Cooperation of LP Solvers for Solving MILPs

Christine SOLNON
LISI - Université Claude Bernard and INSA de Lyon
43, bd du 11 novembre 1918 - Bat. 710
69622 Villeurbanne Cedex - France
e-mail: csolnon@lisihp1.univ-lyon1.fr

Abstract

A standard approach to solve Mixed Integer Linear Programs is to perform a global branch and bound search through all possible combinations. Due to the hardness of the problem, this search must be closely controlled by a constraint solver which uses constraints to prune the search space in an a priori way.

In this paper, we define a new domain reduction solver which uses in a cooperative way a set of Linear Programming solvers. The idea is to compute the actual range of values of the integer variables with respect to the continuous relaxation of the problem, and then narrow these domains to the closest integer interval. This narrowing is iteratively performed until a fixed point is reached where all domains are bound by integer values which belong to the continuous relaxation of the problem. This fixed point corresponds to a new partial consistency, which is stronger than the continuous relaxation and allows us to solve MILPs more efficiently.

1 Motivations

MILPs (Mixed Integer Linear Programs) have many industrial applications such as network design, crew scheduling, resource allocation or cutting stock problems. The general form of a MILP is displayed in figure 1.

$$\begin{array}{lll}
\text{minimize} & \sum_{j=1}^{n} c_j x_j & \\
\text{subject to} & \sum_{j=1}^{n} a_{ij} x_j \ \leq \ b_i & \forall i \in 1..m \\
 & l_j \leq x_j \leq u_j & \forall j \in 1..n \\
 & x_j \ \text{ integral} & \forall x_j \in X_{int}
\end{array}$$

Figure 1: General form of a MILP

In this MILP, X_{int} is the subset of variables that are constrained to have integer values. When dropping these integrality restrictions, the resulting linear program —called the continuous relaxation— can easily be solved (by using the simplex algorithm for instance). However, when integrality constraints must be taken into account, the problem becomes much harder (NP-complete).

A standard approach to solve MILPs is to perform a global branch and bound search [14] (also called branch and relax search in [11]). The idea is to solve a relaxation of the MILP and then, if some of the relaxed constraints are not satisfied, to split the MILP into sub-MILPs on which this process is repeated.

The relaxation step is performed at each node of the search in order to detect infeasibility and prune the search space in an a priori way. Most of the time, this relaxation step is performed by a Linear Programming solver (LP solver) which verifies that the continuous relaxation of the MILP is feasible. LP solvers are usually based on the Simplex algorithm[1] and can efficiently solve linear programming problems over the reals and the rationals[2] [8]. However, LP solvers are not able to take into account integrality constraints. These constraints are checked a posteriori, when the search tries to assign an integer value to a variable. As a consequence, many branches of the search tree are tediously explored while some integer variables cannot take integer values any more. In order to partially take into account integrality constraints, and reduce the search space, two main local techniques have been proposed : preprocessing and probing techniques, and domain reduction techniques.

Preprocessing and probing techniques

The main idea of preprocessing [21] is to consider each constraint, together with the lower and upper bounds of its variables, and
- drop the constraint if it is redundant,
- detect inconsistency if the constraint is infeasible or
- tigthen bounds on the variables if the constraint becomes infeasible when the variable is moved to that bound.

If after processing every constraint some variables have been fixed or some bounds have been improved, this process is repeated.

[1] The interior point algorithm, which is in average more efficient for solving large-scale linear programs, is not widely employed in branch and bound searches as it is less incremental than the simplex algorithm.

[2] However, LP solvers are not complete with floating point numbers, due to rounding errors.

Probing techniques [21] are used to look ahead one step further : a variable is temporarily set to one of its bounds and then preprocessing techniques are performed. If the problem becomes infeasible, then this value can be discarded ; if some other variable becomes fixed then a new constraint can be derived that tightens the continuous relaxation ; if a constraint becomes redundant, then its formulation can be tightened by coefficient reduction. Such probing techniques are usually performed on binary variables (0-1 variables) only.

Preprocessing and probing techniques have been incorporated in MINTO, a Mixed INTeger Optimizer [20]. The results indicate that these techniques can be very effective and can enhance the performance of a branch and bound search on many MILPs. However, on some MILPs these techniques are less effective and the performances are significantly degraded [14].

Domain reduction techniques

The idea of domain reduction is to successively use each constraint to reduce the domains of its variables until no reduction can be performed for any of the constraints. This fixed point corresponds to some local consistency, usually an approximation of arc-consistency [2]. In order to obtain stronger consistencies (like path-consistency), one can go one step further and look ahead to the remaining possible values of the variables and check whether the constraint can be satisfied with these values.

Domain reduction techniques are widely used to solve constraint satisfaction problems over finite domains (e.g., in CHIP [10], CC(FD) [12] and clp(FD) [9]) and over the reals (e.g., in Newton [3], Interlog [15] and Prolog IV [17]).

Berenger and De Backer defined in [4, 5] a cooperative solver which combines a solver based on domain reduction techniques with a LP solver. The domain reduction solver sends to the LP solver every new integer bound it has inferred, while the LP solver checks for the consistency of the continuous relaxation of the problem, within the bounds sent by the domain reduction solver. Such a cooperation allows them to solve some MILPs more efficiently, both in size of the search tree and time.

Local versus global techniques

Domain reduction techniques as well as basic preprocessing and probing techniques restrict their attention to information in a single constraint at a time. These local checks are computationally efficient but are not always very effective.

Some advanced preprocessing techniques that exploit information from more than one constraint at a time have been proposed [21]. These techniques take advantage of special substructures to reformulate them and tighten the continuous relaxation. They are more effective than basic local preprocessing but, as a counterpart, computationally less efficient.

In this paper, we define another global technique which tightens the continuous relaxation but which is not problem-specific as in advanced preprocessing techniques. The idea is to use LP solvers to narrow the variables domains. Indeed, LP solvers have a global view of the set of linear constraints and are able to compute the actual domains of the integer variables with respect to the continuous relaxation. These computed domains are then narrowed to the closest integer interval. This narrowing is performed until a fixed point is reached where all the domains are bound with integer values which belong to the continuous relaxation of the MILP. This fixed point corresponds to a new partial consistency, called Linear Integer Bounding (LIB) consistency. LIB-consistency is stronger than the continuous relaxation. It partially takes into account integrality constraints and is more effective to reduce the search space. As a counterpart, LIB-consistency is also more expensive to perform. However, first experimentations show that on many examples it allows us to solve MILPs more than twice faster.

2 Notations

The general MILP of figure 1 will be denoted by the triple (X_{int}, C, z) such that X_{int} is the set of integer variables, C the set of linear constraints (linear inequations and variables bounds) and z the objective function to be minimized :

$$
\begin{aligned}
C = \quad & \{ \textstyle\sum_{j=1}^{n} a_{ij} x_j \ \leq \ b_i \ , \quad \forall i \in 1..m \ \} \ \cup \\
& \{ \ l_j \leq x_j \leq u_j \ , \qquad\quad \forall j \in 1..n \ \} \\
z = \quad & \textstyle\sum_{j=1}^{n} c_j x_j
\end{aligned}
$$

The inequations that define the variables bounds are integrated within the set of linear constraints C. We shall note $D_C(x_i)$ the domain associated with a variable x_i in a set of constraints C, i.e., $D_C(x_i) = [l_i..u_i]$ if the constraint $l_i \leq x_i \leq u_i$ belongs to C.

Let v be a real value, we shall note :
- $\lceil v \rceil$ the smallest integer value such that $v \leq \lceil v \rceil$,
- $\lfloor v \rfloor$ the greatest integer value such that $\lfloor v \rfloor \leq v$.

3 Definition of LIB-consistency and LIB-narrowing

We propose to partially take into account integrality constraints in order to tighten the continuous relaxation and reduce the search space. The idea is to check that the bounds of each integer variable are integer values which belong to the set of solutions of the continuous relaxation of the problem. In other words, each facet of the hyper-box defined by the bounds of the integer variables must have a non empty intersection with the polyhedron defined by the continuous relaxation of the MILP. More formally, this partial consistency, called Linear Integer Bounding (LIB) consistency, is defined as follows :

Definition of LIB-consistency :

The MILP (X_{int}, C, z) is LIB-consistent if for each integer variable $x_j \in X_{int}$ such that $D_C(x_j) = [l_j..u_j]$,

1. the bounds l_j and u_j are integer values,
2. $C \cup \{x_j = l_j\}$ is feasible over the reals,
3. $C \cup \{x_j = u_j\}$ is feasible over the reals.

When solving a MILP, the domains of the integer variables are often much larger than the convex polyhedron defined by the continuous relaxation of the MILP, so that the problem is not LIB-consistent. In this case, one has to narrow some domains until the MILP becomes LIB-consistent. This narrowing has to be performed without removing any integer solution. More formally, a LIB-narrowing is defined as follows :

Definition of LIB-narrowing :

A LIB-narrowing of a MILP (X_{int}, C, z) is a MILP (X_{int}, C', z) such that :

1. $\forall x_i \in X_{int}, D_{C'}(x_i) \subseteq D_C(x_i)$,
2. C' and C have the same set of integer solutions,
3. (X_{int}, C', z) is LIB-consistent.

4 LIB-narrowing algorithm

LIB-narrowing is performed by the fixpoint algorithm —adapted from the Waltz algorithm [22]— described in figure 2. In this algorithm, Q contains the set of integer variables to be narrowed. When starting narrowing, Q is set to X_{int}. At each iteration, an integer variable x_i is extracted from Q and its associated LP solvers $min[x_i]$ and $max[x_i]$ respectively compute its lower and upper bounds l_f and u_f with respect to the continuous relaxation of the MILP[3]. If the corresponding integer interval $[\lceil l_f \rceil..\lfloor u_f \rfloor]$ is strictly included within the current domain of x_i, then these new bounds are added to C and the set Q of variables to be narrowed is re-initialized to $X_{int} - \{x_i\}$.

This process is repeated until either the continuous relaxation of the MILP becomes inconsistent, or a fixed point is reached (when Q is empty). Termination is ensured as there is a finite number of variables, the domains of which are finite : at each iteration, either a variable is removed from Q and no domain is changed, or the set Q of variables to be narrowed increases but the domain of an integer variable is reduced.

Proposition 1 : The set of constraints returned by `LIB-narrowing`(C) has the same set of integer solutions as C.

[3] Each LP solver is based on the Simplex algorithm. The implementation of LP solvers is discussed in section 5.

function `LIB-narrowing`(C: a set of constraints)
returns a set of constraints
  $Q \leftarrow X_{int}$
  while $Q \neq \emptyset$ **and** C is feasible over the reals **do**
    extract from Q an integer variable x_i
    such that $D_C(x_i) = [l_i..u_i]$
    $u_f \leftarrow max[x_i].optimize(C)$
    $l_f \leftarrow min[x_i].optimize(C)$
    if $l_i < \lceil l_f \rceil$ **or** $\lfloor u_f \rfloor < u_i$
    then
      $C \leftarrow C \cup \{\lceil l_f \rceil \leq x_i \leq \lfloor u_f \rfloor\}$
      $Q \leftarrow X_{int} - \{x_i\}$
    end if
  end while
  return(C)
end `LIB-narrowing`

where $min[x_i]$ and $max[x_i]$ are two LP solvers such that the function $optimize(C)$ of $min[x_i]$ (resp. $max[x_i]$) returns the minimum (resp. maximum) value of x_i w.r.t. the set of constraints C.

Figure 2: LIB-narrowing algorithm

Indeed, the algorithm only adds to C constraints like $\lceil l_f \rceil \leq x_i \leq \lfloor u_f \rfloor$. These constraints remove solutions over the reals such that $l_f \leq x_i < \lceil l_f \rceil$ or $\lfloor u_f \rfloor < x_i \leq u_f$. As these intervals do not contain any integer value, their suppression does not cut off any integer solution.

Proposition 2 : The set of constraints returned by `LIB-narrowing`(C) is either LIB-consistent, or inconsistent over the reals.

Indeed, the algorithm returns a set of constraints that is consistent over the reals when the set Q of variables to be narrowed is empty. As a consequence, during the last sequence of iterations, for each integer variable $x_i \in X_{int}$, we have $u_i = \lfloor u_f \rfloor$ and $l_i = \lceil l_f \rceil$. In other words, the bounds u_i and l_i of x_i are integer values which belong to the set of solutions of the continuous relaxation of the problem.

5 Implementation issues

LIB-narrowing has been implemented in a cooperative way. The program is composed of a monitor, which manages the set Q of variables to be narrowed, and for each integer variable $x_i \in X_{int}$, two LP solvers $min[x_i]$ and $max[x_i]$ which respectively are in charge of computing the lower and upper bounds of x_i.

Each LP solver $min/max[x_i]$ uses the simplex algorithm to compute the optimal value of x_i with respect to the current set of linear constraints. The computed optimal value is sent to the monitor, which constrains it to an integer value and then forwards it (if necessary) to all other LP solvers. Hence, each $min/max[x_i]$ solver receives numerous new bounds computed by other LP solvers, and has to update consequently the optimal value of x_i.

The key point concerns the way of handling these new bounds. Obviously, one cannot simply add to the simplex a new constraint, each time a new bound is received. Hence, we use *a simplex with explicit bounds* [8, 13] so that the new bounds are simply incorparated by modifying the corresponding bounds in the simplex. In some cases, these modifications lead to a simplex which is still in feasible form, and no further computations are needed. However, if the modified simplex is no longer in feasible form, some iterations have to be performed. In this case, we perform *a dual optimization* [8] so that the new bound is computed in an incremental way : usually, the new optimal value is very close to the last computed one and only few dual iterations are needed to reach the new optimum.

The simplex algorithm is complete over real and rational numbers, but not over floating point numbers, due to rounding errors. In order to minimize these rounding errors, we use a *revised simplex algorithm* which is numerically more stable [8, 18]. However some rounding errors can still occur. In a pragmatic way, we add (resp. subtract) a small value $\epsilon = 1.10^{-6}$ to each upper (resp. lower) bound computed by a LP solver. This practical solution actually prevents nearly all rounding errors. However, from a theoretical point of view, the only correct solution would be to use rational numbers.

LIB-narrowing is performed at each choice point of the search and the state of each LP solver has to be restored after each backtrack. A first solution would be to save the whole state of each solver at each choice point. This solution requires too much memory to be practical. Therefore, we only save the upper and lower bounds of each integer variable, and incrementally re-compute after each backtrack the state of the LP solver by a dual optimization of the simplex.

A prototype has been implemented in C. We used for the implementation of the LP solvers some modules of the `lp_solve` system [6].

6 Application of LIB-narrowing

We evaluate in this section the effectiveness and efficiency of LIB-narrowing : we solve MILPs by a branch and bound procedure and we compare the results obtained when the relaxation step is performed by a LIB solver (which verifies that the set of constraints is LIB-consistent) with the results obtained when the relaxation step is performed by a simple LP solver (which verifies that the continuous relaxation is consistent). This LP solver is implemented with the same algorithm as the LP solvers of the LIB solver, i.e., a revised dual simplex with explicit bounds.

6.1 Branch and bound algorithm

We compare the LP and LIB solvers for solving MILPs. We use in both cases the branch and bound search algorithm described in figure 3.

function `branch-and-bound`(X_{int}, C, z)
returns the solution of the MILP (X_{int}, C, z)
    $Solution \leftarrow \infty$
    $Q \leftarrow \{(X_{int}, C, z)\}$
    while $Q \neq \emptyset$ **do**
        extract from Q a MILP (X_{int}, C', z)
        if `relax`($C' \cup \{z < Solution\}$)=true
        then
            let $\{v_1, \ldots, v_n\}$ be the solution
            that minimizes z w.r.t. C'
            if $\forall x_i \in X_{int}$, v_i is an integer
            then
                /* *an integer solution has been found* */
                $Solution \leftarrow$ value of z corresponding
                        to $\{v_1, \ldots, v_n\}$
            else
                /* *creation of 2 new sub-MILPs* */
                choose a variable $x_i \in X_{int}$
                such that v_i is not an integer
                $C'_{low} \leftarrow C' \cup \{x_i \leq \lfloor v_i \rfloor\}$
                $C'_{up} \leftarrow C' \cup \{x_i \geq \lceil v_i \rceil\}$
                $Q \leftarrow Q \cup \{(X_{int}, C'_{low}, z), (X_{int}, C'_{up}, z)\}$
            end if
        end if
    end while
    return($Solution$)
end `branch-and-bound`

Figure 3: Branch and bound search algorithm

In this algorithm, Q is the set of MILPs to be solved. When starting the optimization, Q only contains the initial MILP. At each iteration, a MILP is extracted from Q. If its relaxation is consistent and the corresponding optimal solution does not meet all integrality constraints, then the MILP is splitted into two new sub-MILPs that are added to Q. In the current implementation, this set Q is organised as a stack, in a last-in/first-out strategy corresponding to a depth first exploration of the search tree.

Upon each iteration, the `relax` function is used to check that a relaxation of the selected MILP is feasible. This function depends on the solver chosen :

- when the LP solver is used, `relax`(C) returns "true" if the set of constraints C is feasible over the reals,
- when the LIB solver is used, `relax`(C) first LIB-narrows the variables domains by the domain reduction algorithm of figure 2, and then returns "true" if the resulting set of constraints is feasible over the reals.

The LIB and LP solvers are successively compared when using the two following heuristics for choosing the variable x_i to be splitted at each iteration :

- `max-fract` chooses the variable x_i such that v_i is the value the fractional part of which is the closest to 0.5, i.e., x_i is the variable that has the largest integer infeasibility.
- `min-dom` chooses the variable x_i that has the smallest domain, i.e., the variable which should "first-fail".

6.2 Test set

We compare the LIB and LP solvers on problems extracted from the test set of real-world MILPs described in [7]. These problems are respectively described in figure 4 by means of the number of variables over the reals (R), the number of general integer variables (I), the number of binary integer variables (B) (the domains of which are {0,1}), the number of constraints (C) and the solution of the MILP.

Some of these MILPs (egout, p0033, pp08a, pp08aCUTS, rgn and stein27) only involve binary integer variables. Hence, these problems have not been solved with the `min-dom` heuristic (as all domains contain two values). The p0033-10 problem is the same problem than p0033 but where the domains of the variables have been relaxed to $\{0, ..., 10\}$.

6.3 Results

Figures 5 and 6 respectively display the results obtained when using the `max-fract` and the `min-dom` heuristic. Each figure successively displays the number of choice points, the number of dual simplex iterations and the cpu time (in seconds, on a HP 715/100 station). For each of these criteria, the first column indicates the results obtained when using the LIB solver,

Problem	Variables R	I	B	C	Optimal solution
bell3a	62	32	39	123	878 430
bell5	36	28	30	91	8 966 406
egout	94	0	55	98	568
flugpl	6	11	0	17	1 201 500
gt2	0	164	24	29	21 166
p0033	0	0	33	14	3 089
p0033-10	0	19	14	14	2 455
pp08a	176	0	64	136	7 350
pp08aCUTS	176	0	64	246	7 350
rgn	80	0	100	24	82
stein27	0	0	27	118	18

Figure 4: Problems description

the second when using the LP solver and the third one gives the ratio between the LP and the LIB solvers. The fourth array gives results for the harder problems that were not solved after more than 30 minutes of cpu time. For these problems, the execution has been stopped at 30 minutes of cpu time and we display the best solution found and the difference between the value of the best solution found and the optimal value (as a percentage of the optimal value).

On these examples, one can remark that LIB narrowing actually tightens the continuous relaxation and drastically reduces the number of performed choice points. However, the results are not uniform and the effectiveness of LIB-narrowing depends on the MILP considered and the heuristic used : on some problems, like bell3a with the max-fract heuristic or flugpl with the min-dom heuristic, LIB-narrowing is very effective and allows one to perform more than 200 times less choice points ; on other problems, like egout or p0033, it reduces the number of performed choice points by a factor of nearly 100 ; finally, on the two problems rgn and stein27, it is not so effective and the LIB solver only performs five times less choice points than the LP solver.

However, there is a trade off between efficiency and effectivity. Indeed, if the LIB solver is actually effective to prune the search space, it is also more expensive than a simple LP solver : at each choice point, the LIB solver usually performs successive dual optimizations while the LP solver only performs one. Each dual optimization itself requires successive iterations of the Simplex. Hence, at each choice point, the LIB solver performs an average of more than 100 iterations while the LP solver only performs 4 iterations. As a consequence, the overall efficiency of LIB narrowing depends on its degree of effectivity : when LIB-narrowing reduces the number of choice points by more than 100 times, the number of performed iterations is more than twice less. However, on the rgn and stein27 problems, on which LIB narrowing is not so ef-

Number of choice points :

Problem	LIB	LP	$\frac{LP}{LIB}$
bell3a	4 407	892 414	202.5
p0033	28	4 276	152.7
p0033-10	509	52 641	103.4
egout	384	33 090	86.2
flugpl	212	17 513	82.6
rgn	460	2 450	5.3
stein27	520	2 604	5.0

Number of simplex iterations :

Problem	LIB	LP	$\frac{LP}{LIB}$
bell3a	437 781	4 530 351	10.3
p0033	6 225	20 956	3.4
p0033-10	89 143	248 644	2.8
egout	67 923	76 393	1.1
flugpl	36 859	40 922	1.1
rgn	516 617	91 979	0.2
stein27	230 994	30 806	0.1

Cpu time, in seconds :

Problem	LIB	LP	$\frac{LP}{LIB}$
bell3a	1 192.9	6 165.3	5.2
p0033	2.4	7.1	3.0
p0033-10	33.5	83.0	2.5
egout	100.8	158.4	1.6
flugpl	10.4	18.2	1.8
rgn	245.6	42.1	0.2
stein27	417.9	73.7	0.2

Problems stopped at 30 mn cpu time :

Problem	LIB		LP	
	last solution	gap (%)	last solution	gap (%)
bell5	8969078	0.1	9377729	4.6
pp08a	8290	6.5	9760	25.4
pp08aCUTS	7930	7.9	9130	24.2
gt2	101266	378.4	103520	389.1

Figure 5: Results with the `max-fract` heuristic

Number of choice points :

Problem	LIB	LP	$\frac{LP}{LIB}$
bell5	11 788	1 487 699	126.2
flugpl	31	15 003	484.0
bell3a	2 991	122 667	41.0
p0033-10	593	28 344	47.8

Number of simplex iterations :

Problem	LIB	LP	$\frac{LP}{LIB}$
bell5	755 969	11 429 974	15.1
flugpl	6 344	28 789	4.5
bell3a	144 373	1 215 710	8.4
p0033-10	53 755	126 252	2.4

Cpu time, in seconds :

Problem	LIB	LP	$\frac{LP}{LIB}$
bell5	1 692.4	10 892.4	6.4
flugpl	2.4	15.2	6.3
bell3a	409.3	1 498.7	3.7
p0033-10	21.7	42.3	2.0

Problem stopped at 30 mn cpu time :

Problem	LIB		LP	
	last solution	gap (%)	last solution	gap (%)
gt2	106 378	402.6	135 468	540.0

Figure 6: Results with the `min-dom` heuristic

fective and only performs 5 times less choice points, the LIB solver is not efficient and performs 5 times more iterations.

Cpu time essentially depends on the number of iterations performed. Hence, for the rgn and stein27 problems LIB narrowing is five times slower, whereas for all other problems, it is twice or more faster (even 6 times faster for the bell5 and flugpl problems with the `min-dom` heuristic).

In general, the `min-dom` heuristic is more efficient than the `max-fract` one, for both the LIB and LP solvers. However, the improvement is slightly more important for the LIB solver than for the LP one. This could be explained by the fact that LIB narrowing dynamically narrows domains during the resolution.

Finally, one can remark that for harder problems like gt2 or pp08a, that have been stopped after 30 minutes of cpu time, the best solution found with the LIB solver is better than the one found with the LP solver.

7 Related works

Branch and cut approaches

Branch and cut approaches are often used to solve MILPs. The idea is to add, at each step of the search, some new linear constraints which cut off the current optimal solution of the continuous relaxation, but which do not remove any integer solution. The key point in this approach is to compute "good" cuts, i.e., cuts that are as close as possible to the convex hull of all integer solutions. In the general case, this appears to be as hard as the resolution of the MILP. However, on some specific problems with restricted types of constraints, this approach allows one to solve MILPs efficiently (see for example CLP($\mathcal{PB}$) [1]). A critical point of the branch and cut approach lies in the fact that, when adding cuts, the MILP becomes larger and typically harder to solve.

The LIB-narrowing algorithm defined in this paper can be viewed as a branch and cut approach : each domain reduction cuts off some solutions over the reals while preserving the integer ones. These cuts can be deduced independently from the kind of constraints and are rather easily computed, provided that one uses an incremental simplex algorithm. Moreover, these cuts can be taken into account in a straightforward way without increasing the size of the simplex (i.e., the number of constraints), provided that one uses a Simplex with explicit bounds.

Domain narrowing and basic preprocessing techniques can also be viewed as branch and cut approaches. In these techniques, the cuts are deduced in a local way, by considering each constraint separately. These cuts are more easily computed but, as a counterpart, they are less effective.

Non linear problems over the reals

We define in [19] an active bounding linear solver, based on a similar cooperative architecture as the LIB solver. This solver cooperatively uses a set of LP solvers to compute the actual range of values of variables with respect to some linear constraints. This cooperative linear solver is integrated with a domain reduction solver into a concurrent architecture. This concurrent cooperative system —named `CCC`— is used to solve over the reals mixed sets of constraints (containing linear equations and inequations and non-linear equations).

8 Further works

When using a constraint solver to prune a search tree, a major efficiency issue lies in the choice of the completeness of the solver : one has to break the tie between the larger cost of a more complete solver and the smaller search space that such a solver can give rise to. Hence, if LIB-propagation drastically reduces the search space, it is also very expensive, both in time and space. However, first experimentations show that it can solve MILPs more efficiently than a simple LP solver.

A major interest of this approach lies in the fact that it could be parallelized in a straightforward way : each LP solver $min/max[x_i]$ can run independently, and only has to communicate to the other solvers the computed optimal values. Hence, we intend to implement the LIB-narrowing algorithm on a parallel machine, in order to compare such a AND-parallelism with the more usual OR-one (which consists in exploring different branches of the search tree in parallel) [16].

Thanks to Jean-Marc Fouet, Simone Pimont and Michel Rueher for reading successive versions of this paper.

References

[1] P. Barth and A. Bockmayr. Finite domain and cutting plane techniques in CLP($\mathcal{PB}$). In Leon Sterling, editor, *Proceedings of ICLP'95*, pages 133–147, Kanagawa, Japan, 1995. MIT Press.

[2] F. Benhamou. Interval Constraint Logic Programming. In Andreas Podelski, editor, *Constraint Programming: Basics and Trends*, volume 910 of *LNCS*, pages 1–21. Springer-Verlag, 1995.

[3] F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) Revisited. In *Proceedings of ILPS'94*, pages 124–138, Ithaca, NY, USA, 1994.

[4] H. Beringer and B. De Backer. Combinatorial problem solving in constraint logic programming with cooperative solvers. In C. Beierle and L. Plume, editors, *Logic Programming: formal methods and practical applications*. Elsevier Science Publisher, 1994.

[5] H. Beringer and B. De Backer. Cooperative solvers and global constraints: the case of linear arithmetic constraints. In *ILPS Workshop*, 1996.

[6] M. Berkelaar. lp_solve. Technical report, available at `ftp://ftp.es.ele.tue.nl/pub/lp_solve`, 1996.

[7] R.E. Bixby, E.A. Boyd, and R.R. Indovina. MIPLIB: a test set of real-world mixed integer programming problems. *SIAM News*, 25(2):16, 1992. (available at ftp://softlib.cs.rice.edu/pub/miplib).

[8] V. Chvatal. *Linear Programming*. W.H. Freeman and Co, 1983.

[9] D. Diaz. *Etude de la compilation des langages logiques de programmation par contraintes sur les domaines finis: le système clp(FD)*. PhD thesis, INRIA, 1995.

[10] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proceedings of the International Conference on Fith Generation Computer Systems*, 1988.

[11] P. Van Hentenryck. Constraint solving for combinatorial search problems. In *PPCP'95: Conference on Principles and Practice of Constraint Programming*. MIT Press, 1995.

[12] P. Van Hentenryck., V. Saraswat, and Y. Deville. Design, implementation and evaluation of the constraint language cc(FD). In *PPCP'95: Conference on Principles and Practice of Constraint Programming*. MIT Press, 1995.

[13] C. Holzbaur. A specialized, incremental solved form algorithm for systems of linear inequalities. Technical Report TR-94-07, University of Vienna, 1994.

[14] E. Johnson, G. Nemhauser, and M. Savelsbergh. Progress in integer programming: an exposition. Research report, Georgia Institute of Technology, 1997.

[15] O. Lhomme. Consistency techniques for numeric CSPs. In R. Bajcsy, editor, *Proceedings of the 13th IJCAI*, pages 232–238, Chambéry, France, 1993. IEEE Computer Society Press.

[16] B. Mans and C. Roucairol. Performances des algorithmes branch and bound parallèles à stratégie meilleur d'abord. Research Report RR-1716, INRIA, 1992.

[17] PrologIA. PrologIV — Constraint inside (handbook of PrologIV). Technical report, PrologIA, Parc Technologique de Luminy, Case 919, 13288 Marseille Cedex 09, France, 1996.

[18] P. Refalo and P. Van Hentenryck. CLP($\mathcal{R}_{lin}$) revised. In *ICLP'96: Joint International Conference on Logic Programming*. MIT Press, 1996.

[19] M. Rueher and C. Solnon. Concurrent cooperating solvers over reals. *Journal of Reliable Computing, Kluwer academic publishers*, 3(3), 1997.

[20] M. Savelsbergh and G. Nemhauser. Functional description of MINTO, a Mixed INteger Optimizer. Technical Report COC-91-03A, Georgia Institute of Technology, 1993.

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

[22] D.L. Waltz. Generating semantic descriptions from drawing of scenes with shadows. Technical Report AI-TR-271, MIT Cambridge, 1972.