Local Branching

Matteo Fischetti*, Andrea Lodi°

*DEI, University of Padova, Via Gradenigo 6/A, 35100 Padova, Italy
°DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
e-mail: matteo.fischetti@unipd.it, alodi@deis.unibo.it

May 2002, Revised December 2002

Abstract

The availability of effective exact or heuristic solution methods for general Mixed-Integer Programs (MIPs) is of paramount importance for practical applications. In the present paper we investigate the use of a generic MIP solver as a black-box "tactical" tool to explore effectively suitable solution subspaces defined and controlled at a "strategic" level by a simple external branching framework. The procedure is in the spirit of well-known local search metaheuristics, but the neighborhoods are obtained through the introduction in the MIP model of completely general linear inequalities called *local branching* cuts.

The new solution strategy is exact in nature, though it is designed to improve the heuristic behavior of the MIP solver at hand. It alternates high-level strategic branchings to define the solution neighborhoods, and low-level tactical branchings to explore them. The result is a completely general scheme aimed at favoring early updatings of the incumbent solution, hence producing high-quality solutions at early stages of the computation.

The method is analyzed computationally on a large class of very difficult MIP problems by using the state-of-the-art commercial software ILOG-Cplex 7.0 as the black-box tactical MIP solver. For these instances, most of which cannot be solved to proven optimality in a reasonable time, the new method exhibits consistently an improved heuristic performance: in 23 out of 29 cases, the MIP solver produced significantly better incumbent solutions when driven by the local branching paradigm.

1 Introduction

Mixed-integer linear programming plays a central role in modeling difficult-to-solve (NP-hard) combinatorial problems. However, the exact solution of the resulting models often cannot be carried out for the problem sizes of interest in real-world applications, hence one is interested in effective heuristic methods.

Although several heuristics have been proposed in the literature for specific classes of problems, only a few papers deal with general-purpose MIP heuristics, including [1], [2], [7], [8], [9], [11], [12], [15], [16], and [20] among others.

Exact MIP solvers are nowadays very sophisticated tools designed to hopefully deliver, within acceptable computing time, a provable optimal solution of the input MIP model, or at least a heuristic solution with a practically-acceptable error. In fact, what matters in many practical cases is the possibility of finding reasonable solutions as early as possible during the computation. In this respect, the "heuristic behavior" of the MIP solver plays a very important role: an aggressive solution strategy that improves the incumbent solution at very early stages of the computation is strongly preferred to a strategy designed for finding good solutions only at the late steps of the computation (that for difficult problems will unlikely be reached within the time limit).

Many commercial MIP solvers allow the user to have a certain control on their heuristic behavior through a set of parameters affecting the visit of the branching tree, the frequency of application of the internal heuristics, the fact of emphasizing the solution integrality rather than its optimality, etc. Unfortunately, in some hard cases a general-purpose MIP solver may prove not adequate even after a clever tuning, and one tends to quit the MIP framework and to design ad-hoc heuristics for the specific problem at hand, thus loosing the advantage of working in a generic framework.

In this paper we investigate the use of a general-purpose MIP solver as a black-box "tactical" tool to explore effectively suitable solution subspaces defined and controlled at a "strategic" level by a simple external branching framework. The procedure is in the spirit of well-known local search metaheuristics, but the neighborhoods are obtained through the introduction in the MIP model of (invalid) linear inequalities called *local branching* cuts. This allows one to work within a perfectly general MIP framework, and to take advantage of the impressive research and implementation effort that nowadays is devoted to the design of MIP solvers.

The new solution strategy is exact in nature, though it is designed to improve the heuristic behavior of the MIP solver at hand. It alternates high-level strategic branchings to define solution neighborhoods, and low-level tactical branchings (performed within the MIP solver) to explore them. The result can then be viewed as a two-level branching strategy aimed at favoring early updatings of the incumbent solution, hence producing improved solutions at early stages of the computation.

The paper is organized as follows. In Section 2 we outline a well-know heuristic scheme based on variable fixing (diving), and propose a variant of the scheme intended at overcoming its drawbacks. This leads naturally to the concept of "local branching" cuts, which are formally defined in Section 3. The basic method is then improved upon in Section 4, with the aim of enhancing its heuristic behavior through appropriate diversification mechanisms borrowed from local search paradigms. The resulting scheme is then analyzed computationally, in Section 5, on a large class of very difficult MIP problems[1]. We used the state-of-the-art commercial software ILOG-Cplex 7.0 [6] as the black-box "tactical" MIP solver. For these instances, most of which cannot be solved to proven optimality in a reasonable time, the new method exhibits consistently an improved heuristic performance: in 23 out of 29 cases, the MIP solver produced significantly better incumbent solutions when driven by the local branching paradigm. Some conclusions are finally drawn in Section 6.

An early version of the present paper was presented by the authors at the Sixth Workshop on Combinatorial Optimization held in Aussois, January 6-12, 2002.

2 Soft vs. hard variable fixing heuristics

A commonly used, and often effective, heuristic scheme fitting into the framework described in the introduction is based on the so-called *(hard) variable fixing* or *diving* idea, that can be described as follows. We assume to have an exact or heuristic black-box solver for the problem at hand. The solver is first applied to the input data, but its parameters are set so as to quickly abort execution and return a (possibly infeasible) "target solution" $\bar{x}$. This solution is defined, e.g., as the solution of the root-node Linear Programming (LP) relaxation, possibly after

[1] Instances available at `http://www.or.deis.unibo.it/research_pages/ORinstances/MIPs.html`

a clever rounding of some of its fractional variables, or as any heuristic solution of the problem. Solution $\bar{x}$ is then analyzed, and some of its nonzero variables are heuristically rounded-upito the nearest integer (if non-integer) and then fixed to this value. The method is then iteratively re-applied on the restricted problem resulting from fixing: the black-box solver is called again, a new target solution is found, some of its variables are fixed, and so on. In this way the problem size reduces after each fixing, hence the black-box solver can concentrate on smaller and smaller "core problems" with increasing chances of solving them to proven optimality.

A critical issue in variable-fixing methods is of course related to the choice of the variables to be fixed at each step. As a matter of fact, for difficult problems high-quality solutions are only found after several rounds of fixing. On the other hand, wrong choices at early fixing levels are typically very difficult to detect, even when bounds on the optimal solution value are computed before each fixing: in the hard cases, the bound typically remains almost unchanged for several fixings, and increases suddenly after an apparently-innocent late fixing. Therefore one has to embed in the scheme backtracking mechanisms to recover from bad fixings, a very difficult task.

The question is then how to fix a relevant number of variables without losing the possibility of finding good feasible solutions. To better illustrate this point, suppose one is given a heuristic 0-1 solution $\bar{x}$ of a pure 0-1 MIP model with n variables, and wants to concentrate on a core subproblem resulting from fixing to 1 at least 90% (say) of its nonzero variables. How should one choose the actual variables to be fixed? Put in these terms, the question lends itself to a simple answer: just add to the MIP model a linear *soft fixing* constraint of the form

$$\sum_{j=1}^{n} \bar{x}_j \; x_j \geq \lceil 0.9 \sum_{j=1}^{n} \bar{x}_j \rceil \tag{1}$$

and apply the black-box solver to the resulting MIP model. In this way one avoids a too-rigid fixing of the variables in favor of a more flexible condition defining a suitable neighborhood of the current target solution, to be explored by the black-box solver itself. In the example, the underlying hypothesis is the 10% of slack left in the right-hand side of (1) drives the black-box solver as effectively as fixing a large number of variables, but with a much larger degree of freedom–hence better solutions can be found.

Soft fixing is used as a refining tool in the crew-scheduling software TURNI [14, 22]. Here, the basic black-box solver is a specialized crew-scheduling heuristic based on a set partitioning model solved by Lagrangian relaxation and (hard) variable fixing, in a vein similar to the one proposed in [5] for the solution of pure set covering problems. The soft-fixing refining procedure is used on top of the basic TURNI module, and considers constraint (1) with respect to the best incumbent solution $\bar{x}$. Different percentage values in the right-hand side of (1) are considered, for each of which the basic heuristic is re-applied in the attempt of producing improved solutions. The overall approach proved quite effective for the solution of large-scale crew scheduling instances. For instance, we addressed the solution a real-world crew scheduling instance involving the scheduling of 800+ drivers to cover 8000+ time-tabled trips, as provided to us by NS Reizigers−the Dutch railways company. The experiment was run on a PC AMD Athlon 2100+ with 512 Mbyte of RAM. The basic TURNI module found, after 41 CPU minutes, a solution involving 833 duties. Starting from this reference solution, the soft-fixing refining procedure was applied, which produced a sequence of improvements converging to a solution with 809 duties after 55 minutes from the refining start.

3 The local branching framework

The soft fixing mechanism outlined in the previous section leads naturally to the general framework described in the sequel. We consider a generic MIP with 0-1 variables of the form:

$$
\begin{aligned}
(P) \qquad & \min c^T x && (2)\\
& Ax \geq b && (3)\\
& x_j \in \{0,1\} \qquad \forall j \in \mathcal{B} \neq \emptyset && (4)\\
& x_j \geq 0,\ \text{integer} \quad \forall j \in \mathcal{G} && (5)\\
& x_j \geq 0 \qquad \forall j \in \mathcal{C} && (6)
\end{aligned}
$$

Here, the variable index set $\mathcal{N} := \{1, \cdots, n\}$ is partitioned into $(\mathcal{B}, \mathcal{G}, \mathcal{C})$, where $\mathcal{B} \neq \emptyset$ is the index set of the 0-1 variables, while the possibly empty sets $\mathcal{G}$ and $\mathcal{C}$ index the general integer and the continuous variables, respectively.

Given a feasible *reference solution* $\bar{x}$ of (P), let $\overline{S} := \{j \in \mathcal{B} : \bar{x}_j = 1\}$ denote the binary support of $\bar{x}$. For a given positive integer parameter k, we define the *k-OPT neighborhood* $\mathcal{N}(\bar{x}, k)$ of $\bar{x}$ as the set of the feasible solutions of (P) satisfying the additional *local branching constraint*:

$$\Delta(x, \bar{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \overline{S}} x_j \leq k \qquad (7)$$

where the two terms in left-hand side count the number of binary variables flipping their value (with respect to $\bar{x}$) either from 1 to 0 or from 0 to 1, respectively.

In the relevant case in which the cardinality of the binary support of any feasible solution of (P) is a constant, this constraint can more conveniently be written in its equivalent "asymmetric" form

$$\sum_{j \in \overline{S}} (1 - x_j) \leq k' \ (= k/2) \qquad (8)$$

The above definition is consistent, e.g., with the classical k'-OPT neighborhood for the Traveling Salesman Problem (TSP), where constraint (8) allows one to replace at most k' edges of the reference tour $\bar{x}$.

As its name suggests, the local branching constraint can be used as a branching criterion within an enumerative scheme for (P). Indeed, given the incumbent solution $\bar{x}$, the solution space associated with the current branching node can be partitioned by means of the disjunction

$$\Delta(x, \bar{x}) \leq k \quad \text{(left branch)} \quad \textbf{or} \quad \Delta(x, \bar{x}) \geq k + 1 \quad \text{(right branch)} \qquad (9)$$

As to the neighborhood-size parameter k, it should be chosen as the largest value producing a left-branch subproblem which is likely to be much easier to solve than the one associated with its father. The idea is that the neighborhood $\mathcal{N}(\bar{x}, k)$ corresponding to the left branch must be "sufficiently small" to be optimized within short computing time, but still "large enough" to likely contain better solutions than $\bar{x}$. According to our computational experience, the choice of k is seldom a problem by itself, in that values of k in range $[10, 20]$ proved effective in most cases. We face here a situation similar to the one arising when using k'-OPT exchanges for TSP problems: even though the value of k' should in principle depend on the size and structure of the TSP instance at hand, a fixed "sufficiently small" constant value for k' is successfully used in practice. (Of course, this is also due to the fact that, the TSP neighborhood being searched by complete enumeration, a value of k'

larger than 3 would be impractical in most cases.) Moreover, as explained in the next section, the value of k can easily be increased/decreased automatically at run time, in an adaptive way.

A first implementation of the local branching idea is illustrated in Figure 1, where the triangles marked by the letter "T" (for Tactical) correspond to the branching subtrees to be explored through a standard "tactical" branching criterion such as, e.g., branching on fractional variables–i.e., they represent the application of the black-box exact MIP solver.

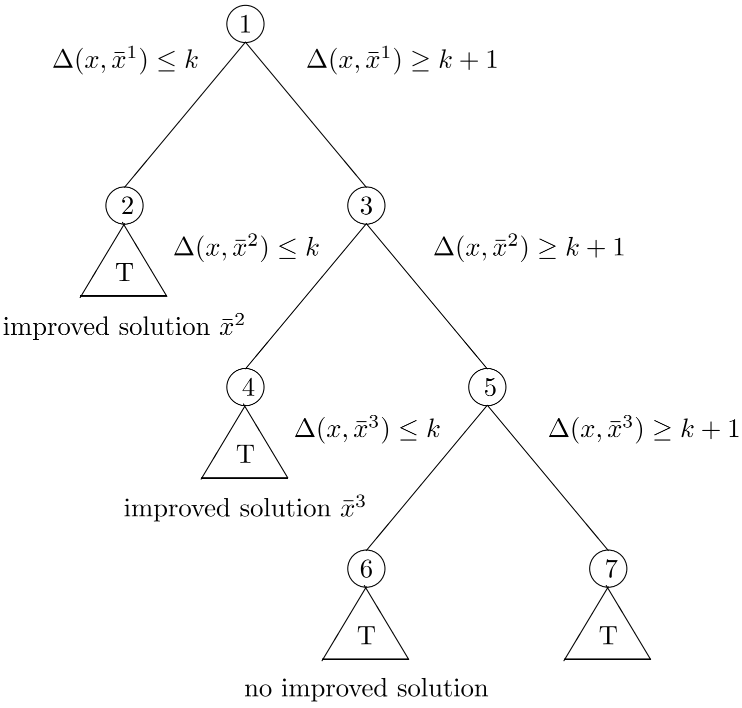

Figure 1: The basic local branching scheme.

In the figure, we assume to have a starting incumbent solution $\bar{x}^1$ at the root node 1. The left-branch node 2 corresponds to the optimization within the k-OPT neighborhood $\mathcal{N}(\bar{x}^1, k)$, which is performed through a tactical branching scheme converging (hopefully in short computing time) to an optimal solution in the neighborhood, say $\bar{x}^2$. This solution then becomes the new incumbent solution. The scheme is then re-applied to the right-branch node 3, where the exploration of $\mathcal{N}(\bar{x}^2, k) \setminus \mathcal{N}(\bar{x}^1, k)$ at node 4 produces a new incumbent solution $\bar{x}^3$. Node 5 is then addressed, which corresponds to the initial problem (P) amended by the two additional constraints $\Delta(x, \bar{x}^1) \geq k + 1$ and $\Delta(x, \bar{x}^2) \geq k + 1$. In the example, the left-branch node 6 produces a subproblem that contains no improving solution. In this situation the addition of the branching constraint $\Delta(x, \bar{x}^3) \geq k + 1$ leads to the right-branch node 7, which is explored by tactical branching. Note that the fractional LP solution of node 1 is not necessarily cut off in *both* son nodes 2 and 3, as is always the case when applying standard branching on variables. The same holds for nodes 3 and 5. In fact, the local branching philosophy is quite different from the standard one: we do not want to force the value of a fractional variable, but we rather instruct the solution method to explore first some promising regions of the solution space. The expected advantage of the local-branching scheme is an early (and more frequent) update of the incumbent solution. In other words, we expect to find quickly better and better solutions until we reach a point where local branching cannot be applied anymore (node 7, in the example), hence we have to resort to tactical branching to conclude the enumeration.

This behavior is illustrated in Figure 2, where we solved MIP instance `tr24-15` [23] by means of three codes: the commercial solver ILOG-Cplex 7.0 in the two versions emphasizing the solution *optimality* or *feasibility*, respectively, and the local branching scheme where ILOG-Cplex 7.0 (optimality version) is used to explore the "T-triangle" subtrees, and the local branching constraints are of type (7) with $k = 18$. Apart from the emphasizing setting, all the three codes were run with the same parameters. As to the initial reference solution $\bar{x}^1$ needed in the local branching framework, it was obtained as the first feasible solution found by ILOG-Cplex 7.0 (optimality version)−the corresponding computing time is included in the local-branching running time. The test was performed on Digital Alpha Ultimate Workstation 533 MHz. According to the figure, the performance of the local branching scheme is quite satisfactory, in that it is able to improve the initial solution several times in the early part of the computation. As a matter of fact, the local-branching incumbent solution is significantly better than that of the two other codes during almost the entire run. As to optimality convergence, the local branching method concludes its run after 1,878 CPU seconds, whereas ILOG-Cplex 7.0 in its optimization version converges to optimality within 3,827 CPU seconds (the feasibility version is unable to prove optimality within a time limit of 6,000 CPU seconds). Note, however, that the enhanced convergence behavior of the local branching scheme in proving optimality cannot be guaranteed in all cases–investigating this issue would require a different computational study, and is left to future research.

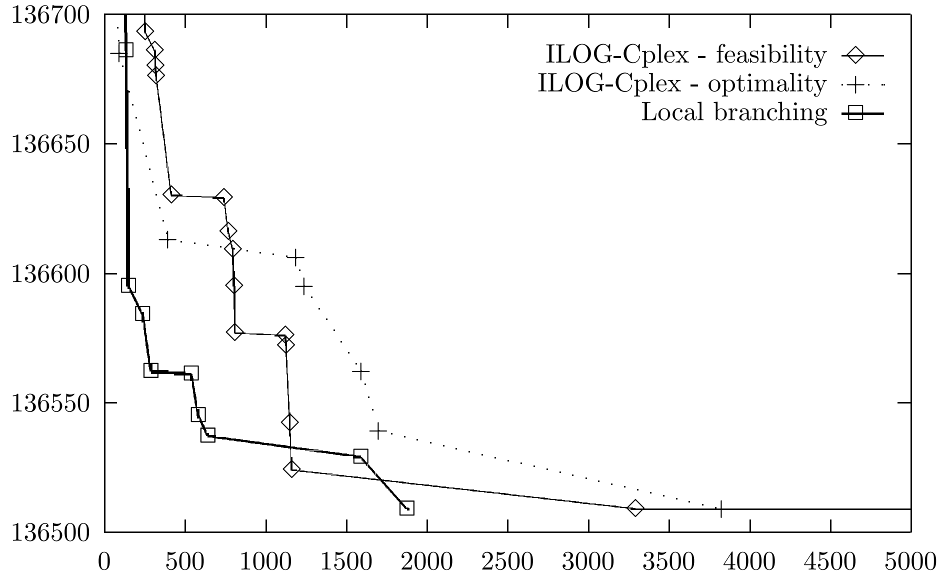

Figure 2: Solving MIP instance `tr24-15` (solution value vs. CPU seconds).

4 An enhanced heuristic solution scheme

We now elaborate the basic idea introduced in the previous section with the aim of enhancing its heuristic performance.

Imposing a time limit on the left-branch nodes

The first improvement is related to the fact that, in some cases, the exact solution of the left-branch node can be very time consuming for the value of the parameter k at hand. Hence, from the point of view of a heuristic, it is reasonable to impose a time limit for the left-branch computation. In case the time limit is exceeded, we have two cases.

(a) If the incumbent solution has been improved, we backtrack to the father node and create a new left-branch node associated with the new incumbent solution, without modifying the value of parameter k. This situation is illustrated in Figure 3, where node 3 actually has three sons: node 4, for which the time limit is reached with an improved solution $\bar{x}^3$, and the regular left- and right-branch nodes $4'$ and 5. Notice that, in the example, the neighborhood associated with node 4 was not explored completely, hence it would be mathematically incorrect to impose the associated right-branch condition $\Delta(x, \bar{x}^2) \geq k + 1$ on nodes $4'$ and 5.

(b) If the time limit is reached with no improved solution, instead, we reduce the size of the neighborhood in an attempt to speed-up its exploration. This is obtained by reducing the right-hand side term by, e.g., $\lceil k/2 \rceil$. This situation is illustrated in Figure 4, where again node 3 has three sons: node 4, for which the time limit is reached with no improved solution, node $4'$, for which the reduction of the neighborhood size allowed for finding a provable optimal solution $\bar{x}^3$ in the neighborhood, and node 5.

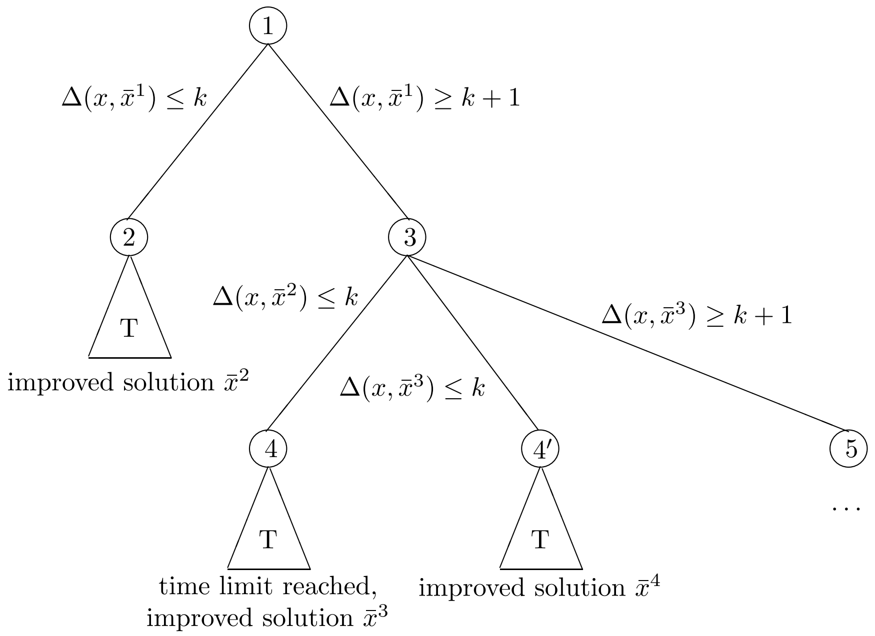

Figure 3: Working with a node time limit: case (a).

Diversification

A further improvement of the heuristic performance of the method can be obtained by exploiting well-known diversification mechanisms borrowed from local search metaheuristics. In our scheme, diversification is worth applying whenever the current left-node is proved to contain no improving solutions. This case arises

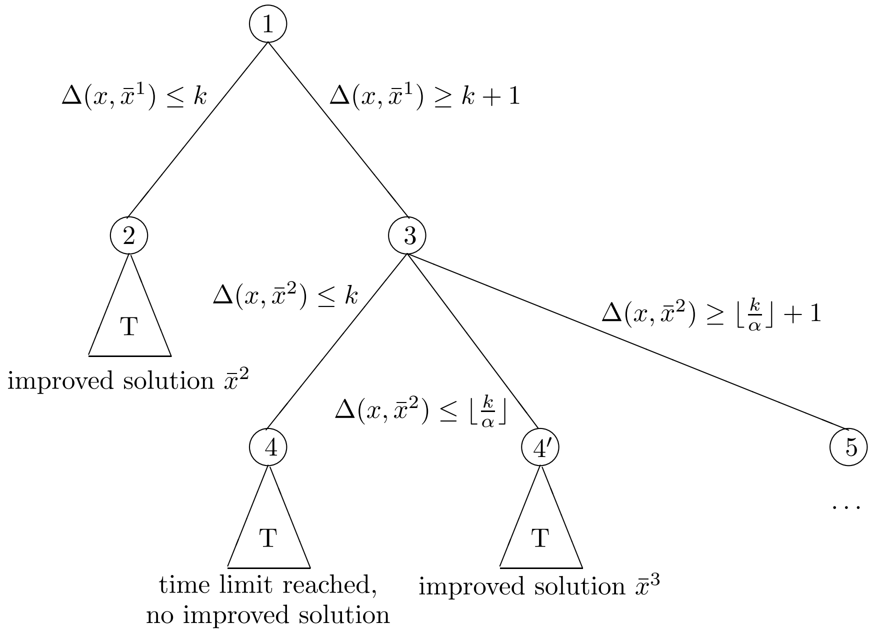

Figure 4: Working with a node time limit: case (b).

at node 6 in Figure 1, where the standard scheme would have switched to the exploration of node 7 through tactical branching. In order to keep a strategic control on the enumeration even in this situation, we use two different diversification mechanisms. We first apply a "soft" diversification consisting in *enlarging* the current neighborhood by increasing its size by, e.g., $\lceil k/2 \rceil$. Diversification then produces a left-branch node which is processed by tactical branching within a certain time limit. In case no improved solution is found even in the enlarged neighborhood (within the time limit), we apply a "strong" diversification step in the spirit of *Variable Neighborhood Search* [19], where we look for a solution (typically worse than the incumbent one) which is not "too far" from the current reference solution $\bar{x}$.

In our implementation, this is achieved by applying tactical branching to the current problem amended by constraint $\Delta(x, \bar{x}^2) \leq k + 2\lceil k/2 \rceil$, but without imposing any upper bound on the optimal solution value. The exploration is aborted as soon as the first feasible solution is found. This solution (typically worse than the current best one) is then used as the new reference solution, and the method is re-applied in an attempt to iteratively improve it, and eventually the incumbent one.

The overall enhanced scheme is illustrated by the pseudo-code in Figure 5. Function `LocBra` receives on input the neighborhood size (k), the overall time limit (`total_time_limit`), the time limit for each tactical branching exploration (`node_time_limit`), and the maximum number of diversifications allowed (`dv_max`). It returns on output the best/optimal feasible solution found (x^*) along with the final optimization status (*opt*). (In case no feasible solution has been found but there is no guarantee of infeasibility, `LocBra` will return *opt* = `false` and x^* = `undefined`.)

The method consists of a main repeat-until loop which is iterated until either the total time limit or the maximum number of diversifications is exceeded. At each iteration, a MIP problem is solved through the tactical black-box solver `MIP_SOLVE`

```
   function LocBra(k, total_time_limit, node_time_limit, dv_max, x*);
      rhs := bestUB := UB := TL := +∞; x* := undefined;
      opt := first := true; dv := 0; diversify := false;
      repeat
         if (rhs < ∞) then add the local branching constraint Δ(x, x̄) ≤ rhs endif;
         TL := min{TL, total_time_limit − elapsed_time};
         stat := MIP_SOLVE(TL, UB, first, x̃);
         TL := node_time_limit;
1.       if (stat = "opt_sol_found") then
            if (c^T x̃ < bestUB) then bestUB := c^T x̃; x* := x̃ endif;
            if (rhs ≥ +∞) return(opt);
            reverse the last local br. constraint into Δ(x, x̄) ≥ rhs +1;
            diversify := first := false; x̄ := x̃; UB := c^T x̃; rhs := k
         endif;
2.       if (stat = "proven_infeasible") then
            if (rhs ≥ +∞) return(opt);
            reverse the last local br. constraint into Δ(x, x̄) ≥ rhs +1;
            if (diversify) then UB := TL := +∞; dv := dv + 1; first := true endif;
            rhs := rhs +⌈k/2⌉; diversify := true
         endif;
3.       if (stat = "feasible_sol_found") then
            if (rhs < ∞) then
               if (first) then
                  delete the last local br. constraint Δ(x, x̄) ≤ rhs
               else
                  replace the last local br. constraint Δ(x, x̄) ≤ rhs by Δ(x, x̄) ≥ 1
               endif
            endif;
            REFINE(x̃);
            if (c^T x̃ < bestUB) then bestUB := c^T x̃; x* := x̃ endif;
            first := diversify := false; x̄ := x̃; UB := c^T x̃; rhs := k
         endif;
4.       if (stat = "no_feasible_sol_found") then
            if (diversify) then
               replace the last local br. constraint Δ(x, x̄) ≤ rhs by Δ(x, x̄) ≥ 1;
               UB := TL := +∞; dv := dv + 1; rhs := rhs +⌈k/2⌉; first := true
            else
               delete the last local br. constraint Δ(x, x̄) ≤ rhs;
               rhs := rhs −⌈k/2⌉
            endif;
            diversify := true
         endif;
      until (elapsed_time > total_time_limit) or (dv > dv_max);
      TL := total_time_limit − elapsed_time; first := false;
      stat := MIP_SOLVE(TL, bestUB, first, x*);
      opt := (stat = "opt_sol_found") or (stat = "proven_infeasible");
      return(opt)
   end.
```

Figure 5: The overall local branching function LocBra.

that receives on input three parameters: the local time limit TL, the upper bound UB used to interrupt the optimization as soon the best lower bound becomes greater or equal to UB, and the flag $first$ to be set to `true` for aborting the computation as soon as the first feasible solution is found. `MIP_SOLVE` returns on output the optimal/best solution $\widetilde{x}$ along the final optimization status $stat$.

`LocBra` uses an internal flag $diversify$ indicating whether the next required diversification will be of type "soft" ($diversify =$ `false`) or "strong" ($diversify =$ `true`). As a rule, a strong diversification is performed only if a soft one was performed in the previous iteration, i.e., any iteration that does not require diversification resets $diversify$ to `false`.

Four different cases may arise after each call to `MIP_SOLVE`:

1. `opt_sol_found`: the current MIP has been solved to proven optimality, hence the last local branching constraint is reversed, the reference solution $\bar{x}$ of value UB (and possibly the incumbent one x^* of value $bestUB$) is updated, and the process is iterated. Of course, an immediate return is performed in case this situation arises at the root node, where no local branching constraint is imposed ($rhs \geq +\infty$).

2. `proven_infeasible`: the current MIP is proven to have no feasible solution of cost strictly less than the input upper bound UB, hence the last local branching constraint is reversed, a soft or strong diversification is performed depending on the current value of flag $diversify$, and the process is iterated. Of course, an immediate return is performed in case this situation arises at the root node ($rhs \geq +\infty$), where no local branching constraint nor local time limit is imposed.

3. `feasible_sol_found`: a solution of cost strictly less than the upper bound UB (i.e., improving the reference solution) has been found, but the MIP solver was not capable of proving its optimality for the current problem (due to the imposed time limit or to the requirement of aborting the execution after the first feasible solution is found). As already mentioned, in this case we are not allowed to reverse the last local branching constraint. In order to still cut off the current reference solution $\bar{x}$, we replace the last local branching constraint $\Delta(x, \bar{x}) \leq rhs$ by the "tabu" constraint $\Delta(x, \bar{x}) \geq 1$ (unless this constraint has been already introduced at step 4, in which case the last local branching constraint is simply deleted). In this way, however, one could cut off also the optimal solution of the overall problem, as in principle we have no guarantee that $\bar{x}$ is actually optimal under the condition $\Delta(x, \bar{x}) \leq 0$, i.e., when fixing $x_j = \bar{x}_j$, $\forall j \in \mathcal{B}$. Such a guarantee always exists if, e.g., all the problem variables are binary ($\mathcal{G} = \mathcal{C} = \emptyset$). In the general case, the correctness of the method requires the use a "refining" procedure `REFINE(`$\widetilde{x}$`)` that replaces the input solution $\widetilde{x}$ by the optimal one (computed through the MIP solver) in the neighborhood $\Delta(x, \widetilde{x}) \leq 0$, thus producing a certificate of "optimality when fixing all binary variables" for $\widetilde{x}$. Notice that `REFINE` is implicitly applied to *every* solution $\widetilde{x}$ provided by `MIP_SOLVE`, though it is only invoked at step 3 (as this is clearly useless at step 1). Also worth noting is that procedure `REFINE` may turn out to be excessively time-consuming in the cases where fixing the binary variables does not lead to an easy-to-solve subproblem–e.g., due to the presence of a large number of general-integer variables. In this situation, one is allowed to deactivate the `REFINE` call, but the last local branching constraint $\Delta(x, \bar{x}) \leq rhs$ cannot be replaced by $\Delta(x, \bar{x}) \geq 1$ since this may affect the optimality of the method.

4. `no_feasible_sol_found`: no feasible solution of cost strictly less than UB has been found within the node time limit, but there is no guarantee that such a

solution does not exist. In this case, either the last local branching constraint is deleted or it is replaced by the "tabu" constraint $\Delta(x, \bar{x}) \geq 1$ depending on the type of diversification to be performed, i.e., on the value of flag $diversify$. In case of soft diversification, a new local branching neighborhood is addressed (and the previous local branching constraint deleted), whereas in a strong diversification the "tabu" constraint $\Delta(x, \bar{x}) \geq 1$ is introduced in order to escape from the current solution, and the upper bound UB is reset to $+\infty$.

On exit from the repeat-until loop, the remaining computing time (if any) is used in the attempt of solving the current MIP to proven optimality−this corresponds to the processing of node 7 in Figure 1. As a consequence, the local branching scheme acts as an exact method when (`total_time_limit` $= +\infty$ and) `dv_max` $< +\infty$, whereas for `dv_max` $= +\infty$ the overall scheme can be viewed as a local search heuristic. This latter execution mode is illustrated in Figure 6, where the hard MIP instance `B1C1S1` [23] is heuristically solved by `LocBRa` with the input parameters described in the next section.

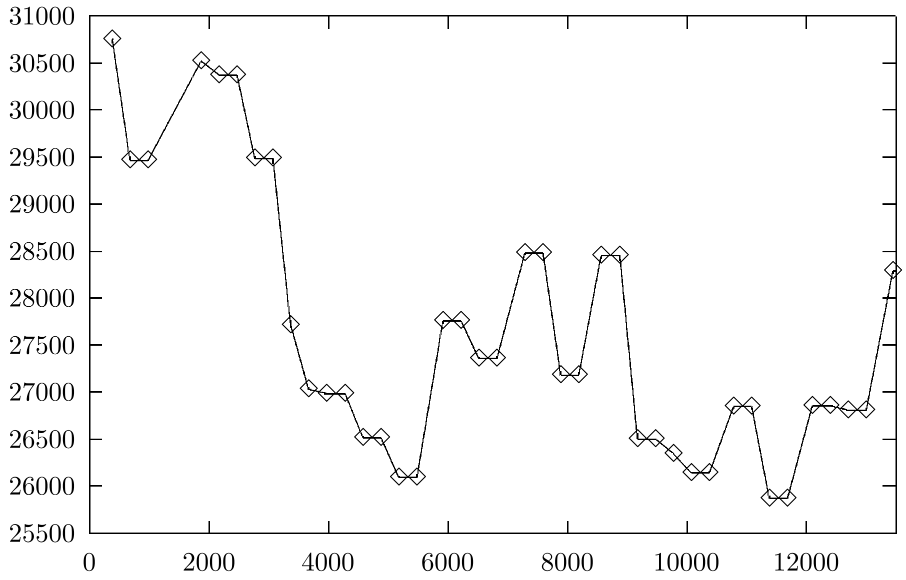

Figure 6: `LocBra` acting as a heuristic for instance `B1C1S1` (solution value vs. CPU seconds).

5 Computational results

The local branching procedure `LocBra` has been computationally tested on a large set of MIP instances containing 7 difficult instances from the MIPLIB 3.0 library [4], plus 22 very hard instances collected from several authors and originating in different optimization contexts. To be more specific, our test bed includes the following instances[2]:

- 7 MIPLIB-3.0 instances, `mkc`, `swath`, `danoint`, `markshare1`, `markshare2`, `arki001` and `seymour`;
- 1 network design instance, `net12`, provided by Belotti [3];

[2]Instances available at `http://www.or.deis.unibo.it/research_pages/ORinstances/MIPs.html`

- 2 crew scheduling instances, `biella1` and `NSR8K`, provided by Double-Click sas [22];
- 5 railway crew scheduling instances, `rail507`, `rail2536c`, `rail2586c`, `rail4284c` and `rail4872c`, discussed e.g. in Caprara, Fischetti and Toth [5]; in the instance name, the final letter "c" refers to the fact that the MIP model actually corresponds to a suitable "core problem" defined as in [5];
- 1 nesting instance, `glass4`, provided by Luzzi [17];
- 2 telecommunication network design instances, namely: instance `UMTS` provided by Polo [21], and instance `van` provided by Mannino and Parrello [18];
- 5 lot-sizing instances, `A1C1S1`, `A2C1S1`, `B1C1S1`, `B2C1S1` and `tr12-30`, discussed in Van Vyve and Pochet [23];
- 2 rolling stock and line planning instances, `roll3000` and `nsrand_ipx`, provided by Kroon [13];
- 4 railway line planning instances, `sp97ar`, `sp97ic`, `sp98ar` and `sp98ic`, proposed by Goossens, van Hoesel and Kroon [10].

For each instance we report in Table 1 the name (Name), a letter indicating the set (set), the size in terms of number of constraints (m), of total variables (n), and of binary variables ($|\mathcal{B}|$), the best available solution value (bestVal, to be discussed later), and the source of the instance in terms of context of application (context) and reference (ref.).

We compared the performance of three codes: the commercial solver ILOG-Cplex 7.0 in the two versions emphasizing the solution optimality (`cpx-O`) and feasibility (`cpx-F`), respectively, and the local branching procedure `LocBra` where ILOG-Cplex 7.0 (optimality version) is used at the tactical level.

None of the instances in our test bed can be solved to proven optimality by any of the three codes within a time limit of 5 CPU hours on a Digital Alpha Ultimate workstation 533 MHz, SPECint 16.1, SPECfp95 30.5. (A time limit of 10 hours has been given to instance `NSR8K` due to its very large size.) Thus, the solution values reported in the "BestVal" column of Table 1 refer to the best heuristic solution computed by the three codes within the time limit above.

Within `LocBra`, the local branching constraints are of type (7) with $k = 20$, and `dv_max` $:= +\infty$. As to the node time-limit, `node_time_limit`, it was set according to the instance size, as reported, in seconds, in column `node_tl` of Table 3. Moreover, for instance `arki001` we disabled the REFINE procedure invoked at Step 3 of `LocBra` as it turned out to be very time consuming due to the presence in the MIP model of a significant set of general-integer variables.

Table 2 compares the heuristic performance of the three methods (`cpx-O`, `cpx-F`, and `LocBra`) after 1, 3, and 5 hours of computation (more detailed information is reported in Tables 6-11 in the appendix). Column "Gap" reports either the percentage gap (%Gap, computed as $100 * (heuristic\_value - bestVal)/bestVal$) or the absolute gap (Abs. Gap, computed as $heuristic\_value - bestVal$) with respect to the best solution found by the three codes–as reported in column $bestVal$ of Table 1).

According to the table, the final `LocBra` heuristic solution ranked first in 23 out of the 29 instances in the test bed, whereas the two ILOG-Cplex versions `cpx-O` and `cpx-F` ranked first in 2 and 4 cases, respectively. Moreover, for many instances the `LocBra` incumbent solution is significantly better than that produced by the two other codes during the entire run. On the whole, the results clearly indicate the effectiveness of `LocBra` as a general-purpose heuristic for hard MIPs.

Name	set	m	n	$\|\mathcal{B}\|$	bestVal	context	ref.
mkc	A	3411	5325	5323	-559.51	MIPLIB 3.0	[4]
swath	A	884	6805	6724	471.03	MIPLIB 3.0	[4]
danoint	A	664	521	56	65.67	MIPLIB 3.0	[4]
markshare1	A	7	74	60	7.00	MIPLIB 3.0	[4]
markshare2	A	6	62	50	14.00	MIPLIB 3.0	[4]
arki001	A	1048	1388	415	7,581,034.85	MIPLIB 3.0	[4]
seymour	A	4944	1372	1372	424.00	MIPLIB 3.0	[4]
net12	B	14021	14115	1603	255.00	network design	[3]
biella1	C	1203	7328	6110	3,070,810.15	crew scheduling	[22]
NSR8K	C	6284	38356	32040	21,520,487.01	crew scheduling	[22]
rail507	D	509	63019	63009	175.00	railway crew scheduling	[5]
rail2536c	D	2539	15293	15284	691.00	railway crew scheduling	[5]
rail2586c	D	2589	13226	13215	957.00	railway crew scheduling	[5]
rail4284c	D	4287	21714	21705	1078.00	railway crew scheduling	[5]
rail4872c	D	4875	24656	24645	1556.00	railway crew scheduling	[5]
glass4	E	396	322	302	1,587,515,737.50	nesting	[17]
UMTS	F	4465	2947	2802	30,160,547.00	telecomm. network	[21]
van	F	27331	12481	192	5.09	telecomm. network	[18]
roll3000	G	2295	1166	246	13,065.00	railway rolling stock	[13]
nsrand_ipx	G	735	6621	6620	51,520.00	railway line planning	[13]
A1C1S1	H	3312	3648	192	11,834.02	lot-sizing	[23]
A2C1S1	H	3312	3648	192	11,251.10	lot-sizing	[23]
B1C1S1	H	3904	3872	288	25,869.15	lot-sizing	[23]
B2C1S1	H	3904	3872	288	26,297.63	lot-sizing	[23]
tr12-30	H	750	1080	360	130,596.00	lot-sizing	[23]
sp97ar	I	1761	14101	14101	667,735,390.40	railway line planning	[10]
sp97ic	I	1033	12497	12497	436,984,606.56	railway line planning	[10]
sp98ar	I	1435	15085	15085	531,942,554.88	railway line planning	[10]
sp98ic	I	825	10894	10894	449,915,159.36	railway line planning	[10]

Table 1: The hard MIP instances in the test bed.

Additional statistics on the `LocBra` execution are reported in Table 3, where for each 5-hour run[3] we report the number of occurrences of cases 1. ($stat =$ "`opt_sol_found`"), 2. ($stat =$ "`proven_infeasible`"), 3. ($stat =$ "`feasible_sol_found`"), and 4. ($stat =$ "`no_feasible_sol_found`") of Figure 5. Moreover, the table gives the total number of diversifications (dv), the value of the diversification counter when the best solution was found (dv_{best}), and the node time-limit (in seconds) we provided on input to `LocBra` (`node_tl`).

In the previous experiments we deliberately avoided any fine tuning of the `LocBra` parameters. However, the knowledge of the structure of the set of instances to be solved can obviously improve the results significantly. This is the case, in particular, when set covering/partitioning instances (possibly with side constraints) are addressed. Indeed, according to our experience the use of the "asymmetric" version (8) of the local branching constraint (7) results into improved results for this kind of instances. To illustrate this behavior, in Table 4 we report the results obtained for the set-covering instances `seymour` and `rail` when `LocBra` is driven by the local branching constraint (8) with $k' = 10$−corresponding roughly to $k = 20$ in (7); the improvements with respect to best value reported in Table 1 are marked by $*$.

Moreover, changing some main ILOG-Cplex parameter at the tactical level of `LocBra` may produce improved results in some cases. As an example, in Table 5

[3] For instance `NSR8K` the results in Table 2 refer to a 10-hour run.

Name	Gap	1 hour cpx-O	1 hour cpx-F	1 hour LocBra	3 hours cpx-O	3 hours cpx-F	3 hours LocBra	5 hours cpx-O	5 hours cpx-F	5 hours LocBra
mkc	%	3.765	2.399	2.281	3.765	1.294	0.404	3.765	1.294	0
swath	%	94.504	2.507	1.599	26.374	1.153	0	26.374	1.153	0
danoint	%	0	0	0	0	0	0	0	0	0
markshare1	Abs.	1	5	10	1	0	2	1	0	2
markshare2	Abs.	9	1	34	9	0	11	9	0	11
arki001	%	0.024	0.028	0	0.017	0.016	0	0.013	0.013	0
seymour	Abs.	11	5	2	11	3	0	11	3	0
net12	Abs.	–	–	–	41	–	41	41	0	41
biella1	%	0.256	31.313	0.241	0.256	18.272	0.172	0.256	11.475	0
NSR8K*	%	–	–	–	1843.8	1526.4	109.3	1426.8	1526.4	0
rail507	Abs.	2	6	1	2	5	0	1	5	0
rail2536c	Abs.	5	3	0	1	3	0	0	3	0
rail2586c	Abs.	54	34	7	14	34	2	14	34	0
rail4284c	Abs.	51	40	10	42	40	2	38	40	0
rail4872c	Abs.	73	69	27	70	49	9	45	49	0
glass4	%	22.835	7.087	19.685	14.226	7.087	10.236	13.386	7.087	0
UMTS	%	6.403	0	2.413	6.403	0	1.216	6.403	0	0.126
van	%	–	–	–	30.845	0	0	5.108	0	0
roll3000	%	2.763	3.804	3.131	2.763	3.804	0	2.763	3.375	0
nsrand_ipx	%	0.932	0.932	0.621	0.932	0.932	0	0.932	0.932	0
A1C1S1	%	7.297	5.438	4.464	5.569	5.438	2.361	5.429	3.423	0
A2C1S1	%	7.615	7.261	0.995	6.379	5.123	0	5.846	5.079	0
B1C1S1	%	11.672	13.689	4.495	11.672	7.749	0.863	11.672	7.132	0
B2C1S1	%	18.196	0.268	11.642	18.196	0	5.037	14.734	0	5.037
tr12-30	%	0.036	0.573	0.622	0.007	0.410	0.332	0	0.389	0.332
sp97ar	%	2.494	0.842	1.171	2.494	0.428	0	2.376	0.124	0
sp97ic	%	5.453	0.622	3.675	3.834	0.622	0.761	3.834	0.622	0
sp98ar	%	1.724	2.715	0.602	1.724	1.409	0	1.724	0.282	0
sp98ic	%	1.350	0.872	0.247	1.350	0.872	0	1.350	0.872	0

*Gaps for NSR8K refer to 1 hour, 5 hours, and 10 hours of CPU time, respectively.

Table 2: Heuristic performance comparison after 1, 3, and 5 hours.

we report the results obtained by LocBra when emphasizing feasibility (instead of optimality) on the six instances for which it was not able to find the best solution. Table 5 reports the gaps obtained with respect to the best solution values in Table 1. The improvements with respect to original LocBra runs are marked by $*$. In four out of the six cases, the final solution turned out to be as good as in the previous LocBra run; the solution improved in three cases, in two of which it was either equal or strictly better than the best known solution.

6 Conclusions and Extensions

We have proposed a new solution framework for general MIPs, based on a clever exploration of solution neighborhoods defined through (invalid) linear cuts called local branching constraints. The neighborhoods are searched by means of a black-box general purpose MIP solver, thus exploiting the level of sophistication reached nowadays by the MIP solvers. Though very simple, this idea proved quite effective

Name	cases of Figure 5 1.	2.	3.	4.	diversifications dv_{best}	dv	node_tl
mkc	4	-	36	29	7	8	300
swath	17	2	19	29	0	5	300
danoint	-	-	42	35	0	16	300
markshare1	-	1	7	3	1	2	2,400
markshare2	-	-	8	2	0	1	2,400
arki001	8	-	12	16	0	5	600
seymour	6	2	4	4	1	2	2400
net12	-	-	6	13	0	7	600
biella1	1	1	14	5	1	2	900
NSR8K	1	-	7	3	1	1	2,400
rail507	3	-	23	40	3	19	300
rail2536c	4	2	21	36	0	18	300
rail2586c	9	-	13	14	3	6	600
rail4284c	10	-	5	9	3	4	900
rail4872c	7	-	9	6	2	3	900
glass4	-	-	39	27	4	5	300
UMTS	3	-	13	17	4	5	600
van	-	-	3	4	0	1	2,400
roll3000	-	-	30	39	5	15	300
nsrand_ipx	7	-	32	30	4	8	300
A1C1S1	-	-	37	29	4	7	300
A2C1S1	-	-	40	26	2	5	300
B1C1S1	-	-	28	33	5	10	300
B2C1S1	-	-	28	35	6	12	300
tr12-30	-	-	26	10	0	5	600
sp97ar	-	-	9	11	0	3	600
sp97ic	-	-	38	29	7	8	300
sp98ar	2	-	32	32	3	7	300
sp98ic	3	1	33	28	4	7	300

Table 3: LocBra statistics.

elapsed Time	LocBra with local branching constraint $\sum_{j\in\overline{S}}(1-x_j) \le k' = 10$ seymour	rail507	rail2536c	rail2586c	rail4284c	rail4872c
1:00	* 423	* 174	691	964	1081	1588
3:00	* 423	* 174	690	* 954	* 1076	1561
5:00	* 423	* 174	* 690	* 954	* 1071	* 1552

Table 4: Improved solution values for set covering instances.

elapsed Time	LocBra emphasizing feasibility at the tactical level Abs. Gap markshare1	Abs. Gap markshare2	Abs. Gap net12	% Gap UMTS°	% Gap B2C1S1	% Gap tr12-30
1:00	4	* 5	–	* -0.048	8.317	2.597
3:00	3	* 5	–	* -0.066	8.317	2.129
5:00	2	* 3	* 0	* -0.069	7.299	2.018

°The negative gaps for instance UMTS indicate an improvement of the best known solution of Table 1. The new best value is 30,139,634.

Table 5: Alternative LocBra runs.

when solving hard MIPs.

We conclude the paper with an outline of possible modifications of the basic local branching idea that can extend its range of application. The validation of these ideas through computational testing is beyond the scope of the present paper, hence it is left to future research projects.

Tighter integration within the MIP solver

There are two ways of exploiting local branching constraints. The first uses the local branching constraints as a "strategic" branching rule within an exact solution method, to be alternated with a more standard "tactical" branching rule. This approach has been discussed in Sections 3 and 4, where we used the MIP solver as a black-box for performing the tactical branchings. This is remarkably simple to implement, but has the disadvantage of wasting part of the computational effort devoted, e.g., to the exploration the nodes where no improved solution could be found within the node time limit. Therefore, a more integrated (and flexible) framework where the two branching rules work in tight cooperation is expected to produce an enhanced performance.

Local search by branch-and-cut

A second way of using the local branching constraints is within a genuine heuristic framework akin to *Tabu Search* (TS) or *Variable Neighborhood Search* (VNS). As a matter of fact, all the main ingredients of these metaheuristics (defining the current solution neighborhood, dealing with tabu solutions or moves, imposing a proper diversification, etc.) can easily be modeled in terms of linear cuts to be dynamically inserted and removed from the model. This will lead naturally to a completely general (and hopefully powerful) TS or VNS framework for MIPs. Some very promising preliminary results in this direction are reported in [21].

Working with infeasible reference solutions

As stated, the local branching framework requires a starting (feasible) reference solution $\bar{x}^1$, that we assume is provided by the MIP black-box solver. However, for difficult MIPs (such as, e.g., hard set partitioning models) even the definition of this solution may require an excessive computing time. In this case, one should consider the possibility of working with *infeasible* reference solutions. It is then advisable to adopt the widely-used mechanism in metaheuristic frameworks consisting in modifying the MIP model by adding slack variables to (some of) the constraints, while penalizing them in the objective function.

Dealing with general-integer variables

Local branching is based on the assumption that the MIP model contains a relevant set of 0-1 variables to be used in the definition of the "distance function" $\Delta(x, \bar{x})$. According to our computational experience, this definition is effective even in case of MIPs involving general integer variables, in that the 0-1 variables (which are often associated with big-M terms) are likely to be largely responsible for the difficulty of the model. However, it may be the case that the MIP model at hand does not involve any 0-1 variable, or that the 0-1 variables do not play a relevant role in the model−hence the introduction of the local branching constraint does not help the MIP solver. In this situation, one is interested in modified local branching constraints including the general-integer variables in the definition of the distance function $\Delta(x, \bar{x})$. To this end, suppose MIP model (P) involves the bounds $l_j \leq x_j \leq u_j$ for the integer variables x_j $(j \in \mathcal{I} := \mathcal{B} \cup \mathcal{G})$. Then a suitable local branching constraint can be defined as follows:

$$\Delta_1(x,\bar{x}) := \sum_{j\in\mathcal{I}:\bar{x}_j=l_j} \mu_j(x_j - l_j) + \sum_{j\in\mathcal{I}:\bar{x}_j=u_j} \mu_j(u_j - x_j) + \sum_{j\in\mathcal{I}:l_j<\bar{x}_j<u_j} \mu_j(x_j^+ + x_j^-) \le k$$

where weights μ_j are defined, e.g., as $\mu_j = 1/(u_j - l_j)$ for all $j \in \mathcal{I}$, while the variation terms x_j^+ and x_j^- require the introduction into the MIP model of additional constraints of the form:

$$x_j = \bar{x}_j + x_j^+ - x_j^-, \quad x_j^+ \ge 0, \; x_j^- \ge 0, \quad \forall j \in \mathcal{I} : l_j < \bar{x}_j < u_j$$

Alternatively, one could avoid the use of the additional variables x_j^+ and x_j^- (and of the associated constraints) through one of the following relaxed definitions:

$$\Delta_2(x,\bar{x}) := \sum_{j\in\mathcal{I}:\bar{x}_j=l_j} \mu_j(x_j - l_j) + \sum_{j\in\mathcal{I}:\bar{x}_j=u_j} \mu_j(u_j - x_j) \le k'$$

or

$$\Delta_3(x,\bar{x}) := \sum_{j\in\mathcal{I}:\bar{x}_j=l_j} \mu_j(x_j - l_j) \le k''$$

Acknowledgements

Work partially supported by MIUR and CNR, Italy, and by the EU project "DONET". Thanks are due to anonymous referees for detailed and helpful comments.

References

[1] E. Balas, S. Ceria, M. Dawande, F. Margot and G. Pataki. OCTANE: A New Heuristic For Pure 0-1 Programs. Operations Research 49(2), 207–225, 2001.

[2] E. Balas and C.H. Martin. Pivot-And-Complement: A Heuristic For 0-1 Programming. Management Science 26(1), 86–96, 1980.

[3] P. Belotti. Personal communication, 2002.

[4] R.E. Bixby, S. Ceria, C.M. McZeal, M.W.P. Savelsbergh. MIPLIB 3.0. http://www.caam.rice.edu/∼bixby/miplib/miplib.html.

[5] A. Caprara, M. Fischetti and P. Toth. A Heuristic Method For The Set Covering Problem. Operations Research 47, 730–743, 1999.

[6] Cplex. *ILOG Cplex 7.0 User's Manual and Reference Manual.* ILOG, S.A., 2001 (http://www.ilog.com)

[7] F. Glover and M. Laguna. General Purpose Heuristics For Integer Programming: Part I. Journal of Heuristics 2, 343–358, 1997.

[8] F. Glover and M. Laguna. General Purpose Heuristics For Integer Programming: Part II. Journal of Heuristics 3, 161–179, 1997.

[9] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publisher, Boston, Dordrecht, London, 1997.

[10] J.W. Goessens, S. van Hoesel and L. Kroon. A Branch-and-Cut Approach to Line Planning Problems. Working Paper, Erasmus University, 2001.

[11] F. S. Hillier. Effcient Heuristic Procedures For Integer Linear Programming With An Interior. Operations Research 17(4), 600–637, 1969.

[12] T. Ibaraki, T. Ohashi and H. Mine. A Heuristic Algorithm For Mixed-Integer Programming Problems. Mathematical Programming Study 2, 115–136, 1974.

[13] L. Kroon. Personal communication, 2002.

[14] L. Kroon and M. Fischetti. Crew Scheduling for Netherlands Railways: Destination Customer. In S. Voss and J.R. Daduna (ed.s) *Computer-Aided Scheduling of Public Transport*, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, 181–201, 201.

[15] A. Løkketangen. Heuristics for 0-1 Mixed-Integer Programming. In P.M. Pardalos and M.G.C. Resende (ed.s) *Handbook of Applied Optimization*, Oxford University Press, 474–477, 2002.

[16] A. Løkketangen and F. Glover. Solving Zero/One Mixed Integer Programming Problems Using Tabu Search. European Journal of Operational Research 106, 624-658, 1998.

[17] I. Luzzi. Personal communication, 2001.

[18] C. Mannino and E. Parrello. Personal communication, 2002.

[19] N. Mladenovíc and P. Hansen. Variable Neighborhood Search. Computers and Operations Research 24, 1097–1100, 1997.

[20] M. Nediak and J. Eckstein. Pivot, Cut, and Dive: A Heuristic for 0-1 Mixed Integer Programming Research Report RRR 53-2001, RUTCOR, Rutgers University, October 2001.

[21] C. Polo. Algoritmi Euristici per il Progetto Ottimo di una Rete di Interconnessione. Tesi di laurea in Ingegneria Informatica, Universtità degli Studi di Padova, 2002 (in Italian).

[22] TURNI. *User's Manual*, Double-Click sas, 2001 (http://www.turni.it)

[23] M. Van Vyve and Y. Pochet. A General Heuristic for Production Planning Problems. CORE Discussion Paper 56, 2001.

7 Appendix: Detailed Results

We next report more detailed results on the performance of the three heuristics we compared in Section 5.

elapsed	`mkc` - %Gap			`swath` - %Gap			`danoint` - %Gap			`markshare1` - Abs. Gap			`markshare2` - Abs. Gap		
Time	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra
0:06	13.114	7.656	11.112	–	11.035	–	0	0	0	17	13	34	30	11	62
0:12	13.114	6.897	10.205	94.504	9.789	94.504	0	0	0	17	13	26	30	1	43
0:18	13.114	6.094	10.205	94.504	9.789	22.693	0	0	0	8	10	17	30	1	43
0:24	13.114	3.775	5.701	94.504	9.681	14.971	0	0	0	8	10	16	30	1	43
0:30	13.114	2.637	3.538	94.504	9.681	10.558	0	0	0	1	10	16	30	1	43
0:36	13.114	2.637	2.925	94.504	7.876	6.728	0	0	0	1	7	16	30	1	34
0:42	13.114	2.453	2.925	94.504	5.138	3.745	0	0	0	1	7	10	9	1	34
0:48	13.114	2.435	2.674	94.504	3.843	1.694	0	0	0	1	7	10	9	1	34
0:54	13.114	2.435	2.674	94.504	2.870	1.694	0	0	0	1	7	10	9	1	34
1:00	3.765	2.399	2.281	94.504	2.507	1.599	0	0	0	1	5	10	9	1	34
1:12	3.765	2.399	1.959	94.504	2.437	1.059	0	0	0	1	4	10	9	1	27
1:24	3.765	2.399	1.208	94.504	1.766	0	0	0	0	1	4	10	9	1	27
1:36	3.765	2.363	1.191	94.504	1.766	0	0	0	0	1	4	10	9	1	26
1:48	3.765	2.238	1.191	94.504	1.766	0	0	0	0	1	4	10	9	1	26
2:00	3.765	2.238	1.191	26.374	1.153	0	0	0	0	1	0	10	9	0	26
2:12	3.765	2.088	1.083	26.374	1.153	0	0	0	0	1	0	2	9	0	23
2:24	3.765	1.294	0.583	26.374	1.153	0	0	0	0	1	0	2	9	0	23
2:36	3.765	1.294	0.404	26.374	1.153	0	0	0	0	1	0	2	9	0	23
2:48	3.765	1.294	0.404	26.374	1.153	0	0	0	0	1	0	2	9	0	17
3:00	3.765	1.294	0.404	26.374	1.153	0	0	0	0	1	0	2	9	0	11
3:20	3.765	1.294	0.404	26.374	1.153	0	0	0	0	1	0	2	9	0	11
3:40	3.765	1.294	0.404	26.374	1.153	0	0	0	0	1	0	2	9	0	11
4:00	3.765	1.294	0.404	26.374	1.153	0	0	0	0	1	0	2	9	0	11
4:30	3.765	1.294	0.404	26.374	1.153	0	0	0	0	1	0	2	9	0	11
5:00	3.765	1.294	0	26.374	1.153	0	0	0	0	1	0	2	9	0	11

Table 6: Detailed results for some instances A in Table 1.

elapsed	arki001 - %Gap			seymour - Abs. Gap			net12 - Abs. Gap			biella1 - %Gap		
Time	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra
0:06	0.055	0.043	0.026	11	29	11	–	–	–	—	—	—
0:12	0.055	0.039	0.005	11	8	5	–	–	–	—	—	—
0:18	0.055	0.029	0.004	11	8	4	–	–	–	0.256	—	0.256
0:24	0.055	0.029	0.004	11	8	3	–	–	–	0.256	43.628	0.242
0:30	0.055	0.029	0.004	11	8	3	–	–	–	0.256	43.628	0.242
0:36	0.041	0.029	0.004	11	8	3	–	–	–	0.256	31.313	0.242
0:42	0.038	0.028	0.003	11	7	3	–	–	–	0.256	31.313	0.242
0:48	0.024	0.028	0.003	11	6	3	–	–	–	0.256	31.313	0.242
0:54	0.024	0.028	0.003	11	5	3	–	–	–	0.256	31.313	0.241
1:00	0.024	0.028	0	11	5	2	–	–	–	0.256	31.313	0.241
1:12	0.024	0.028	0	11	5	2	–	–	–	0.256	31.313	0.239
1:24	0.023	0.025	0	11	4	2	41	–	41	0.256	26.336	0.239
1:36	0.023	0.022	0	11	4	2	41	–	41	0.256	26.336	0.176
1:48	0.023	0.022	0	11	4	1	41	–	41	0.256	18.272	0.173
2:00	0.021	0.021	0	11	4	1	41	–	41	0.256	18.272	0.172
2:12	0.017	0.016	0	11	3	1	41	–	41	0.256	18.272	0.172
2:24	0.017	0.016	0	11	3	1	41	–	41	0.256	18.272	0.172
2:36	0.017	0.016	0	11	3	1	41	–	41	0.256	18.272	0.172
2:48	0.017	0.016	0	11	3	1	41	–	41	0.256	18.272	0.172
3:00	0.017	0.016	0	11	3	0	41	–	41	0.256	18.272	0.172
3:20	0.017	0.014	0	11	3	0	41	–	41	0.256	18.272	0.172
3:40	0.017	0.014	0	11	3	0	41	0	41	0.256	18.272	0.084
4:00	0.017	0.014	0	11	3	0	41	0	41	0.256	16.036	0.005
4:30	0.015	0.014	0	11	3	0	41	0	41	0.256	11.475	0
5:00	0.013	0.013	0	11	3	0	41	0	41	0.256	11.475	0

elapsed	NSR8K - %Gap		
Time	cpx-O	cpx-F	LocBra
1:00	—	—	—
2.00	—	—	—
2.30	—	—	—
3.00	—	—	—
3.30	—	—	—
3.45	1843.8	1843.8	1843.8
4:00	1843.8	1843.8	475.2
4:15	1843.8	1843.8	111.4
4:30	1843.8	1843.8	111.4
4:45	1843.8	1843.8	109.3
5:00	1843.8	1526.4	109.3
5:15	1843.8	1526.4	34.3
5:30	1843.8	1526.4	34.3
5:45	1843.8	1526.4	10.6
6:00	1843.8	1526.4	10.6
6:15	1843.8	1526.4	10.6
6:30	1843.8	1526.4	10.6
6:45	1843.8	1526.4	10.6
7:00	1843.8	1526.4	6.9
7:30	1843.8	1526.4	6.9
8:00	1426.8	1526.4	6.9
8:30	1426.8	1526.4	6.9
9:00	1426.8	1526.4	6.9
9:30	1426.8	1526.4	6.9
10:00	1426.8	1526.4	0

Table 7: Detailed results for instances A, B, C in Table 1.

elapsed	`rail507` - Abs. Gap			`rail2536c` - Abs. Gap			`rail2586c` - Abs. Gap			`rail4284c` - Abs. Gap			`rail4872c` - Abs. Gap		
Time	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra
0:06	15	9	4	8	8	1	–	–	–	–	–	–	–	–	–
0:12	9	9	2	8	3	0	63	34	26	–	–	–	–	–	–
0:18	7	9	2	8	3	0	63	34	11	–	–	–	–	–	–
0:24	2	9	2	8	3	0	63	34	11	–	–	–	–	–	–
0:30	2	9	1	5	3	0	54	34	10	51	45	33	–	–	–
0:36	2	9	1	5	3	0	54	34	10	51	45	27	–	–	–
0:42	2	9	1	5	3	0	54	34	9	51	45	17	73	69	37
0:48	2	7	1	5	3	0	54	34	7	51	45	14	73	69	31
0:54	2	6	1	5	3	0	54	34	7	51	40	14	73	69	31
1:00	2	6	1	5	3	0	54	34	7	51	40	10	73	69	27
1:12	2	6	0	4	3	0	30	34	6	51	40	7	70	49	23
1:24	2	6	0	3	3	0	30	34	5	51	40	5	70	49	23
1:36	2	6	0	3	3	0	30	34	5	51	40	5	70	49	20
1:48	2	6	0	3	3	0	30	34	5	51	40	5	70	49	20
2:00	2	6	0	3	3	0	30	34	5	51	40	3	70	49	11
2:12	2	6	0	2	3	0	20	34	5	51	40	3	70	49	10
2:24	2	6	0	2	3	0	17	34	5	51	40	2	70	49	10
2:36	2	5	0	2	3	0	17	34	5	42	40	2	70	49	9
2:48	2	5	0	1	3	0	17	34	4	42	40	2	70	49	9
3:00	2	5	0	1	3	0	14	34	2	42	40	2	70	49	9
3:20	2	5	0	1	3	0	14	34	2	42	40	2	60	49	5
3:40	2	5	0	1	3	0	14	34	0	42	40	2	45	49	2
4:00	2	5	0	1	3	0	14	34	0	42	40	2	45	49	2
4:30	2	5	0	0	3	0	14	34	0	42	40	0	45	49	0
5:00	1	5	0	0	3	0	14	34	0	38	40	0	45	49	0

Table 8: Detailed results for instances D in Table 1.

elapsed	`glass4` - %Gap			`UMTS` - %Gap			`van` - %Gap			`roll3000` - %Gap			`nsrand_ipx` - %Gap		
Time	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra
0:06	32.283	19.685	44.882	16.120	0.024	3.293	—	—	—	34.604	18.523	18.638	0.932	1.553	9.938
0:12	32.283	19.685	41.522	6.403	0.024	3.293	—	—	—	34.604	11.420	14.068	0.932	1.553	6.832
0:18	31.444	16.535	30.709	6.403	0.024	3.293	—	—	—	34.604	7.585	14.068	0.932	1.553	4.969
0:24	29.134	15.748	30.709	6.403	0.024	2.936	—	—	—	34.604	7.287	8.006	0.932	1.553	4.348
0:30	29.134	13.386	29.134	6.403	0.024	2.936	—	—	—	34.604	7.287	8.006	0.932	1.553	3.727
0:36	29.134	13.386	19.685	6.403	0.024	2.936	—	—	—	34.604	5.779	8.006	0.932	1.553	2.795
0:42	29.134	7.087	19.685	6.403	0.024	2.920	—	—	—	2.763	3.804	8.006	0.932	0.932	2.174
0:48	29.134	7.087	19.685	6.403	0.014	2.413	—	—	—	2.763	3.804	3.131	0.932	0.932	2.174
0:54	29.134	7.087	19.685	6.403	0.014	2.413	—	—	—	2.763	3.804	3.131	0.932	0.932	1.863
1:00	22.835	7.087	19.685	6.403	0	2.413	—	—	—	2.763	3.804	3.131	0.932	0.932	0.621
1:12	21.785	7.087	19.685	6.403	0	2.413	—	—	—	2.763	3.804	3.131	0.932	0.932	0.311
1:24	21.785	7.087	19.685	6.403	0	2.285	30.845	30.845	30.845	2.763	3.804	3.131	0.932	0.932	0.311
1:36	17.323	7.087	19.685	6.403	0	1.216	30.845	30.845	30.845	2.763	3.804	3.131	0.932	0.932	0.311
1:48	16.535	7.087	19.685	6.403	0	1.216	30.845	30.845	30.845	2.763	3.804	0	0.932	0.932	0.311
2:00	16.535	7.087	17.747	6.403	0	1.216	30.845	0	30.845	2.763	3.804	0	0.932	0.932	0.311
2:12	16.535	7.087	13.386	6.403	0	1.216	30.845	0	0	2.763	3.804	0	0.932	0.932	0
2:24	16.535	7.087	10.236	6.403	0	1.216	30.845	0	0	2.763	3.804	0	0.932	0.932	0
2:36	16.535	7.087	10.236	6.403	0	1.216	30.845	0	0	2.763	3.804	0	0.932	0.932	0
2:48	16.535	7.087	10.236	6.403	0	1.216	30.845	0	0	2.763	3.804	0	0.932	0.932	0
3:00	14.226	7.087	10.236	6.403	0	1.216	30.845	0	0	2.763	3.804	0	0.932	0.932	0
3:20	14.226	7.087	10.236	6.403	0	1.204	30.845	0	0	2.763	3.804	0	0.932	0.932	0
3:40	14.226	7.087	10.236	6.403	0	1.159	5.108	0	0	2.763	3.804	0	0.932	0.932	0
4:00	13.386	7.087	7.087	6.403	0	0.256	5.108	0	0	2.763	3.375	0	0.932	0.932	0
4:30	13.386	7.087	0	6.403	0	0.126	5.108	0	0	2.763	3.375	0	0.932	0.932	0
5:00	13.386	7.087	0	6.403	0	0.126	5.108	0	0	2.763	3.375	0	0.932	0.932	0

Table 9: Detailed results for instances E, F, G in Table 1.

elapsed	A1C1S1 - %Gap			A2C1S1 - %Gap			B1C1S1 - %Gap			B2C1S1 - %Gap			tr12-30 - %Gap		
Time	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra
0:06	9.753	9.055	18.149	10.021	13.802	12.000	—	—	—	18.196	8.317	19.116	0.227	0.789	1.171
0:12	9.126	5.438	7.943	8.303	11.670	8.082	18.513	13.689	13.903	18.196	7.173	14.491	0.036	0.736	1.171
0:18	9.126	5.438	5.570	8.303	9.065	5.596	18.513	13.689	13.903	18.196	7.173	11.642	0.036	0.659	1.113
0:24	7.297	5.438	5.570	8.303	8.772	4.623	18.513	13.689	13.903	18.196	7.173	11.642	0.036	0.659	1.085
0:30	7.297	5.438	5.514	7.615	8.318	3.716	18.513	13.689	13.903	18.196	7.173	11.642	0.036	0.659	1.085
0:36	7.297	5.438	5.277	7.615	8.287	3.716	18.513	13.689	13.903	18.196	7.173	11.642	0.036	0.659	0.782
0:42	7.297	5.438	4.536	7.615	8.124	3.289	18.513	13.689	13.903	18.196	0.505	11.642	0.036	0.651	0.739
0:48	7.297	5.438	4.536	7.615	7.604	2.166	18.513	13.689	13.903	18.196	0.505	11.641	0.036	0.608	0.712
0:54	7.297	5.438	4.464	7.615	7.604	2.166	11.672	13.689	7.115	18.196	0.268	11.642	0.036	0.573	0.712
1:00	7.297	5.438	4.464	7.615	7.261	0.995	11.672	13.689	4.495	18.196	0.268	11.642	0.036	0.573	0.622
1:12	6.508	5.438	4.464	7.615	7.144	0.964	11.672	13.689	4.290	18.196	0.268	11.642	0.036	0.573	0.470
1:24	6.508	5.438	4.464	6.379	7.144	0.264	11.672	11.442	0.863	18.196	0.268	11.642	0.036	0.573	0.332
1:36	5.569	5.438	4.464	6.379	7.144	0.264	11.672	11.221	0.863	18.196	0.268	11.642	0.036	0.573	0.332
1:48	5.569	5.438	3.992	6.379	7.144	0.264	11.672	10.589	0.863	18.196	0.268	11.642	0.036	0.436	0.332
2:00	5.569	5.438	3.359	6.379	7.144	0.264	11.672	8.457	0.863	18.196	0.268	11.642	0.036	0.410	0.332
2:12	5.569	5.438	3.359	6.379	7.144	0.264	11.672	8.457	0.863	18.196	0.268	11.642	0.036	0.410	0.332
2:24	5.569	5.438	3.359	6.379	5.556	0.264	11.672	8.457	0.863	18.196	0	5.037	0.013	0.410	0.332
2:36	5.569	5.438	3.359	6.379	5.123	0	11.672	8.457	0.863	18.196	0	5.037	0.007	0.410	0.332
2:48	5.569	5.438	3.177	6.379	5.123	0	11.672	8.457	0.863	18.196	0	5.037	0.007	0.410	0.332
3:00	5.569	5.438	2.361	6.379	5.123	0	11.672	7.749	0.863	18.196	0	5.037	0.007	0.410	0.332
3:20	5.429	5.438	0.780	6.379	5.079	0	11.672	7.749	0	18.196	0	5.037	0.007	0.410	0.332
3:40	5.429	5.438	0.695	6.379	5.079	0	11.672	7.749	0	18.196	0	5.037	0.007	0.410	0.332
4:00	5.429	4.978	0.160	6.379	5.079	0	11.672	7.749	0	18.196	0	5.037	0.007	0.410	0.332
4:30	5.429	4.793	0	6.379	5.079	0	11.672	7.749	0	14.734	0	5.037	0	0.389	0.332
5:00	5.429	3.423	0	5.846	5.079	0	11.672	7.132	0	14.734	0	5.037	0	0.389	0.332

Table 10: Detailed results for instances H in Table 1.

elapsed	sp97ar - %Gap			sp97ic - %Gap			sp98ar - %Gap			sp98ic - %Gap		
Time	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra	cpx-O	cpx-F	LocBra
0:06	9.796	7.601	4.690	8.730	6.029	8.498	6.121	4.338	2.894	7.285	6.002	8.580
0:12	4.404	4.126	4.278	8.574	3.698	6.485	5.808	4.069	2.894	5.436	4.090	6.869
0:18	3.952	2.631	4.278	8.574	1.487	3.675	5.017	4.069	2.578	5.209	3.360	5.129
0:24	3.952	2.570	3.588	8.574	0.846	3.675	5.017	4.069	2.366	5.209	3.360	0.247
0:30	2.494	2.570	3.534	8.574	0.846	3.675	4.675	3.470	2.163	5.209	0.872	0.247
0:36	2.494	2.570	2.514	6.564	0.846	3.675	1.724	3.470	1.028	5.209	0.872	0.247
0:42	2.494	1.157	1.171	6.564	0.622	3.675	1.724	2.771	1.028	5.209	0.872	0.247
0:48	2.494	1.157	1.171	6.564	0.622	3.675	1.724	2.771	0.841	5.209	0.872	0.247
0:54	2.494	0.842	1.171	6.564	0.622	3.675	1.724	2.771	0.841	1.350	0.872	0.247
1:00	2.494	0.842	1.171	5.453	0.622	3.675	1.724	2.715	0.602	1.350	0.872	0.247
1:12	2.494	0.842	1.171	5.453	0.622	2.029	1.724	2.134	0.158	1.350	0.872	0.247
1:24	2.494	0.842	1.171	5.055	0.622	2.029	1.724	1.409	0.158	1.350	0.872	0.247
1:36	2.494	0.842	0.649	5.055	0.622	2.029	1.724	1.409	0.158	1.350	0.872	0.247
1:48	2.494	0.842	0.649	5.055	0.622	2.029	1.724	1.409	0.158	1.350	0.872	0.247
2:00	2.494	0.666	0.649	5.055	0.622	2.029	1.724	1.409	0.049	1.350	0.872	0.247
2:12	2.494	0.666	0.649	5.055	0.622	2.029	1.724	1.409	0.012	1.350	0.872	0.247
2:24	2.494	0.666	0.649	5.055	0.622	2.029	1.724	1.409	0.012	1.350	0.872	0.247
2:36	2.494	0.666	0.441	3.834	0.622	0.761	1.724	1.409	0	1.350	0.872	0.247
2:48	2.494	0.666	0	3.834	0.622	0.761	1.724	1.409	0	1.350	0.872	0.114
3:00	2.494	0.428	0	3.834	0.622	0.761	1.724	1.409	0	1.350	0.872	0
3:20	2.494	0.124	0	3.834	0.622	0.761	1.724	0.282	0	1.350	0.872	0
3:40	2.376	0.124	0	3.834	0.622	0.761	1.724	0.282	0	1.350	0.872	0
4:00	2.376	0.124	0	3.834	0.622	0.761	1.724	0.282	0	1.350	0.872	0
4:30	2.376	0.124	0	3.834	0.622	0.761	1.724	0.282	0	1.350	0.872	0
5:00	2.376	0.124	0	3.834	0.622	0	1.724	0.282	0	1.350	0.872	0

Table 11: Detailed results for instances I in Table 1.