mixed integer programming (MIP) methodologies. CIP is a generalization of MIP that supports the notion of general constraints as in CP. This approach is supported by the CIP framework SCIP, which also integrates techniques for solving satisfiability problems. SCIP is available in source code and free for noncommercial use.

We demonstrate the usefulness of CIP on three tasks. First, we apply the constraint integer programming approach to pure mixed integer programs. Computational experiments show that SCIP is almost competitive to current state-of-the-art commercial MIP solvers. Second, we demonstrate how to use CIP techniques to compute the number of optimal solutions of integer programs. Third, we employ the CIP framework to solve chip design verification problems, which involve some highly nonlinear constraint types that are very hard to handle by pure MIP solvers. The CIP approach is very effective here: it can apply the full sophisticated MIP machinery to the linear part of the problem, while dealing with the nonlinear constraints by employing constraint programming techniques.

1 Introduction

In the recent years, several authors showed that an integrated approach of constraint programming (CP) and mixed integer programming (MIP) can help to solve optimization problems that were intractable with either of the two methods alone [21, 36, 56]. Different approaches to integrate CP and MIP into a single framework have been proposed, [7, 11, 20, 33, 51, 52] amongst others.

Most of the existing work followed the concept of extending a CP framework by basic MIP techniques. In contrast, this paper introduces a way to incorporate CP specific solving methods and its strong modeling capability into the sophisticated MIP solving machinery.

A previous version of this paper can be found in [3].

* Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.

** Supported by the DFG Priority Program 1307 "Algorithm Engineering".

This is achieved by a very low-level integration of the two concepts. The constraints of a CP usually interact through the domains of the variables. As in [11, 20, 51, 52], the idea of *constraint integer programming* (CIP) is to offer a second communication interface, namely the *linear programming (LP) relaxation*. Furthermore, the definition of CIP restricts the generality of CP modeling as little as needed to still gain the full power of all primal and dual MIP solving techniques.

Therefore, CIP is well suited for problems that contain a MIP core complemented by some nonlinear constraints. As an example for such a problem type, the property checking problem is presented in Section 6.

The concept of constraint integer programming is realized in the branch-and-cut framework SCIP. It combines solving techniques from CP, MIP, and the field of solving satisfiability problems (SAT) such that all involved algorithms operate on a single search tree, which yields a very close interaction. A detailed description of the concepts and the software can be found in [2].

The plugins that are provided with the standard distribution of SCIP suffice to turn the CIP framework into a full-fledged MIP solver. In combination with either SoPlex [58] or CLP [25] as LP solver, it is currently one of the fastest noncommercial MIP solvers, see [45] and the results in Section 4. Using Cplex [34] as LP solver, the performance of SCIP is even comparable to state-of-the-art commercial codes.

As a library, SCIP can be used to develop branch-cut-and-price algorithms, and it can be extended to support additional classes of nonlinear constraints by providing so-called constraint handler plugins. We present a solver for the chip design verification problem as one example of this usage.

SCIP is freely available in source code for academic and noncommercial use and can be downloaded from `http://scip.zib.de`. The current version 1.1.0—as of this writing—has interfaces to five different LP solvers and consists of 275 640 lines of C code. The code is actively maintained and extended.

The article is organized as follows: in Section 2, we introduce constraint integer programs. Section 3 presents the building blocks of the constraint integer programming framework SCIP. In Sections 4–6, we demonstrate the usage of SCIP on three applications. First, we employ SCIP as a stand-alone MIP solver. Second, we count optimal solutions with SCIP. Third, we use SCIP as a branch-and-cut framework to solve chip design verification problems. Computational results are given in Sections 4, 5 and 6.4.

2 Constraint Integer Programming

The hope of integrating CP, MIP, and SAT techniques is to combine their advantages and to compensate for their individual weaknesses. A constraint program is defined as follows.

Definition (constraint program)**.** *A* constraint program *is a triple* $CP = (\mathfrak{C}, \mathfrak{D}, f)$ *with* $\mathfrak{D} = \mathcal{D}_1 \times \ldots \times \mathcal{D}_n$ *representing the domains of finitely many variables* $x_j \in \mathcal{D}_j$, $j = 1, \ldots, n$, $\mathfrak{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ *being a finite set of constraints* $\mathcal{C}_i : \mathfrak{D} \to \{0, 1\}$, $i = 1, \ldots, m$, *and* $f : \mathfrak{D} \to \mathbb{R}$ *being the objective function. It consists of solving*

$$(CP) \quad f^\star = \min\{f(x) \mid x \in \mathfrak{D},\ \mathfrak{C}(x)\},$$

with $\mathfrak{C}(x) :\Leftrightarrow \forall i = 1, \ldots, m : \mathcal{C}_i(x) = 1$. *A CP where all domains* $\mathcal{D} \in \mathfrak{D}$ *are finite is called a* finite domain constraint program *(CP(FD)).*

Note that, with a slight abuse of notation, we use the abbreviation CP for the term constraint programming as well as for the term constraint program. The same holds for MIP and CIP. In case the meaning is not clear from context we use the long versions.

To solve a CP(FD), the problem is recursively split into smaller subproblems, thereby creating a branching tree and implicitly enumerating all potential solutions. At each subproblem, domain propagation is performed to exclude further values from the variables' domains.

Due to the very general definition of a CP, solvers have to rely on constraint propagators, each of them exploiting the structure of a single constraint class. Usually, the only communication between the individual constraints takes place via the variables' domains. An advantage of CP is, however, the possibility to model the problem more directly than in MIP, using very expressive constraints, which maintain the structure of the problem. In MIP, we are restricted to linear constraints, a linear objective function, and integer or real-valued domains. A mixed integer program is defined as follows.

Definition (mixed integer program)**.** *Given a matrix* $A \in \mathbb{R}^{m \times n}$*, vectors* $b \in \mathbb{R}^m$*, and* $c \in \mathbb{R}^n$*, and a subset* $I \subseteq N = \{1, \ldots, n\}$*, the corresponding* mixed integer program $MIP = (A, b, c, I)$ *is to solve*

$$(MIP) \quad c^\star = \min \{c^T x \mid Ax \leq b,\ x \in \mathbb{R}^n,\ x_j \in \mathbb{Z} \text{ for all } j \in I\}.$$

Note, that MIPs in maximization form can be transformed to minimization form by multiplying the objective function vector by -1. Similarly, "$\geq$" constraints can be multiplied by -1 to obtain "$\leq$" constraints. Equations can be replaced by two opposite inequalities.

Like CP solvers, most modern MIP solvers recursively split the problem into smaller subproblems. However, the processing of the subproblems is different. Because MIP includes only one type of constraints, MIP solvers can apply sophisticated techniques that operate on the subproblem as a whole. Usually, for each subproblem, the LP relaxation is solved, which is constructed from the MIP by removing the integrality conditions. The LP relaxation can be strengthened by cutting planes which use the LP information and the integrality restrictions to derive valid linear inequalities that cut off the solution of the current LP relaxation without removing feasible MIP solutions. The LP relaxation usually gives a much stronger bound than the one that is provided by simple dual propagation of CP solvers. Solving the LP relaxation usually requires much more time, however.

Satisfiability problems is also a very specific case of CPs with only one type of constraints, namely Boolean clauses. The Boolean truth values *false* and *true* are identified with the values 0 and 1, respectively, and Boolean formulas are evaluated correspondingly.

Definition (satisfiability problem)**.** *Let* $\mathfrak{C} = \mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_m$ *be a logic formula in conjunctive normal form on Boolean variables* $x_1, \ldots, x_n$*. Each clause* $\mathcal{C}_i = \ell_1^i \vee \ldots \vee \ell_{k_i}^i$ *is a disjunction of literals. A literal* $\ell \in L = \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$ *is either a variable* x_j *or the negation of a variable* $\bar{x}_j$*. The task of the* satisfiability

problem *(SAT) is to either find an assignment* $x^\star \in \{0,1\}^n$*, such that the formula* $\mathfrak{C}$ *is satisfied, i.e., each clause* $\mathcal{C}_i$ *evaluates to* 1*, or to conclude that* $\mathfrak{C}$ *is unsatisfiable, i.e., for all* $x \in \{0,1\}^n$ *at least one* $\mathcal{C}_i$ *evaluates to* 0*.*

Modern SAT solvers also use a branching scheme to split the problem into smaller subproblems and they apply *Boolean constraint propagation* on the subproblems, which is a special form of domain propagation. In addition, they analyze infeasible subproblems to produce *conflict clauses*. These help to prune the search tree later on. Furthermore, SAT solvers support periodic restarts of the search in order to revise the branching decisions after having gained new knowledge about the structure of the problem instance.

Boolean clauses can easily be linearized, but the LP relaxation is rather useless, as it cannot detect the infeasibility of subproblems earlier than domain propagation. Therefore, SAT solvers mainly exploit the special problem structure to speed up the domain propagation algorithm.

To specify our approach of integrating CP, MIP, and SAT solving techniques, we propose the following slight restriction of CP, which allows the application of MIP solving techniques:

Definition (constraint integer program)**.** *A* constraint integer program $CIP = (\mathfrak{C}, I, c)$ *consists of solving*

$$(CIP) \quad c^\star = \min\{c^T x \mid \mathfrak{C}(x),\ x \in \mathbb{R}^n,\ x_j \in \mathbb{Z} \text{ for all } j \in I\}$$

with a finite set $\mathfrak{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ *of constraints* $\mathcal{C}_i : \mathbb{R}^n \to \{0,1\}$*,* $i = 1, \ldots, m$*, a subset* $I \subseteq N = \{1, \ldots, n\}$ *of the variable index set, and an objective function vector* $c \in \mathbb{R}^n$*. A CIP has to fulfill the following additional condition:*

$$\forall \hat{x}_I \in \mathbb{Z}^I\ \exists (A', b') : \{x_C \in \mathbb{R}^C \mid \mathfrak{C}(\hat{x}_I, x_C)\} = \{x_C \in \mathbb{R}^C \mid A' x_C \leq b'\} \qquad (1)$$

with $C := N \setminus I$*,* $A' \in \mathbb{R}^{k \times C}$*, and* $b' \in \mathbb{R}^k$ *for some* $k \in \mathbb{Z}_{\geq 0}$*.*

Restriction (1) ensures that the remaining subproblem after fixing all integer variables always is a linear program. This means that in the case of finite domain integer variables, the problem can be—in principle—completely solved by enumerating all values of the integer variables and then solving the remaining LPs.

Note, that this does not forbid quadratic or even more involved expressions. Only the remaining part after fixing (and thus eliminating) the integer variables must be linear in the continuous variables. Furthermore, the linearity restriction of the objective function can be compensated by introducing an auxiliary objective variable z that is linked to the actual nonlinear objective function with a constraint $z = f(x)$. Analogously, general variable domains can be represented as additional constraints.

Therefore, every CP that meets Condition (1) can be represented as a CIP. Especially, the following proposition holds.

Proposition. *The notion of constraint integer programming generalizes finite domain constraint programming and mixed integer programming:*

(a) Every CP(FD) can be modeled as a CIP.

(b) Every MIP can be modeled as CIP.