A Comparison of CP, IP and Hybrids for Configuration Problems

Mats Carlsson
matsc@sics.se

Greger Ottosson
greger@csd.uu.se

April 1999

SICS Technical Report T99/04

Abstract

We investigate different solution techniques for solving a basic part of configuration problems, namely linear arithmetic constraints over integer variables. Approaches include integer programming, constraint programming over finite domains and hybrid techniques. We also discuss important extensions of the basic problem and how these can be accommodated in the different solution approaches.

ISSN 1100-3154
ISRN:SICS-T–99/04-SE

1 Introduction

This report investigates two classes of Integer Programming problems, which often come up in configuration problems. The first class (IP) can generally be described in Integer Programming form as

$$\begin{array}{ll} \min & cx \\ \text{s.t.} & Ax \geq b \\ & x \geq 0, x \text{ integer} \end{array}$$

with the additional knowledge that c, A and b have positive integer coefficients, and $c > 0$. The second class (IP+) is an extension of the first where we have additional logical or symbolic constraints, $C_1 \wedge \ldots \wedge C_n$:

$$\begin{array}{ll} \min & cx \\ \text{s.t.} & Ax \geq b \\ & x \geq 0, x \text{ integer} \\ & C_i \quad \forall i \end{array}$$

The aim of this report is to evaluate different algorithms for the above problems with respect to several criteria.

Performance How efficiently can these problems be solved to optimality or satisfaction?

Flexibility Is the algorithm general enough to deal with the additional constraints C with maintained performance?

This paper is outlined as follows. Section 1 discusses the available techniques, such as constraint programming, integer programming, cuttting planes, preprocessing, etc. Section 2 then applies and evaluates these techniques to a set of instances of our first, pure problem (IP). Section 3 and 4 concludes this paper and discusses future work, including how the topic of how these techniques extend to IP+, which yet is mainly to explore.

2 Solution Techniques

The general techniques applied here are

Integer Programming using branch-and-bound and Linear Relaxations,

Constraint Programming using constraint propagation, and

Hybrid approaches where CP and LP are combined, either with a CP- or IP-style search.

In addition to these techniques, there are various ways to tighten the linear relaxation prior to search: *preprocessing*, which tries to remove rows and columns, and to adjust remaining coefficients; and *cutting-plane* techniques, which try to generate strong valid inequalities.

2.1 IP branch-and-bound

Let LP $= \{\min cx : Ax \geq b, x \geq 0\}$, that is IP with the integrality restrictions removed, $\bar{x}$ the best integer solution found so far, and $c\bar{x}$ its objective value. Then, for a node in the branch-and-bound tree, do

1. Solve LP, with optimal (possibly fractional) solution vector v.
2. If LP is infeasible, skip this node.
3. If $cv \geq c\bar{x}$ then this node is suboptimal, skip this node.
4. If cv is integral, then let $\bar{x} = v$ since we have a new best solution, and skip this node.
5. Otherwise, branch on each variable $x_i \in x$ with fractional value v_i, creating new nodes $\{\min cx : Ax \geq b, x \geq 0, x_i \leq \lfloor v_i \rfloor\}$ and $\{\min cx : Ax \geq b, x \geq 0, x_i \geq \lceil v_i \rceil\}$.

This search can either be depth-first, breadth-first or something in between. A common scheme is *best-first*, which picks a non-explored (with feasible, fractional LP solution) node with the best LP value.

What characterizes this method is that it's heavily based on the linear relaxation. If the linear relaxation is strong, i.e. $cx_{LP} \sim cx_{IP}$, where x_{LP} and x_{IP} is the optimal LP and IP solution respectively, then this method is very efficient. If the linear relaxation is weak, i.e. $cx_{LP} \ll cx_{IP}$, then one has to try to fix this by adding cutting planes. A cutting plain is a valid inequality for IP that cuts of part of the feasible region of LP, thus strengtening the linear relaxation.

Another very important fact about branch-and-bound with linear relaxations is that it very effectively takes the objective functions into account and thus is good at quickly finding feasible integral solutions that are close to the optimal IP solution.

2.2 Constraint Programming over Finite Domains

In Finite Domain Constraint Programming each integer variable x_i has an associated *domain* D_i, which is the set of possible values this variable can take on in the (optimal) solution. The cartesian product of the domains, $D_1 \times \ldots \times D_n$, forms the solution space of the problem. This space is finite and can be searched exhaustively for a feasible or optimal solution, but to limit this search constraint propagation is used to infer infeasible solutions and prune the corresponding domains. From this viewpoint, CP operates on the set of possible solutions and narrows it down.

It is in general harder to use the objective function effectively in constraint programming — it does not come for free from the use of a linear relaxation as in IP branch-and-bound. Also, the bounds derived on the objective function from constraint propagation are usually much weaker than the ones provided by a linear relaxation.

One main benefit of constraint programming is that any constraint — linear, non-linear or symbolic — that can infer infeasible values and project this on the domains can be combined with any other such constraint. Constraint programming, not restricted to a linear formulation of the problem, often allows for compact models. High-level abstractions of problem constraints can be encoded with special-purpose constraints such as those for scheduling, allocation and permutation.

2.3 Constraint Programming over Rationals

Constraint logic programming over rational (or reals) is basically an embedding of the Simplex algorithm for linear programming in a logic programming framework. It's incremental, which means that it's kept consistent at all times, and there's rudimentary support for mixed-integer programming and non-linear equations.

SICStus has two variants, one operating over real values (R) and one over rational (Q); the latter is the one used here, mainly because of rounding error and numerical instability problems encountered in the former. SICStus (Q) cannot compete with commercial solvers for pure LP/IP problems, this is partly due to the incrementality and the domain being rational values.

2.4 Cutting Planes

We have experimented with two kinds of cutting planes. The first procedure is a heuristic for producing cuts of Chvátal-Gomory rank 1, and the second type of cuts are more specifally the Gomory mixed-integer cuts.

2.4.1 Chvátal-Gomory rank 1 cuts

We use a heuristic to generate cuts of Chvátal-Gomory rank 1 [10, Sect. II.1]. For any linear combination, $u \geq 0$,

$$\lceil u^T A \rceil x \geq \lceil u^T b \rceil$$

is a valid inequality. The heuristic is as follows: For each row i, u is chosen with $u_i > 0$ and $u_{i'} = 0$ for $i' \neq i$. For each row i in column j, $A_{ij} > \gcd(A_i, b_i)$ gives rise to an inequality $u_i = 1/A_{ij}$.

Example:
Given the inequality

$$2x_1 + 4x_2 + 6x_3 \geq 10$$

we derive the two inequalities with $u = (\frac{1}{4})$ and $u = (\frac{1}{6})$, respectively

$$\begin{aligned} x_1 + x_2 + 2x_3 &\geq 3 \\ x_1 + x_2 + x_3 &\geq 2 \end{aligned}$$

$\square$

2.4.2 Gomory Cuts

The Gomory mixed-integer (MIG) cuts [10, p. 250] are derived using a solved linear relaxation in a node. Although general enough to cover MIP, they can also be used for pure IP problems. Given a row of the LP simplex tableau as $x_0 + \sum_{j \in N} a_j x_j = b$ where x_0 is a basic variable and N the indeces for the non-basic variables (some of which might be slack variables), the cut is effective for each fractional b as

$$\sum_{j \in N: f_j \leq f_0} f_j x_j + \frac{f_0}{(1 - f_0)} \sum_{j \in N: f_j > f_0} (1 - f_j) x_j \geq f_0$$

where $f_j = a_j - \lfloor a_j \rfloor$ for $j \in N$, and $b = \lfloor b \rfloor + f_0$.

Example:
Given a row of the simplex tableau

$$x_0 + 2.2x_1 + 3.6x_2 = 0.5$$

we derive a MIG cut as

$$\begin{aligned} &0.2x_1 + \frac{0.5}{(1 - 0.5)}(1 - 0.6)x_2 \geq 0.5 \Longleftrightarrow \\ &0.2x_1 + 0.4x_2 \geq 0.5 \end{aligned}$$

$\square$

These cuts can potentially be applied to any node in the search, and valid within and in any child of that node, as well as derived iteratively. In our tests below, the cut is only derived in the root node in a single iteration.

In general, and commonly, these cuts have fractional coefficients even if, as in our case, the original problem only have integral coefficients. Currently, SICStus (FD) does not allow fractional values in arithmetic FD constraints, so these cuts are safely rounded (i.e. $\sum \lceil a_j \rceil x_j \geq \lfloor b \rfloor$) before added as such. This will in most cases seriously weaken the cuts, but might be improved if coefficients were scaled before rounding. However, the scaling is limited by the maximum and minimum size of integers available in the system.

2.5 Relaxing FD constraints

For the basic problem (IP), the common linear relaxation, strengthened with cuts, is sufficient. For our extended problem, (IP+), we need to find linear relaxations for our additional constraints C to be able to use pure IP branch-and-bound search.

We will here consider two kinds of nonlinear constraints; logical constraints and binary relations.

2.5.1 Logical constraints

It is relatively easy to form a linear relaxation for logical relations over 0-1 integer variables using arithmetic.

Example:
Given 0-1 variables a and b, we can express logical relations as follows.

$$a \vee b \equiv a + b \geq 1$$

□

This can be done systematically, given that the needed logical 0-1 variables exist. Quite often, however, we need to tie a 0-1 variable to a general linear inequality, that is *reify* the inequality. While reification usually denotes an equivalence $C \Leftrightarrow B$ between constraint C and boolean variable B, the corresponding method in IP relies on the introduction of "big-M" constraints which enforce implication $B \Rightarrow C$. In general, given an inequality $a_i x \geq b_i$, a big-M formulation

$$a_i x + M(1 - B) \geq b_i$$

expresses $B \Rightarrow a_i x \geq b_i$ (given a sufficiently large constant M).

Example:
The logical relation

$$x_1 \geq 1 \quad \vee \quad x_2 \leq 0$$

is translated into linear form as follows

$$\begin{aligned} &a + b \geq 1 \\ &x_1 + 1(1 - a) \geq 1 \\ &x_2 \leq 0 + M(1 - b) \end{aligned}$$

where a, b are 0-1 integer variables and $x_1, x_2 \geq 0$. M should be a constant larger than the upper bound of x_2. □

Expressing logical relations in the linear relaxation of course increases the size of LP, but more importantly, the new variables are 0-1 integer variables which (1) must be constrained to represent the truth value of a linear inequality (through reification), and (2) must take integral values and thus be considered in the IP branch-and-bound search along with the original integer variables of IP.

If the amount of such additional logical constraints is limited, it should not be a problem in terms of performance. If relations become more complicated, there will be an increase in temporary 0-1 variables and a weaker relaxation.

It is unclear, for example, if the following constraint (from the truck configuration example in FD Obelics) can effectively be handled using linear relaxations.

```
(Chassi #= 1
#=>
FrameHeight in {1, 3}
#/\
Wheels in {1, 3, 4, 6}
#/\
Suspension#=3
#/\
Engine in {1,2}
#/\
Power #\=360
)
```

2.5.2 Binary relations

For two given variables, the `relation/3` constraint in SICStus (FD) defines a set of pairs of values which are feasible.

Example:
`relation(`X`,[1-{2,3},2-{1,5}],`Y`)` specifies that `<1,2>`,`<1,3>`,`<2,1>` and `<2,5>` are the only feasible combinations of values for X and Y □

We can achieve a linear relaxation for this constraints as before, i.e. create a temporary 0-1 integer variable for each assignment ($X = 1 \iff a_1$, $Y = 2 \iff b_1$, $Y = 3 \iff b_2$, etc), and then form

$$(a_1 \wedge b_1) \vee (a_1 \wedge b_2) \vee \quad \ldots$$

This will lead to $|D_X| + |D_Y| + |D_X| * |D_Y|$ new variables and inequalities, and might also weaken the relaxation too much to be tractable.

There might very well be better ways to relax this constraint, and this is a topic of further study.

2.6 Preprocessing

Three simple techniques are iterated until no more reductions are possible: adjustment of coefficients, elimination of subsumed rows, and elimination of redundant columns.

2.6.1 Adjustment of coefficients

Firstly, the right hand side b_i of any row $A_i x \geq b_i$ can be rounded up to the nearest multiple of the greatest common divisor of the left hand side coefficients.

Secondly, let U denote the upper bound of $b_i - A_i x$. Then any coefficient $A_{ij} > U$ can be replaced by U. Moreover, if $U \leq 0$, the row is entailed and can be removed.

For example, $2x_1 + x_2 + x_3 \geq 2$ can be tightened to $x_1 + x_2 + x_3 \geq 2$ if the lower bound of x_2 is known to be 1.

These are among the techniques described in [12].

2.6.2 Elimination of subsumed rows

An inequality $ax \geq b$ is subsumed by an inequality $a'x \geq b'$ if

$$a_i/b \geq a'_i/b' \quad \forall i$$

2.6.3 Elimination of redundant columns

Let A_i be the column of variable i and cx the objective function. Then a variable x_i is redundant (zero in some optimal solution) if

$\exists x_j$ s.t.

$$A_{ri} \leq \lfloor \frac{c_i}{c_j} \rfloor A_{rj} \quad \forall r$$

or, equivalently

$$\begin{array}{l} A_{rj} = 0 \Rightarrow A_{ri} = 0 \\ A_{rj} > 0 \Rightarrow \left\lceil \frac{A_{ri}}{A_{rj}} \right\rceil \leq \frac{c_i}{c_j} \end{array} \quad \forall r \tag{1}$$

Proof: Assume (1) and some feasible solution $x = v$ where $v_i > 0$. Let $\delta_j = \max_{\{r|A_{rj}>0\}} \left\lceil \frac{v_i A_{ri}}{A_{rj}} \right\rceil$. Then another feasible solution $x = v'$ is obtained as follows:

$$\begin{array}{l} v'_i = 0 \\ v'_j = v_j + \delta_j \\ v'_k = v_k, i \neq k \neq j \end{array}$$

Consider now the objective value cv' of the solution $x = v'$. We have that $c_j\delta_j \leq c_j v_i \max_{\{r|A_{rj}>0\}} \left\lceil \frac{A_{ri}}{A_{rj}} \right\rceil$. Hence and from (1) we obtain $c_j\delta_j \leq c_i v_i$. But since $cv' = cv - c_i v_i + c_j \delta_j$ we get $cv' \leq cv$.

Thus without loss of optimality, from any solution where $x_i > 0$ we can obtain another one where $x_i = 0$. □

2.7 Hybrids of IP and CP

There are several ways in which we can combine IP and CP. We can take the CP framework as a starting point and try to use a linear relaxation on the linear part of the problem. Sections 2.7.1 and 2.7.2 explore how linear programming can be expoited in this context. On the other hand, one can begin with a IP search, and add features from CP, which is described in Sect. 2.7.3.

2.7.1 Bounds Strengthening and Infeasibility Detection

If a linear relaxation is formed on the linear part of the problem, bounds can be propagated from the finite domain constraint store to the LP, and fixed variables can be derived by the LP[1]. This will in some cases improve the situation, partly because infeasibility can be detected earlier and also because the LP solution can be used, albeit a bit roughly, to guide the search. We will report on this approach below, which is similar to what has been done in [11].

The LP can also be used directly for domain reduction, where we minimize and maximize the LP with the objective function x_i for each variable. This quite effectively derives bounds which are projected on the variables' domains, but is generally very expensive. The next section describes a weaker but more efficient way of deriving bounds and doing domain reduction.

2.7.2 Reduced Costs Propagation

Provided for each variable x_i which is zero in the solution of linear program (i.e. in the final simplex tableau) are the *reduced costs*, $\bar{c}_i$. They indicate how much the objective function value would *increase* (per unit) if the variable were to take a non-zero value.

This information is used in some MIP packages (see e.g. [4]), as a way to strengthen the bounds of variables and thus strengthen the relaxation. It is, however, relatively new as applied in a CP context. In [3, 2], the reduced costs are introduced as inference mechanisms and in [5, 6] it is used for `all_different/1` and a relaxation of the assignment problem used in a TSP technique, respectively.

A similar technique could be tried to enhance a CP search with the more general $Ax \geq b$ as linear part.

2.7.3 Generalizing IP Search

A second starting point for integration of CP and IP is to add flexibility to an linear relaxation-based IP search through constraint propagation. The direct problem with this is that the domains, and thus the solution space, are not maintained and pruned in an IP branch-and-bound search to the same extent as it is in CP. An integral solution of the linear relaxation may not satisfy the additional constraints, which have to be handled, ultimately as extensions to the search. If not carefully crafted, this might mean loss of generality (new constraints implies changes in the search) of IP. This is still a topic of further research.

3 Experimental testing

3.1 Benchmark Problems

The algorithms have been tested on the following set of problems. Table 1 shows the original size of the problem, the number of CG and MIG cuts generated, and the LP relaxation and IP optimal values.

Table 2 shows the performance of a few established commercial and non-commercial LP/IP codes on the problem set. ILOG CPLEX is commercial [4], lp_solve is public domain [9], and Lindo is commercial [8]. The 'PP' column displays the problem size after preprocessing, and 'Time' is CPU solution time in milliseconds.

Experiments with MIP search parameters for CPLEX show relatively small variations. Default CPLEX node selection is best-bound; a depth-first search shows similar behavior with the exception of problem 15 with MIG cuts which is about 3 times slower. Changes in branching heuristics have a small impact; default variable selection differs from problem to problem, but maximum infeasibility is common. Strong branching (look-ahead with LP objective value to choose the most promising branch of a node) decreased the number of nodes of problem 7 (without MIG cuts) by a factor 10 and cut the time in half, but no noticeable effects on the other instances. Always branching up on the fractional variable selected showed some improvement in general, and branching down was somewhat worse. All in all, the performance varies with a factor 2 up or down depending on the heuristics, which is a fairly robust behavior.

3.2 CPLEX and preprocessing

Table 3 shows the performance for the benchmarks suite with CPLEX preprocessing turned off.

Problem	Constraints×Variables				IP$_{\text{opt}}$	LP$_{\text{opt}}$			
	Orig	CG	MIG	Pre-proc		Orig	CG	MIG	CG+MIG
1	1x3	3	0	4x3	60000	59999.4	59999.66	59999.39	59999.66
2	1x3	3	0	2x3	60000	60000.0	60000.0	60000.0	60000.0
3	3x5	5	3	6x5	29002	29000.0	29000.0	29002.0	29002.0
4	1x3	2	1	2x2	25	18.0	20.0	24.0	25.0
5	2x4	5	2	9x4	457	455.73	456.54	456.09	456.57
6	3x9	8	3	8x8	60035	60023.53	60027.16	60032.78	60033.53
7	4x6	5	2	5x5	601	600.0	600.0	600.5	600.5
8	1x4	1	0	1x1	15	14.375	15.0	14.375	15.0
9	2x12	9	2	6x11	840	758.33	840.0	762.5	840.0
10	2x63	18	2	9x62	626	600.5	611.0	616.71	616.71
11	1x12	12	1	6x12	196	192.30	196.0	195.99	196.0
12	6x12	14	4	4x3	3170	1005.18	2548.66	2555.27	2555.27
13	6x50	14	4	10x39	18020	17468.64	17923.66	17801.22	17923.66
14	6x50	14	3	7x39	16758	16116.16	16612.0	16448.74	16612.0
15	6x50	14	3	11x39	15635	14988.5	15585.33	15435.001	15585.33
16	5x32	12	2	5x14	3119	2037.33	3119.0	3119.0	3119.0
17	5x32	12	2	3x3	3119	1827.5	3119.0	3119.0	3119.0
18	5x32	12	2	3x4	3371	2546.83	3368.0	3371.0	3371.0

Table 1: Problem statistics

Problem		CPLEX						LP_solve		Lindo	
		Original			MIG			Original	MIG	Original	MIG
		PP	Time	Nodes	PP	Time	Nodes	Time	Time	Time	Time
1	1x3		10	0				950		0	
2	1x3		0	0				0		0	
3	3x5		0	2		0	0	10	0	10	10
4	1x3		0	2	1x3	10	2	0	0	0	0
5	2x4		10	8		10	2	120	120	10	10
6	3x9		0	3	5x9	0	3	7720	0	10	10
7	4x6		250	1290		10	5	3560	2	160	10
8	1x4	1x3	0	0				380		0	
9	2x12		30	99	4x12	20	73	60	080	10	10
10	2x63		40	78		10	1	40	0	50	40
11	1x12		10	19		0	5	10	0	10	10
12	6x12	3x7	10	1	5x7	10	1	0	10	20	10
13	6x50	5x50	10	13	6x50	0	10	311980	222330	60	70
14	6x50	5x50	20	25	6x50	20	27	417280	269480	100	110
15	6x50	5x50	0	4	6x50	180	522	1705270	150580	20	40
16	5x32	1x14	0	0	1x14	0	0	0	0	10	10
17	5x32	1x14	0	0	1x14	10	0	10	10	10	10
18	5x32	1x14	0	0	1x14	0	0	0	0	10	10

Table 2: Benchmark results with commercial and public domain LP/IP codes.

Problem	CPLEX, with preprocessing				CPLEX, no preprocessing			
	Original		MIG		Original		MIG	
	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
1	10	0			0	0		
2	0	0			0	0		
3	0	2	0	0	10	2	0	0
4	0	2	10	2	10	3	0	2
5	10	8	10	2	0	8	10	2
6	0	3	0	3	10	6	0	3
7	250	1290	10	5	260	1290	0	5
8	0	0			10	1		
9	30	99	20	73	10	73	10	73
10	40	78	10	1	30	78	10	1
11	10	19	0	5	0	19	0	5
12	10	1	10	1	0	9	10	2
13	10	13	0	10	880	2564	230	677
14	20	25	20	27	1180	3443	1550	4185
15	0	4	180	522	250	727	190	537
16	0	0	0	0	10	14	0	0
17	0	0	10	0	10	13	0	0
18	0	0	0	0	0	12	0	1

Table 3: Effect of preprocessing in CPLEX

3.3 Approaches in SICStus (FD)

A pure CP approach, using only finite domain constraints is not viable. In the following sections we'll explore some more powerful approaches.

3.3.1 Bounds propagation and infeasibility detection

Using LP for infeasibility detection improves the situation somewhat. This is a basic and weak hybrid approach – it simply consists of global constraint that propagates bounds from domains of FD variables, and checks satisfiability of the LP.

Table 4 shows the search results with binary search on the cost, followed by labeling of the x_i, with $Ax \geq b$ as FD constraints and as a relaxation in LP.

3.3.2 IP branch-and-bound in SICStus (FD/Q)

While still in the FD framework, but now using an IP-style branch-and-bound search, we can improve the results significantly. Table 5 shows the results of doing an branch-and-bound IP search in SICStus (FD/Q) with branching on fractional values of the solution to a series of linear relaxations.

The downside of this is not directly obvious. Note that the focus of the search is this time the linear relaxation; a feasible integral solution to LP is also a solution of IP, but not necessarily a solution to IP+. So we must either linearize IP+ to be able to use this search technique, or incorporate the satisfaction of the side constraints of IP+ in the search.

Problem	SICStus (FD/Q): FD + LP + PP + FD cost splitting search					
	Original			MIG		
	Time	BTs	Initial Domain	Time	BTs	Initial Domain
1	110	10	0..48670	120	10	0..48670
2	30	8	0..48680	20	8	0..48680
3	270	28	0..760023	100	8	0..760023
4	10	0	0..21	10	0	0..21
5	60	11	0..3143	70	11	0..3143
6	460	27	0..48908	270	11	0..48908
7	6680	1654	0..8812	100	14	0..8812
8	20	0	0..60	20	0	0..60
9	70	2	0..5306	110	2	0..5306
10	26840	1343	0..72934	20910	819	0..72934
11	100	2	0..3018	100	2	0..3018
12	63730	21199	420..13923	137970	21199	420..13923
13	186860	7132	1259..1372913	259990	7132	1259..1372913
14	612730	24354	1259..1240118	1029560	24354	1259..1240118
15	49330	2039	1259..1114769	62510	2039	1259..1114769
16	150	0	2577..78682	160	0	2577..78682
17	120	0	2577..72701	120	0	2577..72701
18	180	5	2577..85118	240	5	2577..85118

Table 4: Benchmark results for a finite domain search with domain splitting on cost.

Problem	SICStus (FD/Q): FD + LP + PP + IP search							
	Original		CG		MIG		CG+MIG	
	Time	BTs	Time	BTs	Time	BTs	Time	BTs
1	40	0	40	0	50	0	40	0
2	10	0	20	0	10	0	10	0
3	270	6	270	6	110	0	110	0
4	30	1	20	1	10	0	10	0
5	140	15	310	15	220	13	120	0
6	140	1	250	4	130	0	240	0
7	2630	741	2610	741	80	1	80	1
8	0	0	0	0	0	0	0	0
9	910	68	50	0	1270	69	130	0
10	4020	30	2420	1	1060	0	4850	3
11	200	2	200	3	110	0	230	3
12	110	6	40	0	30	0	50	0
13	230410	2842	1710	9	227300	1313	2260	9
14	764710	9838	2750	29	1028920	7995	5130	30
15	417510	5365	950	2	125140	1003	1150	2
16	250	5	120	0	70	0	120	0
17	90	5	40	0	20	0	50	0
18	90	3	90	1	30	0	50	0

Table 5: Benchmark results with IP branch-and-bound search in SICStus (FD/Q).

3.3.3 Hybrid CP/IP search

The algorithm of [11] is a compromise between IP branch-and-bound and FD-style search. It's a CP search in the sense that it reduces all domains to singletons during search, but an IP-style search because it centers the labeling around the values given by the linear relaxations.

Table 6 shows the benchmark results for an implementation of this algorithm in SICStus (FD/Q).

Problem	SICStus (FD/Q): FD + LP + PP + Hybrid search					
	Original		CG		CG+MIG	
	Time	BTs	Time	BTs	Time	BTs
1	70	0	80	0	120	0
2	50	0	50	0	40	0
3	410	64	420	64	330	50
4	30	3	40	4	20	0
5	170	31	150	16	170	8
6	320	39	540	46	410	14
7	8110	1679	8120	1679	250	28
8	10	0	0	0	0	0
9	700	77	140	0	260	0
10	13250	412	19960	240	20590	254
11	380	24	290	6	410	14
12	110	6	30	1	50	1
13	116360	4560	8850	285	10210	285
14	355100	14693	11640	490	15620	490
15	101700	4066	8150	223	9510	222
16	420	26	220	0	230	0
17	110	3	60	0	60	0
18	100	5	140	8	60	0

Table 6: Benchmark results with hybrid search in SICStus (FD/Q).

3.4 Satisfaction of IP

A second use of the constraint model discussed in this paper, apart from optimization, is satisfaction. The use of this is mostly to fix or constrain one or a few variables and then, possibly given a constrained objective function, check satisfaction of the problem.

To illustrate the efficiency one can expect for such a situation, we have compared the different solution techniques for obtaining solutions within 1%, 5% and 10% of the optimal value. The results are summarized in the Table 7.

4 Conclusion

For the pure problem (IP), an IP-style branch-and-bound search using linear relaxations is superior. The easier of the benchmarks are solvable in reasonable time also with an CP search, but breaks down when the problem grows. Hybrid search [11] is better than CP, but cannot compete with IP branch-and-bound.

For IP+, it is still unknown if we can find good linear relaxations for the additional logical and symbolic constraints so that a pure IP search can handle the problem.

Problem	SICStus (FD/Q): CG + MIG + PP + FD + LP + IP search						SICStus (FD/Q): CG + MIG + PP + FD + LP + FD bisect, ffc					
	1%		5%		10%		1%		5%		10%	
	Time	BT	Time	BT	Time	BT	Time	BT	Time	BT	Time	BT
1	30	0	40	0	40	0	100	7	240	10	220	6
2	40	0	30	0	40	0	80	7	170	8	200	9
3	100	0	100	0	100	0	200	6	200	7	210	5
4	10	0	10	0	10	0	10	0	10	0	20	0
5	70	0	80	1	80	1	120	2	210	11	140	2
6	160	1	160	1	160	1	180	10	290	6	340	6
7	60	1	50	1	60	1	110	7	120	7	120	4
8	0	0	10	0	0	0	0	0	0	0	0	0
9	130	0	140	0	140	2	160	1	160	0	160	0
10	1800	1	1910	2	1510	0	2110	3	2670	1	6600	72
11	120	1	100	0	90	0	140	1	150	2	160	1
12	30	0	30	0	30	0	40	0	40	0	40	0
13	6140	73	5350	72	5340	72	26560	534	482210	13197	140750	6658
14	2740	13	1870	10	1120	1	12260	345	15390	526	2800	70
15	12660	110	2220	10	1100	0	15340	251	3170	46	148750	5843
16	110	0	100	0	110	0	130	0	120	0	130	0
17	50	0	50	0	50	0	50	0	50	0	50	0
18	50	0	50	1	50	0	50	0	50	0	60	0

Table 7: Benchmark results for finding solution within 1%,5% and 10% of optimum.

Solving satisfiability instances of IP is easier for the CP search, but only marginally so for IP branch-and-bound.

A good commercial solver like CPLEX or Lindo is about a magnitude or two faster than a similar code in SICStus (Q). It is unclear whether this is due to CLP(Q), the link FD-Q or the search.

5 Future Work

The main future research direction is to continue to explore how IP+ can be solved efficently and flexibly. This involves defining the side constraints of IP+ and extend the techniques used the IP problem to handle them. For example, a linearization of the side constraints are needed for the pure branch-and-bound approach. We also need get hold of or create representative benchmarks problems.

There is also a set of minor research topics that should treated. We should investigate using other LP solvers than SICStus (Q) as backends, since there is some evidence that this would improve solution times. Another path to explore further is preprocessing and search strategies; both can be further refined and for example CPLEX or other IP solvers could be used as reference and comparison.

Extending SICStus (FD)'s arithmetic constraints to allow for real or rational coefficients would allow a more succinct use of e.g. the mixed-integer Gomory or any other fractional cuts to improve propagation. This would be a more succinct and precise alternative to scaling of these cuts.

Finally, there are also a couple of interesting new techniques for improved domain reduction. One is the reduced-cost propagation, which was outlined in Sec. 2.7.2, the other that might be interesting to explore is projection propagation [7]. As domain reduction techniques, they would primarily benefit the FD and hybrid searches investigated above, but might also prove valuable in the extension needed for branch-and-bound to handle the side constraints of IP+.

6 Acknowledgements

The research reported herein was funded in part by SICS with a grant from the Swedish National Board for Technical and Industrial Development (NUTEK), and in part by Tacton Systems AB. The benchmark data was kindly provided by Tacton Systems AB.

References

[1] H. Beringer and B. De Backer. Combinatorial problem solving in constraint logic programming with cooperating solvers. In C. Beierle and L. Plümer, editors, *Logic Programming: Formal Methods and Practical Applications*, Studies in Computer Science and Artificial Intelligence, chapter 8, pages 245–272. Elsevier, 1995.

[2] Y. Caseau and F. Laburthe. Solving various weighted matching problems with constraints. *Lecture Notes in Computer Science*, 1330:17–??, 1997.

[3] Yves Caseau and François Laburthe. Solving small TSPs with constraints. In Lee Naish, editor, *Proceedings of the 14th International Conference on Logic Programming*, pages 316–330, Cambridge, July8–11 1997. MIT Press.

[4] CPLEX. *CPLEX Manual*, 1998. URL http://www.cplex.com.

[5] Filippo Focacci, Andrea Lodi, and Michela Milano. Integration of cp and or methods for matching problems. In *CP-AI-OR'99 Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems.*

[6] Filippo Focacci, Andrea Lodi, Michela Milano, and Danielo Vigo. Solving tsp through the integration of or and cp techniques. In *CP98 Workshop on Large Scale Combinatorial Optimisation and Constraints.*

[7] Andreas Fordan. Linear projection in clp(fd).

[8] Lindo. *Super Lindo 5.3.* URL http://www.lindo.com.

[9] LP_Solve. *LP_Solve 2.2.* URL ftp://ftp.es.ele.tue.nl/pub/lp_solve.

[10] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley and Sons, New York, 1988.

[11] Robert Rodosek, Mark Wallace, and Mozafar Hajian. A new approach to integrating mixed integer programming and constraint logic programming. *Baltzer Journals*, 1997.

[12] M.W.P. Savelsbergh. Preprocessing and probing for mixed integer programming problems. *ORSA Journal on COmputing*, 6:445–454, 1994.