Detecting Infeasibility and Generating Cuts for MIP using CP

Alexander Bockmayr, Nicolai Pisaruk

To cite this version:

Alexander Bockmayr, Nicolai Pisaruk. Detecting Infeasibility and Generating Cuts for MIP using CP. 5th International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems - CPAIOR'03, 2003, Montréal, Canada, 11 p. inria-00107699

HAL Id: inria-00107699

https://hal.inria.fr/inria-00107699

Submitted on 19 Oct 2006

Proceedings CPAIOR'03

Detecting Infeasibility and Generating Cuts for MIP using CP

Alexander Bockmayr **Nicolai Pisaruk**
LORIA, Université Henri Poincaré,
BP 239, 54506 Vandœuvre-lès-Nancy, France
`bockmayr@loria.fr, pisaruk@loria.fr`

Abstract

We study a hybrid MIP/CP solution approach in which CP is used for detecting infeasibilities and generating cuts within a branch-and-cut algorithm for MIP. Our framework applies to MIP problems augmented by monotone constraints that can be handled by CP. We illustrate our approach on a generic multiple machine scheduling problem, and compare it to other hybrid MIP/CP algorithms.

1 Introduction

Mixed integer programming (MIP) and constraint programming (CP) are two complementary approaches for solving complex combinatorial optimization problems. Combining these two methods has been an important research topic during the last years. At the same time, this has also been crucial for solving various real-world industrial applications. In this paper, we study how CP can be used inside a branch-and-cut framework in order to detect infeasibility and to generate cutting planes for MIP. For other uses of cutting planes in hybrid solvers, see for example [3, 9, 5, 8].

The ability of existing CP software to generate cuts using infeasibility is limited. If a filtering procedure for some constraint detects a reason of infeasibility independently of the other constraints, it may possibly infer a cut. However, this happens very seldom. In general, infeasibility is detected as the result of propagation between different constraints. The only known general class of valid inequalities that can be generated by CP in case of infeasibility are so-called "no good" cuts [6] of the following form:

$$\sum_{i \in O} x_i - \sum_{i \in Z} x_i \le |O| - 1,$$

for 0-1 variables x_i, where $O = \{i : x_i = 1\}$ and $Z = \{i : x_i = 0\}$.

In this paper, we develop a hybrid branch-and-cut algorithm for MIP problems augmented by *monotone* constraints that can be handled by CP. The key ingredient of this algorithm are separation heuristics for monotone constraints. We start in Sect. 2 with the definition and some basic properties of monotone constraints. Next we present two heuristic separation procedures. In Sect. 3, we describe a hybrid branch-and-cut algorithm for solving MIP problems with monotone constraints. We illustrate our solution approach in Sect. 4 on a generic multiple machine scheduling problem, and report on a number of computational experiments. Finally, in Sect. 5, we compare our hybrid branch-and-cut algorithm with another hybrid MIP/CP algorithm based on the idea of redundant modeling, which includes both a MIP and a CP formulation.

2 Monotone constraints

For $l, u \in \mathbb{Z}^n$, $l \leq u$, let $[l, u]$ denote the set $\{x \in \mathbb{Z}^n : l \leq x \leq u\}$. A function $F : [l, u] \to \{0, 1\}$ is *monotone* if for all $x, y \in [l, u]$ with $y \leq x$, we have $F(y) \leq F(x)$. A constraint $F(x_1, \ldots, x_n) = 0$ is *monotone* if F is monotone. For $\delta \in \{0, 1\}$, we denote by $F^{-1}(\delta)$ the set $\{x \in [l, u] : F(x) = \delta\}$.

Note that monotone constraints are defined for general integer variables. Such constraints arise naturally in many areas, for example in planning and scheduling. Suppose we have to handle tasks (or batches) of n different types. If $F(x_1, \ldots, x_n) = 0$ expresses that x_i tasks of type i can be executed on a given resource, monotonicity means that removing some of the tasks cannot make the problem infeasible.

2.1 Basic properties

Suppose $x \in F^{-1}(0)$ and $y \in F^{-1}(1)$. Since F is monotone, we cannot have $y \leq x$. Therefore, the disjunction

$$(x_1 \leq y_1 - 1) \ \vee \ (x_2 \leq y_2 - 1) \ \vee \ \ldots \ \vee (x_n \leq y_n - 1) \tag{1}$$

must hold.

Proposition 1 *If all y_i take their lower or upper bound values, i.e., if $y_i = l_i \vee y_i = u_i$, for $i = 1, \ldots, n$, the disjunction (1) is equivalent to the inequality*

$$\sum_{i: y_i > l_i} x_i \ \leq (\sum_{i: y_i > l_i} y_i) - 1 \tag{2}$$

In general, inequality (2) is stronger than the disjunction (1).

Proof: (2) $\Rightarrow$ (1): Suppose $\bigvee_{i=1}^{n}(x_i \leq y_i - 1)$ is false. It follows $\bigwedge_{i=1}^{n}(x_i \geq y_i)$ is true, which implies that $\sum_{i: y_i > l_i} x_i \geq \sum_{i: y_i > l_i} y_i$, contradicting (2).

(1) $\Rightarrow$ (2): Suppose all y_i take their lower or upper bound values. Define $I_1 = \{i : y_i = l_i\}$ and $I_2 = \{i : y_i = u_i\}$. Then the disjunction (1) is equivalent to $\bigvee_{i \in I_1}(x_i \leq y_i - 1) \ \vee \ \bigvee_{i \in I_2}(x_i \leq y_i - 1)$. Since $y_i = l_i \leq x_i$, for $i \in I_1$, the first part is false. For the second part, we have $x_i \leq y_i = u_i$, for $i \in I_2$, which implies $\sum_{i \in I_2} x_i \leq \sum_{i \in I_2} y_i$. Suppose equality holds. Then $x_i = y_i$, for $i \in I_2$, contradicting $\bigvee_{i \in I_2}(x_i \leq y_i - 1)$. $\square$

For binary variables, where $[l, u] = \{0, 1\}^n$, conditions (2) and (1) are equivalent. In the binary case, inequality (2) can be written as the *cardinality inequality*

$$\sum_{i \in S(y)} x_i \leq |S(y)| - 1, \tag{3}$$

where $S(y) \overset{\text{def}}{=} \{i : y_i \neq 0\}$ denotes the *support* of $y \in \mathbb{R}^n$. If $F : \{0,1\}^n \to \{0,1\}$ is a *threshold function*, i.e., for some $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$, we have $F(x) = 0$ if $a^T x \leq b$, then $F(y) = 1$ only if the set $I = S(y)$ is a *cover*, i.e., $\sum_{i \in I} a_i > b$. This is why, for threshold functions, cardinality inequalities are called *cover inequalities.*

2.2 Separation heuristics

Consider an optimization problem involving a monotone constraint

$$F(x_1, \ldots, x_n) = 0.$$

To generate cardinality cuts within a branch-and-cut framework (see also Sect. 3), we propose two heuristic separation procedures. An important point is that these heuristics may be applied to arbitrary *fractional* points $\tilde{x} \in \mathbb{R}^n$, and not just to infeasible integer points.

Heuristic 1: Suppose we have to separate the point $\tilde{x} \in \mathbb{R}^n, l \leq \tilde{x} \leq u$. Define $y \in [l, u]$ by $y_i = l_i$, if $\tilde{x}_i < u_i$, and $y_i = u_i$, otherwise. If y is infeasible, i.e., $F(y) = 1$, it follows from the above proposition that inequality (2) is a cutting plane separating y and $\tilde{x}$ (since $\tilde{x} \geq y$) from the convex hull $\operatorname{conv}(F^{-1}(0))$ of $F^{-1}(0)$. Our first separation heuristic just generates these cuts.

Heuristic 2: Next consider the special case $[l, u] = [0, 1]$. We get

$$F^{-1}(0) = \left\{ x \in \{0,1\}^n : \sum_{i \in S(y)} x_i \leq |S(y)| - 1, \text{ for all } y \in F^{-1}(1) \right\}.$$

Our second heuristic allows separating possibly fractional points $\tilde{x} \in \mathbb{R}^n, 0 \leq \tilde{x} \leq 1$, from $\operatorname{conv}(F^{-1}(0))$.

1. Sort the components of $\tilde{x}$ in non-increasing order:
$$\tilde{x}_{\pi(1)} \geq \ldots \geq \tilde{x}_{\pi(n)}$$
2. Let $r \in \{1, \ldots, n\}$ be the largest index such that
$$\sum_{j=1}^{r} \tilde{x}_{\pi(j)} > r - 1.$$
Define $y \in \{0,1\}^n$ as follows: $y_{\pi(j)} = 1$, if $1 \leq j \leq r$, and $y_{\pi(j)} = 0$, if $r+1 \leq j \leq n$.
3. If $F(y) = 1$, the inequality
$$\sum_{j=1}^{r} x_{\pi(j)} \leq r - 1$$
separates $\tilde{x}$ from $\operatorname{conv}(F^{-1}(0))$.

Of course, this procedure may fail to separate $\tilde{x}$ from $\operatorname{conv}(F^{-1}(0))$ even if $\tilde{x} \notin \operatorname{conv}(F^{-1}(0))$. Nevertheless, it always separates integral infeasible points. Being applied to a threshold function, it becomes the greedy heuristic for generating knapsack cover inequalities [11].

3 MIP problems with monotone constraints

We are interested in mixed integer optimization problems of the following form

$$\max cx + uy, \tag{4}$$
$$Ax + Gy \leq b, \tag{5}$$
$$F_i(x^i) = 0, \quad i = 1, \dots, m, \tag{6}$$
$$l^1 \leq x \leq u^1,\ x \in \mathbb{Z}^{n_1}, \tag{7}$$
$$l^2 \leq y \leq u^2,\ y \in \mathbb{R}^{n_2}, \tag{8}$$

where the vector $x^i = (x_{i_1}, \dots, x_{i_{k(i)}})$, for $i = 1, \dots, m$, is composed of some components of the vector $x = (x_1, \dots, x_{n_1})$, and F_i is a monotone function defined over the set

$$\mathrm{dom}(F_i) = \{x^i \in \mathbb{Z}^{k(i)} :\ l^1_{i_j} \leq x_{i_j} \leq u^1_{i_j},\ j = 1, \dots, k(i)\}.$$

We assume that the function F_i is given by a procedure that computes $F_i(x^i)$ at any point $x^i \in \mathrm{dom}(F_i)$.

In order to solve problem (6), we use a standard branch-and-cut approach [11], with the following subroutines:

A **separation procedure** is needed in order to verify whether an optimal LP solution $\tilde{x}$ at the current node is feasible, and if this is not the case, to produce an inequality cutting off $\tilde{x}$. Here, we may use Heuristic 1 from Sect. 2.2. If all arguments of F_i are binary, we may alternatively use Heuristic 2.

The **branching procedure** is called each time a new branching variable has to be selected. Let $\tilde{x}$ be an optimal LP solution at the current node. If $\tilde{x}$ has integer components that take fractional values, we choose as usual one of these components as the branching variable. Otherwise, either $F_i(\tilde{x}^i) = 0$ for $i = 1, \dots, m$, and thus $\tilde{x}$ is a feasible solution to problem (6), or $F_i(\tilde{x}^i) = 1$ for some i. In the latter case, we choose a component, $\tilde{x}_{i_j} = \tilde{x}_k$, with $l_k < \tilde{x}_k < u_k$. Note that such a component $\tilde{x}_k$ exists since $\tilde{x}$ was not cut off by our separation procedure. We branch on $\tilde{x}_k$ in the following way. In the first subproblem we set $u_k = \tilde{x}_k - 1$, in the second $l_k = \tilde{x}_k$. Although $\tilde{x}$ still remains feasible for the second problem, this does not cause any difficulty:

a) It will not take much time to solve the LP relaxation for problem 2 (in fact, $\tilde{x}$ will be computed during the first LP iteration).

b) Since $\tilde{x}_k$ now takes its lower bound value, at later stages a new variable will be chosen for branching or the node will be cancelled.

4 An example scheduling problem

To illustrate our approach, we consider a generic multiple machine scheduling problem [6, 7, 10]. There are n tasks and m dissimilar machines. Any task can be processed on any machine. The processing cost and the processing time of task i on machine j are c_{ij} and p_{ij}, respectively. Processing of task i can only begin after the release date r_i, and must be completed at the latest by the due date d_i. The problem is to carry out all the tasks at the least possible cost.

The main decisions involved in this scheduling problem are:

1) assignment of tasks to machines;

2) sequencing the tasks on each machine, and defining starting times for all the tasks.

Let x_{ij} be a 0-1 variable which takes value 1 if task i is assigned to machine j, $x_{ij} = 0$ otherwise. For $j = 1, \dots, m$, let $F_j : \{0,1\}^n \to \{0,1\}$ denote a function which takes value 0 at $z \in \{0,1\}^n$ iff it is possible to fulfill the tasks from the support set $S(z)$ on machine j respecting the release and due dates. Then we have the following optimization problem

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} \tag{9}$$

$$\sum_{j=1}^{m} x_{ij} = 1\,, \quad i = 1, \dots, n, \tag{10}$$

$$\sum_{j=1}^{m} p_{ij} x_{ij} \le d_i - r_i\,, \quad i = 1, \dots, n, \tag{11}$$

$$\sum_{i=1}^{n} p_{ij} x_{ij} \le \max_{1 \le i \le n} d_i - \min_{1 \le i \le n} r_i\,, \quad j = 1, \dots, m, \tag{12}$$

$$F_j(x_{1j}, \dots, x_{nj}) = 0, \quad j = 1, \dots, m, \tag{13}$$

$$x_{ij} \in \{0,1\}, \quad i = 1, \dots, n, \quad j = 1, \dots, m. \tag{14}$$

The objective in this problem is to minimize the processing cost of all the tasks. Equalities (10) enforce the assignment of each task to exactly one machine. Constraints (11) ensure that the processing time of any task is not greater than the time interval between the release and the due dates of this task. Inequality (12) is a valid cut and tightens the LP relaxation of the problem. It restricts the total processing time of all the tasks that are assigned to the same machine to be not greater than the difference of the latest due date and the earliest release date. The logical constraints (13) ensure that it is possible to sequence the tasks on each machine so that each task is processed within the time interval given by the release and due dates.

To compute $F_j(z)$, we solve a CP problem with only one global constraint. Let $D_j(z) = \max_{i \in S(z)} d_i$, and let us assume that $S(z) = \{s_1, \dots, s_k\}$, $k = |S(z)|$. Then $F_j(z) = 0$ iff the CP problem

$$t_{s_i} \in [r_{s_i}, d_{s_i} - p_{s_i,j}], \quad i = 1, \dots, k,$$
$$\texttt{cumulative}([[t_{s_1}, p_{s_1,j}, 1], \dots, [t_{s_k}, p_{s_k,j}, 1]], 1, D_j(z))$$

has a solution.

The constraint `cumulative` is defined in the usual way [1]. Suppose there are n tasks; task j is characterized by three parameters, which can be either domain variables or values: the starting time $start_j$, the duration dur_j, and the amount res_j of some resource consumed by the task. We are also given the latest completion time e for all the tasks, and the upper bound v on the resource consumption; e and v again are domain variables or values. The global constraint

$$\texttt{cumulative}([[start_1, dur_1, res_1], \dots, [start_n, dur_n, res_n]], v, e)$$

Test	Obj	Best LP Bound	Time	Number of Iter.	Number of Nodes	Number of Cuts
3-12.a	101	101	1.3	23	1111	34
3-12.b	104	104	20.4	61	34531	102
5-15.a	115	115	2.1	22	1596	34
5-15.b	129	129	184.8	179	121023	410
5-20.a	158	158	18.0	30	22991	63
5-20.b	-	139	3600	107	2990240	268
6-24	-	226	3600	55	2850741	124
7-30	-	209	3600	57	1411405	225

Table 1: Iterative MIP/CP algorithm

is satisfied if the following conditions hold:

$$\begin{aligned} \sum_{\substack{1\le j\le n:\\ start_j\le t< start_j+dur_j}} res_j &\le v\,, \quad t=1,\dots,e,\\ \max_{1\le j\le n}(start_j+dur_j) &\le e\,. \end{aligned}$$

4.1 Implementation and computational results

We have implemented a hybrid MIP/CP branch-and-cut algorithm for problem (9)-(14), following the description in Sect. 3. The combination was done in the MOSEL system [4], using Xpress-MP 2003B as the MIP solver, and CHIP Version 5.3 as the CP solver. All experiments were done on a Pentium III, 600 MHz, with 256 MB memory. Running times are given in seconds. The default time limit was set to 3600 seconds. Cuts were generated at each node of the branch-and-cut tree.

In [7], another hybrid MIP/CP algorithm is developed, which uses an iterative approach. At each iteration, first a MIP subproblem is solved. Then CP is called several times in order to check the feasibility of the MIP solution that has been found. In case of infeasibility, cuts are added and the next iteration starts. It has to be noted that, whenever new cuts are added, the MIP subproblem is solved from scratch. In order to compare the two approaches, this iterative method has also been implemented in MOSEL.

Both programs were tested on a set of 8 problem instances. The three problems 3-12.a, 5-15.a, and 5-20.a, are the hardest instances from [7] (note that the other problems in [7] are very easy). The next three test cases, 3-12.b, 5-15.b, and 5-20.b, where produced by perturbating some of the problem parameters. These new instances are harder to solve than the original ones, in particular this holds for problem 5-20.b. To estimate the performance of the algorithms when the size (number of machine and tasks) grows, we generated two more instances, 6-24, and 7-30. The computational results for solving these instances using the MIP/CP iterative method and the MIP/CP branch-and-cut algorithm are presented in Tab. 1 and Tab. 2, respectively.

The algorithm whose performance is given in Tab. 2 uses only Heuristic 1 for separation. Even with this very simple heuristic, the branch-and-cut algorithm is superior

Test	First Solution		Best Solution		Total	Number	Number
	Obj	Time	Obj	Time	Time	of Nodes	of Cuts
3-12.a	103	0.17	101	0.26	0.44	199	85
3-12.b	104	3.52	104	3.52	3.59	1388	1596
5-15.a	117	0.18	115	0.34	1.32	870	229
5-15.b	138	0.25	129	6.30	6.38	1835	2724
5-20.a	158	0.19	158	0.19	13.73	6617	1557
5-20.b	144	0.22	139	142.05	6030.11	535077	115824
6-24	234	0.34	227	1017.07	> 10h	-	-
7-30	223	44.19	219	2082.61	> 10h	-	-

Table 2: Branch-and-Cut MIP/CP, Heuristic 1

Test	First Solution		Best Solution		Total	Number	Number
	Obj	Time	Obj	Time	Time	of Nodes	of Cuts
3-12.a	104	0.21	101	0.73	0.84	214	243
3-12.b	104	1.78	104	1.78	1.86	288	610
5-15.a	116	0.26	115	0.48	0.83	188	161
5-15.b	138	0.51	129	1.89	2.08	295	538
5-20.a	161	0.53	158	3.77	6.23	1068	675
5-20.b	144	0.42	139	7.64	342.89	44882	4228
6-24	231	0.48	227	495.78	8320.53	437705	13587
7-30	230	30.03	214	1063.10	> 10h	-	-

Table 3: Branch-and-Cut MIP/CP, Heuristic 2

Test	First Solution		Best Solution		Total	Number	Number
	Obj	Time	Obj	Time	Time	of Nodes	of Cuts
3-12.a	101	0.19	101	0.19	0.47	78	93
3-12.b	104	0.70	104	0.70	0.87	149	210
5-15.a	115	0.26	115	0.26	0.70	144	104
5-15.b	137	0.27	129	0.47	0.90	177	164
5-20.a	159	0.42	158	2.07	4.01	924	360
5-20.b	145	0.44	139	7.41	245.77	38303	3179
6-24	231	0.65	227	158.91	5762.89	389372	9901
7-30	218	2.12	213*	36204.6	> 10h	-	-

Table 4: Branch-and-Cut MIP/CP, Heuristic 2 with cycle cuts. (*) Optimal solution

to the iterative approach. This corresponds to results reported in [10], where similar problems were solved in a branch-and-check framework with “no good” cuts.

The computational experiments for Heuristic 2 are given in Tab. 3. The results show that Heuristic 2 is more efficient. As the size of the problem grows, the branch-and-cut tree usually contains many nodes. To further improve the approach, we implemented another separation procedure which generates only cycle cuts (cardinality cuts with minimal support). The computational results for this version of the branch-and-cut algorithm are presented in Tab. 4. As one may expect, the stronger cuts improve the overall performance of the algorithm.

In all these approaches, we observe a degradation of the performance when the problem size increases. In our opinion, this can be explained by the fact that cardinality cuts with fewer variables are stronger than those with many variables. As the average number of tasks per machine grows, cuts are getting weaker and therefore LP relaxations are less tight. However, for large instances, the hybrid branch-and-cut algorithm is the only approach that still produces feasible solutions of good quality.

5 Redundant modeling

The *redundant modeling* approach includes both an integer programming and a constraint programming model. The following MIP formulation extends the MIP part in problem (9)–(14).

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} \tag{15}$$

$$\sum_{j=1}^{m} x_{ij} = 1\,, \quad i = 1, \ldots, n, \tag{16}$$

$$\sum_{i=1}^{n} p_{ij} x_{ij} \le \max_{1 \le i \le n} d_i - \min_{1 \le i \le n} r_i\,, \quad j = 1, \ldots, m, \tag{17}$$

$$start_i + \sum_{j=1}^{m} p_{ij} x_{ij} - stop_i = 0, \quad i = 1, \ldots, n, \tag{18}$$

$$stop_{i_1} - start_{i_2} + U y_{i_1 i_2} \le U, \quad i_1, i_2 = 1, \ldots, n,\ i_1 \ne i_2, \tag{19}$$

$$y_{i_1 i_2} + y_{i_2 i_1} \le 1, \quad i_1 = 1, \ldots, n-1,\ i_2 = i_1, \ldots, n, \tag{20}$$

$$x_{i_1 j} + x_{i_2 j} - y_{i_1 i_2} - y_{i_2 i_1} \le 1, \quad i_1, i_2 = 1, \ldots, n,\ j = 1, \ldots, m, \tag{21}$$

$$x_{i_1 j_1} + x_{i_2 j_2} + y_{i_1 i_2} + y_{i_2 i_1} \le 2, \quad i_1, i_2 = 1, \ldots, n, \tag{22}$$

$$j_1, j_2 = 1, \ldots, m,\ j_1 \ne j_2\,, \tag{23}$$

$$x_{ij} \in \{0,1\}, \quad i = 1, \ldots, n,\ j = 1, \ldots, m, \tag{24}$$

$$y_{i_1 i_2} \in \{0,1\}, \quad i_1, i_2 = 1, \ldots, n,\ i_1 \ne i_2, \tag{25}$$

$$r_i \le start_i, stop_i \le d_i\,, \quad i = 1, \ldots, n. \tag{26}$$

Constraint (18) relates start $start_i$ and completion $stop_i$ times for task i according to the machine assignment. The sequencing constraint (19) ensures that if a *sequencing* variable $y_{i_1 i_2}$ takes the value 1, task i_2 is processed after task i_1. Here, U denotes a

Test	First Solution		Best Solution		Best LP	Total	Number
	Obj	Time	Obj	Time	Bound	Time	of Nodes
3-12.a	106	24	101	376	101	376	96869
3-12.b	104	156	104	156	104	156	46445
5-15.a	124	5	115	106	115	2435	156800
5-15.b	142	31	130	843	123	3600	235483
5-20.a	167	4242	167	4242	156	4242	67274
5-20.b	148	139	148	139	136	3600	61512
6-24	235	306	235	306	224	3600	34739
7-30	-	-	-	-	206	10046	31200

Table 5: Pure MIP

suitable upper bound for $stop_{i_1} - start_{i_2}$. The constraints (20) guarantee that only one of the two tasks i_1, i_2 can be processed before the other one. Constraint (21) defines a logical relationship between the assignment variables x_{ij} and the sequencing variables $y_{i_1 i_2}$. If tasks i_1, i_2 are assigned to machine j, then either $y_{i_1 i_2} = 1$ (task i_1 is processed before task i_2) or $y_{i_2 i_1} = 1$ (task i_2 is processed before task i_1). Constraint (22) ensures that $y_{i_1 i_2}$ is zero if tasks i_1 and i_2 are assigned to different machines.

The main disadvantage of the iterative algorithm is that it produces only one feasible solution, which is also optimal. If the method fails to solve the problem to optimality, we do not get any feasible solution. Therefore, from a practical point of view, even a pure MIP approach might be preferable. Computational results for solving the same test problems using a pure MIP approach for the model (15)–(26) (implemented in MOSEL with all the default settings of XPRESS, including cut generation at the root node) are presented in Tab. 5. For all but one test problems, the pure MIP program was able to produce feasible solutions of reasonable quality rather quickly.

Using MOSEL, we formulated a new CP model in terms of tasks and machines, which is given by the following constraints:

$$\min \sum_{i=1}^{n} c_{i,mach_i} \tag{27}$$

$$dur_i = p_{i,mach_i}\,, \quad i = 1, \dots, n, \tag{28}$$

$$stop_i = start_i + dur_i\,, \quad i = 1, \dots, n, \tag{29}$$

$$\texttt{diffn}([start_i, mach_i, dur_i, 1]_{i=1}^{n}), \tag{30}$$

$$start_i, stop_i \in [r_i, d_i]\,, \quad i = 1, \dots, n, \tag{31}$$

$$dur_i \in [\min_{1 \le j \le m} p_{ij}, \max_{1 \le j \le m} p_{ij}]\,, \quad i = 1, \dots, n, \tag{32}$$

$$mach_i \in [1, m]\,, \quad i = 1, \dots, n. \tag{33}$$

In the CP model, n^2 0-1 variables x_{ij} are replaced by n domain variables $mach_i$ with domain size m, where $mach_i = j$ if task i is assigned to machine j. The variables $start_i$, $stop_i$, dur_i give, respectively, the start time, completion time, and the duration of task i. The global constraint $\texttt{diffn}([x_i, y_i, l_i, w_i]_{i=1}^{n})$ is satisfied if the rectangles $[x_i, y_i, l_i, w_i]$ ((x_i, y_i) is the lower left corner, l_i and w_i are, respectively, the length and width) do not overlap [2].

Test	First Solution		Best Solution		Best LP	Total	Number
	Obj	Time	Obj	Time	Bound	Time	of Nodes
3-12.a	104	9	101	12.2	101.00	17.0	11
3-12	107	1.2	104	1.5	104.00	2.1	5
5-15.a	128	1.2	115	499	113.00	3600	961
5-15	145	0.6	130	621	122.00	3600	1223
5-20.a	177	156	165	3234	156.00	3600	982
5-20	157	348	151	3555	135.66	3600	1012
6-24	262	183	249	952	224.3	3600	976
7-30	-	-	-	-	205.8	3600	816

Table 6: Redundant modeling

The solution process is driven by a standard branch-and-cut algorithm for the MIP model (cuts are only generated at the root node). At each node of the branch-and-cut tree, we build a CP problem with the following constraints :

- the constraints (28) – (33) (with improved bounds),
- precedence relations corresponding to sequencing variables $y_{i_1 i_2}$ that have been fixed to 1 in the MIP,
- the linear constraint $\sum_{i=1}^{n} c_{i,mach_i} \leq v^* - 1$, where v^* is the cost of the best current integer solution of the MIP.

This CP problem is handled in the following way. We first do propagation and fix corresponding 0-1 variables in the MIP part. Then we start search with a time limit of 3 seconds, trying to find a better feasible solution.

Tab. 6 contains the computational results. Clearly, the current implementation of redundant modeling is not competitive with the hybrid branch-and-cut algorithm. However, we conjecture that this approach can be improved by calling CP only at some of the nodes of the branch-and-cut tree. Further research and computational experiments are needed to find a strategy for selecting nodes that are promising for CP.

6 Conclusion

We introduced a hybrid branch-and-cut approach for MIP problems with monotone constraints that can be handled by CP. On a generic multiple machine scheduling problem, this algorithm performs better than an iterative MIP/CP approach or redundant modeling.

When the problem size grows, the performance of all approaches decreases rapidly. The branch-and-cut algorithm with cardinality cuts seems to be efficient for instances where, for $F(x) = 0$, the number of nonzero components of x is small. If this number gets larger, cardinality cuts are no longer tight. But, it is still possible to get feasible solutions of good quality.

Acknowledgement

This work was partially supported by the European Commission, Growth Programme, Research Project LISCOS – Large Scale Integrated Supply Chain Optimisation Software, Contract No. G1RD-CT-1999-00034. We thank our partners from the LISCOS project, in particular Abder Aggoun and Susanne Heipcke for their assistance in preparing the computational experiments.

References

[1] A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathl. Comput. Modelling*, 17(7):57 – 73, 1993.

[2] N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathl. Comput. Modelling*, 20(12):97 – 123, 1994.

[3] A. Bockmayr and T. Kasper. Branch-and-infer: A unifying framework for integer and finite domain constraint programming. *INFORMS J. Computing*, 10(3):287 – 300, 1998.

[4] Y. Colombani and S. Heipcke. Mosel: en extensible environment for modeling and programming solutions. In *4th International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems, CP-AI-OR'02, Le Croisic, France*, pages 277–290, 2002.

[5] F. Focacci, A. Lodi, and M. Milano. Cutting planes in constraint programming: An hybrid approach. In *Principles and Practice of Constraint Programming, CP'2000, Singapore*, pages 187 – 201. Springer, LNCS 1894, 2000.

[6] J. N. Hooker, G. Ottosson, E. S. Thorsteinsson, and H.-J. Kim. On integrating constraint propagation and linear programming for combinatorial optimization. In *Sixteenth National Conference on Artificial Intelligence, AAAI'99*, pages 136–141, 1999.

[7] V. Jain and I. E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS J. Computing*, 13(4):258 – 276, 2001.

[8] G. Ottosson, E. S. Thorsteinsson, and J. N. Hooker. Mixed global constraints and inference in hybrid CLP–IP solvers. *Annals of Mathematics and Artificial Intelligence*, 34(4):271–290, 2002.

[9] P. Refalo. Linear formulation of constraint programming models and hybrid solvers. In *Principles and Practice of Constraint Programming, CP'2000, Singapore*, pages 369 – 383. Springer, LNCS 1894, 2000.

[10] E. S. Thorsteinsson. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In *Principles and Practice of Constraint Programming, CP'2001, Paphos, Cyprus*, pages 16–30. Springer, LNCS 2239, 2001.

[11] L. Wolsey. *Integer programming.* Wiley, 1998.