# Proximity Search for 0-1 Mixed-Integer Convex Programming

Matteo Fischetti and Michele Monaci
DEI, University of Padova, via Gradenigo 6/A, 35100 Padova (Italy)
e-mail: {*matteo.fischetti, michele.monaci*}*@unipd.it*

31 July 2013

**Abstract**

In this paper we investigate the effects of replacing the objective function of a 0-1 Mixed-Integer Convex Program (MIP) with a "proximity" one, with the aim of enhancing the heuristic behavior of a black-box solver. The relationship of this approach with primal integer methods is also addressed. Promising computational results on different proof-of-concept implementations are presented, suggesting that proximity search can be very effective in quickly improving the incumbent in the early part of the search. This is particularly true when a sequence of similar MIPs has to be solved as, e.g., in a column-generation setting.

**Keywords;** Mixed-Integer Convex Optimization, Proximal methods, Primal methods, Heuristics.

## 1 Introduction

In this paper we focus on a generic 0-1 Mixed Integer (possibly nonlinear) Program (MIP, for short) of the form

$$\min f(x) \tag{1}$$
$$g(x) \leq 0 \tag{2}$$
$$x_j \in \{0,1\} \quad \forall j \in J \tag{3}$$

where $f : \Re^n \to \Re$, $g : \Re^n \to \Re^m$, and $J \subseteq N := \{1, \cdots, n\}$, $J \neq \emptyset$, indexes binary variables. Although this is not strictly required by our method, in the following we assume that both $f$ and $g$ are convex functions, so dropping the integrality condition in (3) leads to a polynomially solvable relaxation.

The exact solution of the above problem is generally attempted by using an enumerative scheme (branch-and-bound or branch-and-cut) that computes lower bounds on the optimal solution value at each branching node. As the aim of the search is to converge to proven optimality, a best-bound first strategy is typically used to visit the search tree, and the nodes with best lower bound have the highest priority to be elaborated. A well known drawback of this approach is however that the search tends to be initially trapped in the upper part of the tree (where lower bounds are better), which is counterproductive in terms of probability of updating the incumbent. Hence a mixed strategy is generally preferred, that (i) selects the next node to elaborate according to the best-bound rule (or some variant), (ii) makes a sequence of "diving" branching steps by visiting nodes at increasing distance—in terms of lower bound—from the root, until an infeasibility condition is reached (or the incumbent is updated). A consequence of this approach is that the search tree grows in a region "close to the root" (in terms of lower bounds) with frequent heuristic divings to grasp "far away" feasible solutions.

The above strategy proved quite effective in improving the best lower bound among the open tree nodes—and then in eventually proving optimality of the incumbent. However, its effectiveness in providing early feasible solutions of good quality is less clear. As a matter of fact, for very hard instances some MIP solvers prefer to initially reset $f(x)$ to zero—the objective function being viewed as a disturbing element that limits the search scope and interferes with the internal heuristics.

In this paper we investigate a more elaborated strategy. We start with a feasible solution $\tilde{x}$, and add an explicit *cutoff constraint*

$$f(x) \leq f(\tilde{x}) - \theta \tag{4}$$

to the MIP, where $\theta > 0$ is a given cutoff tolerance. At this point we are free to modify the objective function to heuristically drive the search and to hopefully discover better feasible solutions in the early part of the search. A natural option is to use a proximity function that penalizes a solution $x$ according to its distance from $\tilde{x}$, e.g., by taking the Hamming distance

$$\Delta(x, \tilde{x}) := \sum_{j \in J : \tilde{x}_j = 0} x_j + \sum_{j \in J : \tilde{x}_j = 1} (1 - x_j) \tag{5}$$

The idea is to mimic a primal method that moves from a feasible solution to a nearby better feasible one, by using a black-box MIP solver to actually perform the move. In this respect, our method introduces an important new ingredient in the design of MIP heuristics based on the solution of auxiliary "sub-MIPs": instead of modifying the constraints of the MIP at hand with the aim of reducing the search space, we modify the objective function with the aim of easing the search. Mixed strategies are also possible, but not investigated in the present paper as we prefer to keep our testing environment as simple (and clean) as possible.

The main goal of the present paper is a computational investigation of pros and cons of proximity-driven search in 0-1 MIPs, with the aim of exploring diversified ideas and of attracting more research on this topic.

The outcome of our experiments is that exploiting a proximity function is a very promising way of quickly producing improved solutions for important classes of 0-1 MIPs. This is particularly true when a sequence of similar MIPs has to be solved as, e.g., in a column-generation setting. An unexpected side effect is that the solution time per node is sometimes much *smaller* when $\Delta(x, \cdot)$ is used instead of $f(x)$, meaning that a much larger number of nodes are explored in a given amount of time. This suggests that proximity functions can play an important role in the solution of problems where the original convex relaxation turns out to be very difficult to (re)optimize, as e.g. in very large MILPs where an initial heuristic solution can be quickly computed by ad-hoc methods.

The paper is organized as follows. Our approach is outlined in Section 2, while pointers to related literature are given in Section 3. Three different implementations of the basic approach are proposed in Section 4. Section 5 describes the setup and the outcome of our experiments on three important classes of 0-1 MIPs, namely, set covering, network design, and classification problems. In Section 6, we argue that proximity search can fit very well within column generation schemes, and we report computational experiments supporting our claim. Some conclusions and ideas for future research are finally given in Section 7.

# 2 The basic idea

Our basic approach is sketched in Figure 1 below.

At Step 1, the initial feasible solution $\tilde{x}$ is defined. In practical applica-

`Proximity Search`:

1. let $\tilde{x}$ be the initial heuristic feasible solution to refine;
   **repeat**
2. explicitly add the *cutoff constraint* $f(x) \leq f(\tilde{x}) - \theta$ to the MIP model;
3. replace $f(x)$ by the "proximity" objective function $\Delta(x, \tilde{x})$;
4. run the MIP solver on the new model until a termination condition is reached, and let $x^*$ be the best feasible solution found ($x^*$ empty if none);
   **if** $x^*$ is nonempty and $J \subset N$ **then**
5. refine $x^*$ by solving the convex program
   $$x^* := \text{argmin}\{f(x) : g(x) \leq 0,\ x_j = x^*_j\ \forall j \in J\}$$
   **end**
6. recenter $\Delta(x, \cdot)$ by setting $\tilde{x} := x^*$, and/or update $\theta$
   **until** *an overall termination condition is reached*;

Figure 1: The basic Proximity Search algorithm

tions, this initial solution can be found by a fast ad-hoc heuristic, and our approach can be used to refine it by exploiting an underlying MIP model whose solution from scratch turned out to be problematic. Otherwise, $\tilde{x}$ can be found by running the black-box MIP solver until a first feasible solution is found, or by setting a conservative time/node limit. In all cases, we assume that finding a feasible solution is not really an issue for the problem at hand. If this is not the case, one should resort to a problem reformulation where some constraints are imposed in a soft way through violation penalties attached to slack variables.

At Step 2, the cutoff tolerance $\theta$ is defined. In our experiments we decided to be very conservative and to set $\theta$ to a small value, or just to 1 in case the objective function is integer. In this way, we expect to have a (possible long) series of not-too-difficult sub-MIPs to solve, each leading to a small improvement of the incumbent. Needless to say, more aggressive policies can lead to a significantly better performance, but would require an ad-hoc tuning of $\theta$ that would make the outcome of our experiments less clear.

At Step 3, we use the Hamming distance between $x$ and $\tilde{x}$, computed according to (5). In some cases, one could use $\Delta(x, \tilde{x}) + \rho\, f(x)$ as the objective function to be optimized, where $\rho > 0$ is a weighing factor intended to favor

low-cost solutions. To keep our setting as clean as possible, however, we preferred to avoid any (over)tuning of the additional parameter $\rho$, and just used $\rho = 0$.

Step 4 invokes the black-box MIP solver to hopefully find a new incumbent $x^*$ with $f(x^*) \leq f(\tilde{x}) - \theta$. No input cutoff on the optimal value of the $\Delta$ objective function is heuristically provided to the MIP solver, although in some cases it would speedup the search considerably. A crucial property here is that the root-node solution of the convex relaxation, say $x'$, is expected to be not too different from $\tilde{x}$, as this latter solution would be optimal without the cutoff constraint, that for a small $\theta$ can typically be fulfilled with just local adjustments. This is instrumental for the success of our method, in that it has two main positive effects: (i) the computing time spent at the root node is often very small, and (ii) $x'$ is typically "almost integer" (i.e., with a small number of fractional components indexed by $J$), hence it is more effective in driving the internal heuristics of the black-box MIP solver, as well as in guiding the search path towards integer solutions.

The above beneficial effects will be referred to as *improved relaxation grip*, and are illustrated in Table 1 for the set covering (pure binary) MIPLIB2010 instance `ramos3` with $n = 2,187$ variables, when a reference solution $\tilde{x}$ of value 267 is chosen. The table reports the number of components of the LP relaxation solution $x'$ that belong to the intervals [0,0], (0, 0.1], ..., (0.9, 1], and [1,1], along with computing time (in CPU sec.s), number of simplex iterations (dual pivots), and objective value—i.e., distance $\Delta(x', \tilde{x})$. The LP relaxation becomes infeasible for $\theta > 121$. For small values of $\theta$, the effect on the fractionality of $x'$ and on the LP-solution time is striking—and the relaxation grip is dramatically improved.

If no new solution $x^*$ is found at Step 4 (possibly because the MIP solver was aborted before convergence), we proceed directly to Step 6 where tolerance $\theta$ is reduced. Of course, if the MIP solver proved infeasibility for the given $\theta$, we have that $f(\tilde{x}) - \theta$ is a valid lower bound on the optimal value of the original MIP.

At Step 5, the new solution $x^*$, if any, is possibly improved by solving a convex problem where all binary variables have been fixed to their value in $x^*$ so as to find the best solution within the neighborhood induced by $\Delta(x, x^*) = 0$.

At Step 6, the approach is reapplied on a different $\tilde{x}$ (if available) so as to recenter the distance function $\Delta$, and/or by modifying the cutoff tolerance $\theta$.

| $x$-range | $\theta=0$ | $\theta=1$ | $\theta=2$ | $\theta=3$ | $\theta=4$ | $\theta=5$ | $\theta=10$ | $\theta=20$ | $\theta=30$ | $\theta=50$ | $\theta=99$ | $\theta=121$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $=0$ | 1920 | 1919 | 1919 | 1919 | 1924 | 1920 | 1619 | 1619 | 1600 | 1565 | 1276 | 682 |
| $(0.0, 0.1]$ | 0 | 0 | 0 | 0 | 0 | 0 | 303 | 297 | 293 | 281 | 420 | 926 |
| $(0.1, 0.2]$ | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 6 | 26 | 65 | 194 | 380 |
| $(0.2, 0.3]$ | 0 | 1 | 0 | 5 | 0 | 0 | 0 | 3 | 7 | 15 | 64 | 169 |
| $(0.3, 0.4]$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 8 | 75 | 29 |
| $(0.4, 0.5]$ | 0 | 0 | 6 | 0 | 0 | 0 | 8 | 4 | 3 | 16 | 91 | 0 |
| $(0.5, 0.6]$ | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 5 | 9 | 19 | 47 | 1 |
| $(0.6, 0.7]$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 9 | 35 | 17 | 0 |
| $(0.7, 0.8]$ | 0 | 5 | 0 | 1 | 0 | 1 | 0 | 10 | 25 | 88 | 3 | 0 |
| $(0.8, 0.9]$ | 0 | 0 | 0 | 0 | 0 | 11 | 0 | 28 | 101 | 68 | 0 | 0 |
| $(0.9, 1.0)$ | 0 | 0 | 0 | 0 | 0 | 0 | 249 | 209 | 110 | 26 | 0 | 0 |
| $=1$ | 267 | 262 | 262 | 262 | 263 | 251 | 3 | 3 | 2 | 1 | 0 | 0 |
| time (sec.s) | 0.00 | 0.04 | 0.03 | 0.03 | 0.04 | 0.21 | 0.45 | 0.54 | 0.57 | 0.90 | 4.77 | 30.91 |
| # LP-iter.s | 0 | 352 | 341 | 357 | 358 | 1180 | 2164 | 2543 | 2637 | 3627 | 6829 | 11508 |
| $\Delta$-distance | 0.00 | 1.50 | 3.00 | 4.50 | 6.00 | 7.88 | 17.45 | 37.13 | 56.86 | 96.90 | 208.71 | 292.67 |

Table 1: Relaxation grip induced by proximity search for various values of the cutoff parameter $\theta$.

Although we do not address general integer variables in the present paper, we observe that the definition of $\Delta$ can easily be modified to take such variables into account, though this requires the introduction of additional variables to model terms $|x_j - \tilde{x}_j|$ for all $x_j$'s integer but nonbinary; see [2] for details. Alternatively, one can treat general integer variables as if they were continuous, and apply a MIP heuristic to redefine all non-binary variables at Step 5.

# 3 Related approaches from the literature

The idea of dealing with a proximity term in the objective function is of course not new, and is in fact the basis of *augmented Lagrangian* and of *proximal methods* for (possibly nonconvex) optimization problems; see e.g. the book of [14] for an introduction, and the work of [7] for a theoretical analysis of proximal methods in discrete optimization.

As to MIP heuristics, we next outline some main approaches to refine a given solution $\tilde{x}$ that are related to our approach; the reader is referred to, e.g., [11] for a recent survey on MIP heuristics.

A method from the literature akin to proximity search is the *local branching* paradigm proposed in [10]. Here the objective function is not modified, and the proximity requirement is modeled through an explicit local-branching

constraint of the form

$$\Delta(x, \tilde{x}) \leq k \tag{6}$$

for a certain parameter $k$ that determines the radius of neighborhood of $\tilde{x}$ to be explored. Our approach can therefore be interpreted as a dual version of local branching, with functions $f(x)$ and $\Delta(x, \tilde{x})$ swapping their role.

As in proximity search, for small values of $k$ the local branching constraint is likely to affect the optimal solution $x'$ of the root-node relaxation, bringing it very close to the reference solution $\tilde{x}$. As already noticed, this fact significantly improves the relaxation grip, which is in fact a main reason for the success of the method. On the other hand, this effect vanishes for larger values of $k$, so tuning becomes problematic in those cases where setting a small $k$ prevents finding an improving solution, while setting a larger $k$ vanishes the local-branching grip effect completely. In this respect, proximity search with a small $\theta$ has the advantage of having a very good relaxation grip while not excluding any improving solution through a hard constraint—as local branching does. As there is no free lunch, also proximity search has a drawback compared to local branching: if a large cost improvement is possible within a very small neighborhood radius, local branching can attain it all in a single sub-MIP reoptimization, while proximity search will typically require a sequence of calls. This suggests a hybrid approach (not investigated in the present paper) where local branching with a small $k$ (say, $k = 10$) is repeatedly invoked in the beginning of the search (when large improvements within a small radius are possible), and one switches to proximity search when local branching does not provide any improvement because of its very tight $k$.

Table 2 (akin to Table 1) illustrates the impact of the local branching constraint for various values of the right-hand-side value $k$, again for the set covering instance `ramos3` when a reference solution $\tilde{x}$ of value 267 is chosen. By design, the relaxation grip improves dramatically when a small value for $k$ is chosen, and vanishes for larger values.

Another "dual" version of local branching is the *Feasibility Pump* (FP) heuristic introduced in [9] for 0-1 Mixed-Integer Linear Programs (0-1 MILPs), where the LP relaxation of the original problem is iteratively solved with respect to the objective function $\Delta(x, \lfloor x^* \rceil)$, $x^*$ is the LP optimal solution at the previous iteration, and $\lfloor \cdot \rceil$ denotes rounding to the nearest integer. An interesting interpretation of the feasibility pump in terms of proximal methods was proposed by [4]. Various extensions and adaptations of the

| $x$-range | $k=0$ | $k=1$ | $k=2$ | $k=3$ | $k=4$ | $k=5$ | $k=10$ | $k=20$ | $k=30$ | $k=50$ | $k=99$ | $k=+\infty$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $=0$ | 1920 | 1919 | 1919 | 1919 | 1919 | 1919 | 1920 | 1619 | 1619 | 1606 | 1562 | 672 |
| $(0.0, 0.1]$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 303 | 301 | 302 | 276 | 849 |
| $(0.1, 0.2]$ | 0 | 1 | 0 | 0 | 0 | 5 | 0 | 0 | 2 | 14 | 73 | 551 |
| $(0.2, 0.3]$ | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 4 | 16 | 108 |
| $(0.3, 0.4]$ | 0 | 0 | 1 | 0 | 5 | 0 | 0 | 0 | 3 | 2 | 7 | 7 |
| $(0.4, 0.5]$ | 0 | 0 | 0 | 6 | 0 | 0 | 0 | 8 | 5 | 2 | 17 | 0 |
| $(0.5, 0.6]$ | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 5 | 5 | 9 | 18 | 0 |
| $(0.6, 0.7]$ | 0 | 0 | 5 | 0 | 1 | 0 | 0 | 0 | 0 | 6 | 40 | 0 |
| $(0.7, 0.8]$ | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 0 | 3 | 17 | 86 | 0 |
| $(0.8, 0.9]$ | 0 | 5 | 0 | 0 | 0 | 1 | 0 | 0 | 14 | 81 | 67 | 0 |
| $(0.9, 1.0)$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 249 | 232 | 142 | 24 | 0 |
| $=1$ | 267 | 262 | 262 | 262 | 262 | 262 | 252 | 3 | 3 | 2 | 1 | 0 |
| time (sec.s) | 0.01 | 0.08 | 0.12 | 0.14 | 0.16 | 0.13 | 0.31 | 0.55 | 0.61 | 0.73 | 1.40 | 98.18 |
| # LP-iter.s | 0 | 827 | 1033 | 1145 | 1214 | 1095 | 1930 | 2897 | 3101 | 3476 | 4971 | 23870 |
| LP-bound | 267.00 | 266.33 | 265.66 | 265.00 | 264.33 | 263.66 | 260.88 | 255.70 | 250.62 | 240.47 | 215.97 | 145.80 |

Table 2: Relaxation grip induced by local branching for various values of the right-hand-side parameter $k$.

FP have been introduced in the recent literature. Among them, [5] modifies FP to deal with Mixed-Integer Nonlinear (possibly nonconvex) Problems (MINLPs). The method iteratively updates a pair of points $(x^*, \tilde{x})$ with the aim of reducing their Euclidean distance until they eventually coincide. Point $x^*$ belongs to the nonlinear relaxation of the given MINLP obtained by dropping the integrality requirement, and is computed by solving a nonlinear continuous problem where distance $\|x - \tilde{x}\|^2$ is minimized. As to $\tilde{x}$, it satisfies the integrality requirement and belongs to an LP relaxation of the MINLP (to be iteratively tightened); it is computed by means of a black-box MILP enumerative solver with objective function $\Delta(x, x^*)$. This latter step resembles what is done in proximity search, with two important differences however: (1) to improve relaxation grip, proximity search considers the distance $\Delta(x, \cdot)$ with respect to an integer solution, and not with respect to the possibly highly-fractional solution $x^*$; and (2) the proximity search recipe is to apply enumeration to the original problem and not to a MILP relaxation of it. In particular, even in case the original problem would be just 0-1 MILP the two methods will behave in a completely different way—in the very first iteration, the method in [5] will find the optimal LP solution $x^*$ and then solve a 0-1 MILP with objective function $\Delta(x, x^*)$ whose root-node LP solution is again $x^*$, thus losing any relaxation grip improvement.

Other approaches to refine a given solution $\tilde{x}$ by solving sub-MIPs have been proposed in the literature, that are not related to our approach but are

briefly outlined below as they will be used as a benchmark. These methods are based on the idea of fixing some of the integer variables in $\tilde{x}$. Among them, one of the most effective is the *Relaxation Induced Neighborhood Search* (RINS) introduced in [8]. At certain nodes of the branch-and-bound tree, the current LP relaxation solution $x'$ and the incumbent $\tilde{x}$ are compared, all integer-constrained variables that agree in value are fixed, and the resulting sub-MIP is solved through the black-box MIP solver.

Another example of a general MIP refinement procedure is the *polishing method* proposed in [17]. The method implements an evolutionary metaheuristic to be applied at selected nodes of a branch-and-bound tree. A fixed-size population of feasible solutions is maintained. Iteratively, two or more "parent" solutions are combined with the aim of creating a new "son" member of the population. This is done by fixing all variables whose value coincides in the parents solutions, and by heuristically solving the resulting sub-MIP by invoking an external MIP solver for a limited number of branch-and-bound nodes. Diversification is guaranteed by performing a classical mutation operation that consists in (i) selecting at random a seed solution in the population, (ii) fixing at random some of its variables, and (iii) heuristically solving the resulting sub-MIP.

# 4 Proximity search implementations

In this section we present three possible variants of the basic algorithm outlined in Section 2.

## 4.1 Proximity search without recentering

In this version, we assume the MIP solver can be controlled through a callback function invoked each time the incumbent is going to be updated—as it happens in many modern solvers. Within the callback function, the new incumbent $\hat{x}$ (say) is internally recorded but we prevent the solver to update the incumbent as we declare $\hat{x}$ to be infeasible and immediately add a new cut $f(x) \leq f(\hat{x}) - \theta$ as a global constraint.

This approach is also applied to the initial solution $\tilde{x}$, which is immediately made infeasible through the cutoff constraint $f(x) \leq f(\tilde{x}) - \theta$. In this way the optimal relaxation solution $x'$ at the root node is different from $\tilde{x}$, and violated MIP cuts can possibly be generated at the root node.

Notice that the proximity objective function $\Delta(x, \cdot)$ is never changed during the search, as it remains "centered" with the initial $\tilde{x}$—hence the name of "proximity search without recentering" we use for this scheme.

It should be observed that the simple implementation above has some obvious drawbacks that can affect its performance in a negative way. In particular, as we never update the MIP incumbent explicitly, we never apply any propagation and variable-fixing scheme, nor any refinement heuristic.

## 4.2 Proximity search with recentering

As already noticed, the implementation in the previous subsection has a number of drawbacks related to the need of interacting with the underlying MIP solver through a callback function. In fact, this kind of control might be not available for the MIP solver at hand—or its use can deactivate some important features of the solver itself. In addition, after a significant number of updates of $\tilde{x}$ it would make sense to "recenter" the proximity objective function $\Delta(x, \cdot)$ with respect the new $\tilde{x}$, an operation that cannot be done without restarting the MIP solver—at least, by using MIP solvers where the objective function cannot be changed on the fly.

We therefore analyze a different implementation that uses the MIP solver as a black box (with no callbacks), and just restarts it as soon as a new $\tilde{x}$ if found.

In our new implementation, that we call "proximity search with recentering", Steps 1 to 3 are the same as in Figure 1. At Step 4, after having added the cutoff constraint and changed the objective function, we invoke the MIP solver as a black box, in its default mode and without any callback, and abort its execution as soon as a first feasible solution is found. Because of the cutoff constraint, this solution (if any) is a strict improvement of $\tilde{x}$, so at Step 6 we replace $\tilde{x}$ with the new solution and repeat (without changing $\theta$) from Step 2, until the overall time limit is reached. Of course, if no improving solution is found at Step 4, the algorithm either proves $\theta$-optimality of the incumbent $\tilde{x}$ or hits the time limit.

A drawback of the new implementation is that each restart wastes the tree search performed in the last MIP call, as only the improved $\tilde{x}$ is used in the new MIP call. So it would make sense to address a mixed policy where recentering and restart is executed only when a significantly different $\tilde{x}$ is reached. Testing this approach would however require the definition of a new parameter to be tuned, with the risk of contaminating the results, so we

do not consider this variant in the present paper.

### 4.3 Proximity search with an incumbent

Both implementations above prevent the MIP solver to update its internal incumbent, so powerful refinement heuristics such as RINS [8] are never activated. In principle, this drawback could be fixed by implementing an ad-hoc RINS procedure feeded by an external infeasible solution $\tilde{x}$ (rather than by the internal MIP incumbent), but this would make the interpretation of the final results more difficult as the way RINS is actually implemented would contribute to the success of the method, which is something that would contaminate our experiments.

We therefore investigated the following simple variant of the "proximity search with recentering" implementation of the previous subsection. We just replace cutoff constraint (4) with its "soft version"

$$f(x) \leq f(\tilde{x}) - \theta + z \tag{7}$$

where $z \geq 0$ is a continuous slack variable, and modify the proximity objective function to

$$\Delta(x, \tilde{x}) + Mz \tag{8}$$

where $M$ is a large positive value compared to the feasible values of $\Delta$. In this way, the reference solution $\tilde{x}$ can be provided on input to the sub-MIP as a feasible (though very expensive) warm-start solution to be used to initialize the incumbent and to trigger the internal refinement heuristic. Of course, execution is aborted as soon as a new incumbent with $z = 0$ is found, meaning that a $\theta$-improving solution has been found.

Note that the presence of $z$ does not reduce the relaxation grip of the method, as the root-node relaxation solution $x'$ will have $z = 0$ because of the bigM penalty $M$ in the new objective function. Setting a too aggressive (i.e., large) threshold $\theta$ would interfere with this crucial property of proximity search. As a matter of fact, in the extreme case where $\theta = f(\tilde{x})$ one would get $z = f(x)$ hence (essentially) the original objective function would reappear in (8), making our approach useless.

# 5 Computational results

Proximity search (in the three variants just outlined) was implemented in C, and was used on top of the commercial solver `IBM ILOG Cplex` 12.4—called just `Cplex` in the following. The very first feasible solution $\tilde{x}$ at Step 1 was obtained by running `Cplex` for a small number of nodes and taking the best solution found. This produced a first feasible solution of reasonable quality, i.e., not unrealistically bad nor too good to be improved.

All runs were executed on an Intel i5-750 @ 2.67GHz in single-thread mode without any other concurrent thread—thus allowing for a reliable timing of the methods. In the forthcoming tables, geometric means are all shifted by 0.01, while computing times are in CPU seconds and do not include the time to find the initial solution $\tilde{x}$ at Step 1.

## 5.1 Testbed

We have conducted our experiments on 130 medium-to-hard instances belonging to three very different MIP classes.

**Pure 0-1 Integer Linear case: set covering instances**

As a representative of pure 0-1 Integer Linear Programs, we considered the NP-hard set covering problem. Our testbed was made by 49 hard set covering instances from the literature, namely:

- thirty instances `scpnre*`, `scpnrf*`, `scpnrg*`, `scpnrh*`, `scpcyc*` and `scpclr*` available in the ORLIB (`http://people.brunel.ac.uk/~mastjjb/jeb/info.html`);
- six `rail*` instances, also available in the ORLIB repository, and derived from railway applications. Note that we removed instance `rail516` that was solved to proven optimality within 10 nodes;
- four `railX*c` (c for core) also used in [10], and obtained as a core problem of the associated `rail*` instances;
- the nine instances from the recently proposed MIPLIB2010 library of instances (see [13]) that correspond to set covering problems, including the notoriously hard instance `seymour` and instances `ex1010-pi,`

`ramos3, sts405` and `sts729` that are classified as "hard" and for which the optimal solution value is still unknown.

At Step 1, the initial solution $\tilde{x}$ was obtained by running `Cplex` (default mode, single thread) for 10 enumeration nodes. As to threshold $\theta$, we defined $\theta = 1$ as our set-covering instances have integer costs.

**0-1 Mixed-Integer Linear case: network design instances**

To test our method on instances involving both binary and continuous variables, we addressed the hard capacitated network design problems taken from the SNDlib library [15], namely, the 25 instances considered in [16] and available at `www.zib.de/raack/downloads/INOC2007_instances/`.

To provide a reasonably good initial solution $\tilde{x}$ at Step 1, we initially ran `Cplex` for 5,000 enumeration nodes; four of the 25 instances were solved to proven optimality during this step, hence they were removed from the testbed. As the optimal value of these instances involves several digits, to avoid numerical precision issues we defined $\theta = 0.01|f(\tilde{x})|$.

**0-1 Mixed-Integer Nonlinear case: machine learning classification instances**

As to nonlinear 0-1 MIPs, we considered a set of sixty 0-1 Mixed-Integer Convex Quadratic Programs related to data classification through a so-called "ramp-loss Support Vector Machine with linear kernel" [6]. These instances were kindly provided by J.P. Brooks and are available, on request, from the authors; they are of the form

$$\min \frac{1}{2}||w||^2 + C(\sum_{i=1}^{n} \xi_i + 2\sum_{i=1}^{n} z_i) \tag{9}$$

$$y_i(w^T x_i + b) \geq 1 - \xi_i - M z_i \quad \forall i = 1, \cdots, n \tag{10}$$

$$(w, b) \in \Re^{m+1} \tag{11}$$

$$0 \leq \xi_i \leq 2 \quad \forall i = 1, \cdots, n \tag{12}$$

$$z_i \in \{0, 1\} \quad \forall i = 1, \cdots, n \tag{13}$$

where $(x_i, y_i) \in \Re^m \times \{-1, 1\}$ $(i = 1, \cdots, n)$ are input data, $M$ is a large positive value used to deactivate constraint (10) when $z_i = 1$, and $C$ is an input

parameter representing a trade-off in maximizing margin versus minimizing error. Although not strictly required by our method, to ease our implementation (that assumes a linear objective function) we used a customary transformation and restated the quadratic objective as $q + C(\sum_i \xi_i + 2\sum_i z_i)$ while adding the quadratic constraint $||w||^2 \leq 2q$. For classification instances, the initial solution $\tilde{x}$ at Step 1 was obtained by running `Cplex` for 50,000 enumeration nodes, and $\theta = 0.001|f(\tilde{x})|$ was used.

## 5.2 Comparison metric

To compare the performances of different primal heuristics, we use an indicator recently proposed by [1, 3], aimed at measuring the trade-off between the computational effort required to produce a solution and the quality of the solution itself. In particular, let $\widetilde{z}_{opt}$ denote the optimal solution value for (1)–(3) and $z(t)$ be the value of the best heuristic solution found at a time $t$. Then, a *primal gap function* $p$ can be computed as

$$p(t) = \begin{cases} 1 & \text{if no incumbent found until time } t \\ \gamma(z(t)) & \text{otherwise} \end{cases} \tag{14}$$

where $\gamma(\cdot) \in [0, 1]$ is the *primal gap*, defined as follows

$$\gamma(z) = \begin{cases} 0 & \text{if } |\widetilde{z}_{opt}| = |z| = 0, \\ 1 & \text{if } \widetilde{z}_{opt} \cdot z < 0, \\ \frac{z - \widetilde{z}_{opt}}{\max\{|\widetilde{z}_{opt}|, |z|\}} & \text{otherwise.} \end{cases} \tag{15}$$

Finally, the *primal integral* of a run until time $t_{\max}$ is defined as

$$P(t_{\max}) = \int_0^{t_{\max}} p(t)\, \mathrm{d}t \tag{16}$$

and is actually used to measure the quality of primal heuristics—the smaller $P(t_{\max})$ the better the expected quality of the incumbent solution if we stopped computation at an arbitrary time before $t_{\max}$.

## 5.3 Results

We compared the following methods:

- `proxy_norec`: the proximity search algorithm without recentering described in Section 4.1;

- `proxy_rec`: the proximity search algorithm with recentering described in Section 4.2;
- `proxy_incum`: the proximity search algorithm with an incumbent solution, described in Section 4.3; $M = 100,000$ was used to penalize the extra variable $z$ in the objective function, and `Cplex` parameter CPX_PARAM_EPAGAP was set to $M/2$ so as to abort the sub-MIP as soon as a feasible solution with $z = 0$ is found;
- `cplex_def`: `Cplex` run in its default setting;
- `cplex_heu`: `Cplex` run in its heuristic setting, i.e., with parameter CPX_PARAM_MIPEMPHASIS set to CPX_MIPEMPHASIS_FEASIBILITY;
- `cplex_no_cuts`: `Cplex` run in default setting but disabling all cut-generation routines;
- `cplex_polish`: `Cplex` run in default setting but executing the polishing heuristic at each branch-and-bound node;
- `cplex_gui_div`: `Cplex` run with parameter CPX_PARAM_DIVETYPE set to 3 (guided dive), to visit the search tree so as to explore the neighborhood of the incumbent solution first;
- `locBra_orig`: the original local-branching implementation from [10];
- `locBra_aggr`: an aggressive variant of local branching where constraint (6) with $k = 10$ is initially imposed; if no improving solution exists in the given neighborhood, the search is repeated with an increased value of $k = 20, 30, \ldots$ until an improving solution is found or the time limit is reached; if an improving solution $\tilde{x}$ is found, the method is iterated on the new incumbent in a recentering fashion, starting again with $k = 10$; at each MIP-solver call, the current incumbent $\tilde{x}$ is provided as a warm-start input solution.

We also ran a "zero-objective" version of proximity search (not reported in the forthcoming tables) where $\Delta(x, \tilde{x})$ was just replaced by a null objective function, and verified that this change resulted in a dramatic performance deterioration—thus confirming the importance of using the proximity objective function.

**Set covering instances**

| | 5 | 10 | 30 | 60 | 120 | 300 | 600 | 1200 |
|---|---|---|---|---|---|---|---|---|
| proxy_norec | 0.132 | 0.215 | 0.452 | 0.703 | 1.090 | 1.886 | 2.851 | 4.247 |
| cplex_def | 0.178 | 0.310 | 0.698 | 1.121 | 1.753 | 2.880 | 4.108 | 5.775 |
| cplex_heu | 0.174 | 0.305 | 0.703 | 1.113 | 1.671 | 2.697 | 3.774 | 5.086 |
| cplex_no_cuts | 0.176 | 0.305 | 0.694 | 1.138 | 1.760 | 2.865 | 3.949 | 5.301 |
| cplex_gui_div | 0.175 | 0.297 | 0.651 | 1.031 | 1.594 | 2.605 | 3.565 | 4.750 |
| proxy_incum | 0.124 | 0.195 | 0.374 | 0.550 | 0.797 | 1.232 | 1.600 | 1.978 |
| proxy_rec | 0.122 | 0.198 | 0.400 | 0.599 | 0.858 | 1.335 | 1.749 | 2.182 |
| locBra_orig | 0.170 | 0.278 | 0.551 | 0.803 | 1.122 | 1.722 | 2.304 | 2.900 |
| locBra_aggr | 0.121 | 0.192 | 0.376 | 0.561 | 0.773 | 1.157 | 1.533 | 1.974 |
| cplex_polish | 0.181 | 0.298 | 0.596 | 0.876 | 1.251 | 1.895 | 2.498 | 3.252 |

**Network design instances**

| | 5 | 10 | 30 | 60 | 120 | 300 | 600 | 1200 |
|---|---|---|---|---|---|---|---|---|
| proxy_norec | 0.088 | 0.138 | 0.272 | 0.406 | 0.608 | 1.029 | 1.442 | 1.952 |
| cplex_def | 0.104 | 0.178 | 0.412 | 0.652 | 0.919 | 1.347 | 1.784 | 2.238 |
| cplex_heu | 0.105 | 0.177 | 0.374 | 0.542 | 0.768 | 1.157 | 1.437 | 1.749 |
| cplex_no_cuts | 0.104 | 0.172 | 0.365 | 0.567 | 0.858 | 1.415 | 2.056 | 2.926 |
| cplex_gui_div | 0.102 | 0.172 | 0.366 | 0.539 | 0.739 | 1.064 | 1.438 | 1.927 |
| proxy_incum | 0.084 | 0.129 | 0.233 | 0.317 | 0.411 | 0.529 | 0.629 | 0.742 |
| proxy_rec | 0.091 | 0.147 | 0.288 | 0.424 | 0.591 | 0.841 | 1.035 | 1.262 |
| locBra_orig | 0.099 | 0.160 | 0.340 | 0.536 | 0.793 | 1.149 | 1.395 | 1.633 |
| locBra_aggr | 0.096 | 0.156 | 0.308 | 0.447 | 0.613 | 0.912 | 1.204 | 1.459 |
| cplex_polish | 0.107 | 0.186 | 0.419 | 0.658 | 0.979 | 1.422 | 1.634 | 1.853 |

**Classification instances**

| | 5 | 10 | 30 | 60 | 120 | 300 | 600 | 1200 |
|---|---|---|---|---|---|---|---|---|
| proxy_norec | 0.142 | 0.229 | 0.489 | 0.788 | 1.268 | 2.368 | 3.825 | 6.182 |
| cplex_def | 0.212 | 0.376 | 0.935 | 1.660 | 2.983 | 6.447 | 11.492 | 19.687 |
| cplex_heu | 0.214 | 0.379 | 0.956 | 1.723 | 3.123 | 6.859 | 12.453 | 21.834 |
| cplex_no_cuts | 0.194 | 0.340 | 0.841 | 1.480 | 2.607 | 5.516 | 9.685 | 15.908 |
| cplex_gui_div | 0.193 | 0.330 | 0.780 | 1.360 | 2.393 | 5.028 | 8.738 | 14.505 |
| proxy_incum | 0.104 | 0.146 | 0.240 | 0.313 | 0.406 | 0.580 | 0.772 | 1.045 |
| proxy_rec | 0.107 | 0.153 | 0.260 | 0.359 | 0.492 | 0.754 | 1.058 | 1.486 |
| locBra_orig | 0.144 | 0.216 | 0.402 | 0.576 | 0.781 | 1.094 | 1.382 | 1.744 |
| locBra_aggr | 0.134 | 0.206 | 0.389 | 0.569 | 0.836 | 1.423 | 2.166 | 3.305 |
| cplex_polish | 0.209 | 0.339 | 0.664 | 0.960 | 1.365 | 2.140 | 2.961 | 4.030 |

Table 3: Geometric mean of primal integrals after 5, 10, ..., 1200 sec.s; the lower the better.

All algorithms received the same input solution $\tilde{x}$ at the very beginning of the search.

Table 3 gives a summary of our results on set covering, network design, and classification instances. Each table row reports primal-integral geometric means for a given algorithm after 5, 10, 30, 60, 120, 300, 600, and 1200 seconds, respectively. Methods are grouped in two classes: the top part of the table refers to (potentially) exact methods that develop a single search tree, while the bottom part refers to refining heuristics that generate multiple (partial) search trees. Figure 2 plots the evolution of the primal-integral geometric mean of all competing methods, on the whole testbed.

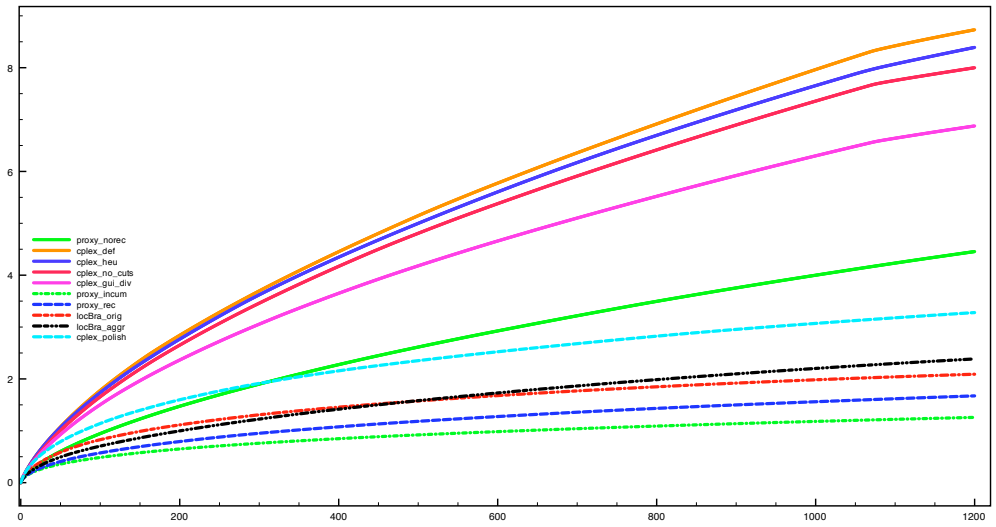


Figure 2: Primal integrals (geometric means) over time on the whole testbed; the lower the better.

According to the reported results, all proximity-search variants have a quite satisfactory performance on the instances in our testbed. In particular, `proxy_incum` qualifies as the best approach, as its primal-integral geometric mean is substantially better than that of all competing methods.

Figure 3 reports a pairwise comparison of the performance of some algorithms over the whole testset including all set-covering, network design, and classification instances. Given a certain pair of competing algorithms $\mathtt{A}_0$ and $\mathtt{A}_1$ (say), each subfigure reports the probability of a method to produce a "significantly better" solution than its competitor after $t = 1, 2, \cdots, 1200$ sec.s,

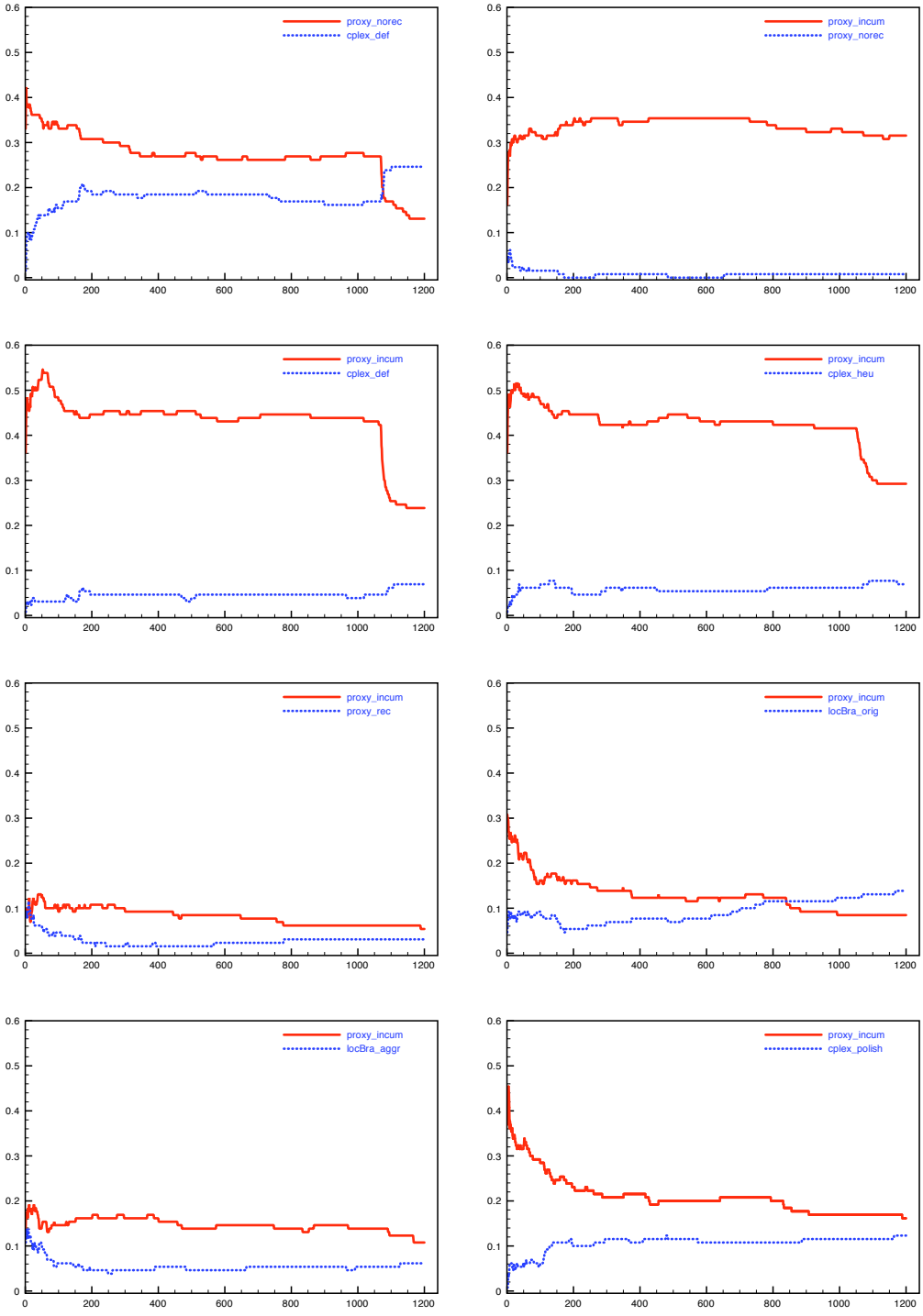


Figure 3: Pairwise performance comparison (probability of being at least 1% better than the competitor) over time; the higher the better.

where "significantly better" refers to a saving of 1% or more. To be specific, for $i \in \{0,1\}$ let $heu(\mathtt{A}_i, t)$ denote the value of the best heuristic solution found by heuristic $\mathtt{A}_i$ at time $t$. Then $\mathtt{A}_i$ is is considered to be significantly better than $\mathtt{A}_{1-i}$ at time $t$ if

$$heu(\mathtt{A}_i, t) \leq heu(\mathtt{A}_{1-i}, t) - \max\{10^{-6}, 0.01 \cdot \min\{|heu(\mathtt{A}_0, t)|, |heu(\mathtt{A}_1, t)|\}\}$$

The performance of `cplex_no_cuts` and of `cplex_gui_div` is very similar to that `cplex_heu`, hence the corresponding plots are not reported. According to the figure, `proxy_incum` dominates all competitors even in this pairwise comparison, with the only exception of `locBra_orig` that becomes better (under this metric) than `proxy_incum` after about 800 sec.s.

Figure 4 illustrates solution updatings during the first 30 seconds for the set-covering instances `neos-1616732` and `scpnrg4` (`cplex_def` vs `proxy_incum`). The left-hand-side figure shows a typical situation where proximity search frequently updates the incumbent solution, while `Cplex` spends a large amount of time for even solving the root relaxation. On the other hand, situations occur where the amount of improvement obtained by `Cplex` in a single step is much larger than the one that can be obtained by multiple updatings of `proxy-search`, as illustrated in the right-hand-side figure.

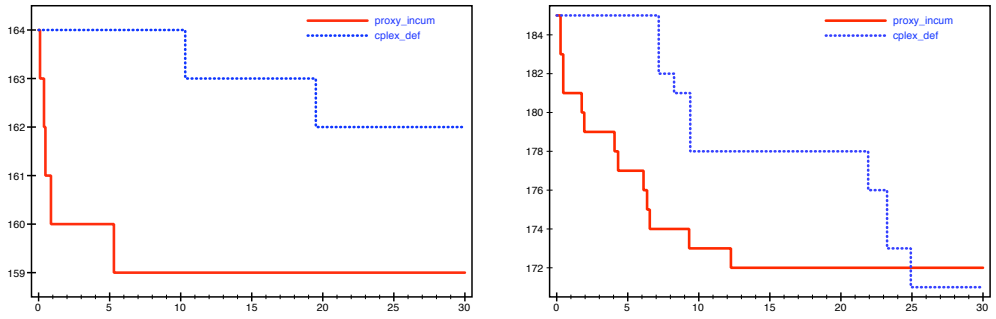


Figure 4: Solution updatings for set covering instances `neos-1616732` (left) and `scpnrg4` (right) over time; the lower the better.

As already mentioned, the use of the proximity objective function can greatly speedup the solution of the convex relaxation at each tree-search node, and in particular at the root node where the reference solution $\tilde{x}$ would be trivially optimal without the cutoff constraint.

This behavior is well exemplified by the set-covering instance `ramos3` that belongs to the "Challenge Reoptimize" class of MIPLIB 2010, meaning that its LP (re)optimization is cumbersome. Indeed, just solving the very first LP relaxation with algorithm `cplex_def` (1 thread) takes 43 CPU seconds on our PC, while the root node requires 98 sec.s in total. Starting with a reference solution of value 267, this algorithm is able to find an improved solution $\hat{x}$ of value 255 and distance $\Delta(\hat{x}, \tilde{x}) = 470$ at node 10, after 1163 sec.s.

Instead, algorithm `proxy_norec`, feeded with the same initial solution, solves the initial LP in just 0.03 sec.s, and improves twice the incumbent at the end of the root node, obtaining a solution of value 265 and $\Delta$-distance 3 after 0.11 sec.s. At node 149, after 51 sec.s and several updatings, the incumbent has value 252, whereas the algorithm reaches a solution of value 241 after 156 sec.s (at node 2000), and is not able to improve it within the 1200-sec. time limit.

Our second variant of proximity search, namely algorithm `proxy_rec`, produces better results, as most reoptimizations require no branching at all and find an improved solution at very small $\Delta$-distance from the previous incumbent: after about one second, the incumbent has value 261, whereas the incumbent is 237 after 75 sec.s, and is equal to 232 after 1136 sec.s.

Even better results are obtained using algorithm `proxy_incum`: due to the availability of an initial incumbent solution, all sub-MIPs but two require no branching, making `proxy_incum` extremely effective on this instance. Indeed, `proxy_incum` finds a solution of value 232 after just 131 sec.s, and a final feasible solution of value 229 after 596 sec.s.

# 6 Proximity search and primal methods

As already observed, our proximity search scheme has a *primal* nature, meaning that it produces a sequence of improved solutions that eventually leads to an optimal one. This is in contrast with, e.g., MIP cutting plane methods that have a *dual* nature and eventually reach the optimal feasible solution through a sequence of more-than-optimal (infeasible) solutions.

The dichotomy between primal and dual methods is well exemplified by considering the behavior of the most famous LP solution method, the simplex algorithm, in its primal and dual version. It is well known that, on typical instances, the dual method outperforms its primal counterpart by a

significant margin, even when the LP instance is solved from scratch. Indeed, phase 1 of the dual version is typically very fast, and after a reasonable number of pivots the dual basis gets closer and closer to feasibility. On the other hand, the primal simplex method suffers from a typically more time consuming phase 1, after which it is forced to visit a sequence of (possible very close one to each other) feasible vertices before reaching the optimum—evidently, this is a too conservative search policy that does not pay off in general. Note however that the primal method is much more satisfactory in terms of "behavior as a heuristic", in that stopping the dual algorithm before convergence does not produce any feasible solution at all. This drawback of dual methods can play a role for very difficult LPs that cannot be solved to optimality within the allowed time limit.

In spite of the above weaknesses, the primal simplex method is still the method of choice in certain settings—most notably, in the column generation framework. Indeed, phase 1 is no longer needed in this context, and one can reasonably assume that the optimal LP solution after the addition of some new columns will not be too far from the previous optimal one—the main so after the very first iterations, and by using some stabilization technique to control the variation of dual variables.

In our view, just because of its primal nature, proximity search can be very useful precisely in the same setting where the primal simplex is preferred to its dual counterpart, and in particular in a column generation context where finding an initial solution is not an issue and reoptimization produces a solution that is close to the previous incumbent.

To validate the above claim, we implemented the following simplified column-generation setting intended for mixed-integer linear instances with a very large number of variables. We start with an initial identity LP basis containing an artificial variable (with a very large cost) for each row. Then we enter a column-generation loop where the (at most) 100 variables with largest (in absolute value) negative reduced cost are added to the current formulation, until no such a column exists, i.e., the optimal LP solution is obtained. As soon as all artificial variables are carried out of the LP basis, a heuristic algorithm `A` is executed on the subinstance of the original problem associated with the current set of columns, in the attempt of improving the incumbent integer solution.

Before algorithm `A` is applied, we compute a near optimal solution $\tilde{x}$ to the *previous* subinstance (i.e., without the columns that have been added in the very last iteration) by running `Cplex` with a large maximum number

of nodes (1,000). This solution is then given as a warm-start incumbent to algorithm `A`, that is executed with a short time limit (10 seconds) in the attempt of improving it.

The experiment is aimed at evaluating the capability of algorithm `A` to possibly improve the current incumbent solution when a small set of new columns is available. In our test, some of the algorithms described in Section 5.3 were used to play the role of algorithm `A`. In particular, we considered

- `Cplex` in its default settings, and in those settings that produced the best results on set covering instances of Table 3, namely using polishing heuristic, and using guided dives;
- local branching in its aggressive version;
- our proximity search algorithm, in its three variants.

We stress again that all methods receive exactly the same subinstance and the same warm-start solution $\tilde{x}$, so their outcomes are comparable. The external column-generation scheme is only used to create a sequence of subinstances with nested variable sets, and is not affected at all by the performance of `A`.

In our computational experiments, we considered the 7 large-scale set-covering instances named `rail*` that are suitable for column generation as they involve a very large number of variables, and produced a total number of 603 subinstances. Table 4 reports, for each instance, the number of subinstances that have been solved (#sub.), and compares the algorithms mentioned above by reporting, for each algorithm, the number of "wins", i.e., of subinstances for which it provided the best solution among all algorithms (because of ties, a single subinstance can have more than one winner). Again, `proxy_incum` qualifies as the most effective method (in total, 489 wins out of 603 runs), followed by `locBra_aggr` (442 wins), `proxy_rec` (424 wins), and `proxy_norec` (382 wins). All `Cplex` methods are instead less effective in this incremental framework (about 300 wins).

# 7 Conclusions and ideas for future research

We have investigated the use of a proximity objective function in integer programming, and have designed a *proximity search* heuristic for 0-1 MIPs. Different implementations of the basic idea have been proposed and tested

| instance | #sub. | cplex_def | cplex_gui_div | cplex_polish | locBra_aggr | proxy_norec | proxy_rec | proxy_incum |
|---|---|---|---|---|---|---|---|---|
| rail507 | 27 | 19 | 23 | 21 | 21 | 23 | 22 | 20 |
| rail516 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 |
| rail582 | 23 | 14 | 15 | 15 | 14 | 14 | 17 | 15 |
| rail2536 | 143 | 92 | 95 | 96 | 114 | 110 | 125 | 132 |
| rail2586 | 109 | 48 | 48 | 50 | 68 | 66 | 71 | 92 |
| rail4284 | 131 | 42 | 41 | 46 | 85 | 65 | 72 | 102 |
| rail4872 | 148 | 50 | 51 | 51 | 118 | 82 | 95 | 106 |
| sum | 603 | 287 | 295 | 301 | 442 | 382 | 424 | 489 |

Table 4: Performance (number of wins) of incremental heuristics on rail* set-covering instances.

on a set of medium-to-hard set covering, network design, and classification instances, with the aim of evaluating pros and cons of the basic method.

In our view, a clear pro of proximity search is its capability of quickly improving the first MIP incumbents. In this respect, we are confident that proximity search will find soon its way in general-purpose MIP solvers—an implementation in the open-source CBC and GLPK frameworks are already available. A natural setting is to use proximity search with recentering for a significant amount of time at the root node, and use it after each incumbent update (or with a certain frequency during tree exploration) but with a much smaller internal time/node limit.

A very important pro of proximity search is that convex (re)optimization can sometimes be orders of magnitude faster when the proximity function is used instead of the original objective. In some cases, proximity search is able to update several times the incumbent in a fraction of the time that would be needed to solve the initial convex relaxation with the original objective function.

As to cons, the need to define a hard cutoff value is perhaps the most relevant one. In our experiments, we decided to be very conservative to avoid to overtune our method and to contaminate its outcomes. We are confident however that a more effective cutoff policy can be implemented, in particular for specific classes of problems.

The primal nature of proximity search is both a con and a pro of the method. It is a con in the sense that proximity search can be trapped by a long series of small improvements, while a more aggressive dual policy can produce less frequent but much larger improvements. But the primal nature

of proximity search is also a pro, in particular in a re-optimization context such as column generation.

Future research should be devoted to the design of a clever hybrid approach that exploits proximity search, local branching, and RINS in a synergic way, possibly tailored for specific classes of hard problems.

Also, the use of a proximity objective function in MIP-based metaheuristics appears to be an interesting research topic. In this respect, proximity search is closely related to the local branching paradigm of [10], and could be considered as a viable alternative to the latter in some of its numerous applications. In our view, a promising framework—called *MIP-and-refine* in [12]—is to run first a fast ad-hoc heuristic (possibly not based on an explicit underlying optimization model), so as to quickly produce an initial solution to be refined through proximity search based on a (possibly weak) MIP model. The good news is that even weak models (notably, those making extensive use of big-M tricks) can become effective when used within a proximity search framework, due to the improved relaxation grip induced by the proximity objective function.

The use of the proximity objective function in column generation is another interesting topic that deserves further investigation. One option has been exemplified in Section 6, and consists of just applying proximity search to a clone of the current master problem, without affecting pricing of the new columns. A more intriguing possibility is to use the proximity objective function also in the pricing phase, with the aim of stabilizing the overall branch-and-cut-and-price scheme.

# Acknowledgements

This research was supported by the *Progetto di Ateneo* on "Computational Integer Programming" of the University of Padova, and by MiUR, Italy (PRIN project "Integrated Approaches to Discrete and Nonlinear Optimization"). We thank J.P. Brooks who provided us with the classification instances.

# References


[1] T. Achterberg, T. Berthold, and G. Hendel. Rounding and propagation heuristics for mixed integer programming. *Operations Research Proceedings 2011*, pages 71–76, 2012.

[2] L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4(1):63–76, 2007.

[3] T. Berthold. Measuring the impact of primal heuristics. ZIB-Report 13-17, Zuse Institute Berlin, 2013.

[4] N. L. Boland, A. C. Eberhard, F. G. Engineer, and A. Tsoukalas. A new approach to the feasibility pump in mixed integer programming. *SIAM Journal on Optimization*, 22(3):831–861, 2012.

[5] P. Bonami, G. Cornuéjols, A. Lodi, and F. Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, 119(2):331–352, 2009.

[6] J. P. Brooks. Support vector machines with the ramp loss and the hard margin loss. *Operations Research*, 59(2):467–479, 2011.

[7] A. Daniilidis and C. Lemarechal. On a primal-proximal heuristic in discrete optimization. *Mathematical Programming*, 104(1):105–128, 2005.

[8] E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102(1):71–90, 2005.

[9] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

[10] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98(1–3):23–47, 2003.

[11] M. Fischetti and A. Lodi. *Heuristics in Mixed Integer Programming*, volume 8, pages 738–747. John Wiley & Sons, Inc., 2011.

[12] M. Fischetti, G. Sartor, and A. Zanette. A MIP-and-refine matheuristic for smart grid energy management. *International Transactions in Operations Research*, pages 1–11, 2013, DOI: 10.1111/itor.12034.

[13] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R.E. Bixby, E. Danna, G. Gamrath, A.M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D.E. Steffy, and K. Wolter. MIPLIB 2010 mixed integer programming library version 5. *Mathematical Programming Computation*, 3(2):103–163, 2011.

[14] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Verlag, 2009.

[15] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski. SNDlib 1.0 - survivable network design library. *Networks*, 55(3):276–286, 2010.

[16] C. Raack, A. Koster, S.Orlowski, and R. Wessäly. On cut-based inequalities for capacitated network design polyhedra. *Networks*, 57(2):141–156, 2011.

[17] E. Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.