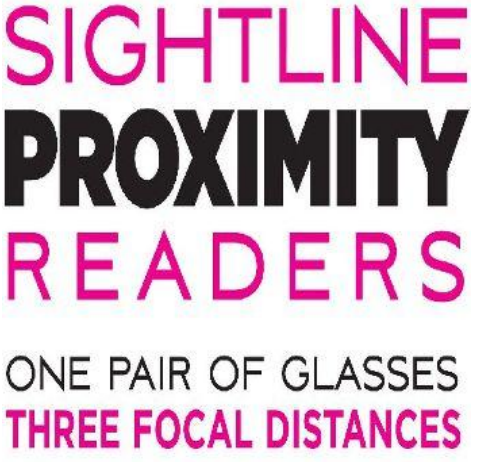
Proximity search

Matteo Fischetti, Michele Monaci
University of Padova



MIP heuristics

- We consider a Mixed-Integer convex 0-1 Problem (0-1 MIP, or just MIP)

$$\begin{aligned} \min\, & f(x) \\ & g(x) \leq 0 \\ x_j \in \{0,1\} \quad & \forall j \in J \end{aligned}$$

  where *f* and *g* are convex functions and $\emptyset \subset J \subseteq N := \{1, \cdots, n\}$

  → removing integrality leads to an easy-solvable continuous relaxation

- A **black-box** (exact or heuristic) MIP solver is available

- How to use the solver to **quickly** provide a **sequence of improved heuristic solutions** (time vs quality tradeoff)?

Large Neighborhood Search

- **Large Neighborhood Search** (LNS) paradigm:
  1. introduce **invalid constraints** into the MIP model to create a nontrivial sub-MIP "centered" at a given heuristic sol. $\tilde{x}$ (say)
  2. Apply the MIP solver to the sub-MIP for a while…

- Possible implementations:
  - **Local branching**: add the following linear cut to the MIP

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

  - **RINS**: find an optimal solution x^* of the continuous relaxation, and fix all binary variables such that $x^*_j = \tilde{x}_j$
  - **Polish**: evolve a population of heuristic sol.s by using RINS to create offsprings, plus mutation etc.

Why should the subMIP be easier?

- What makes a (sub)MIP easy to solve?

  1. fixing many var.s reduces problem size & difficulty
  2. additional contr.s limit **branching's** scope
  3. **something else**?

- In Branch-and-**Bound** methods, the quality of the root-node relaxation is of paramount importance as the method is **driven by** the relaxation solution found at each node

- Quality in terms of **integrality gap** …
- … but also in term of "**similarity**" of the root node solution to the optimal integer solution (the "more integer" the better…)

Relaxation grip

- Effect of **local branching** constr. for various values of the neighborhood radius *k* on MIPLIB2010 instance *ramos3.mps* (root node relaxation)

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.

No Neighborhood Search

- We investigate a different approach to get improved **relaxation grip**
  … where no (risky) invalid constraints are added to the MIP model
  … but the **objective function** is altered somehow to improve grip

A naïve question: what is the role of the MIP objective function?

1. Obviously, it defines the criterion to select an "optimal" solution

But also

2. It shapes the search path towards the optimum, as well all the internal heuristics

The objective function role

- Altering the objective function can have a big impact in

  - **time to get the optimal solution of the continuous relax.**

    working with a simplified/different objective can lead to huge speedups (orders of magnitude)

  - **success of the internal heuristics (diving, rounding, …)**

    the original objective might interfere with heuristics (no sol. found even for trivial set covering probl.s) and sometimes is reset to zero

  - **search path towards the integer optimum**

    search is trapped in the upper part of the tree (where the lower bounds are better), with frequent divings to grasp far-away (in terms of lower bound) solutions

Proximity search

- We want to be free to work with a modified objective function that has a better heuristic "grip" and hopefully allows the black-box solver to quickly improve the incumbent solution $\tilde{x}$

- **"Stay close" principle**: we bet on the fact that improved solutions live in a close neighborhood (in terms of Hamming distance) of the incumbent, and we want to attract the search within that neighborhood

- **Step 1.** Add an explicit **cutoff** constraint $f(x) \leq f(\tilde{x}) - \theta$

- **Step 2**. Replace the objective $f(x)$ by **the proximity function**

$$\Delta(x, \tilde{x}) = \sum_{j \in J: \tilde{x}_j = 0} x_j + \sum_{j \in J: \tilde{x}_j = 1} (1 - x_j) = ||x - \tilde{x}||_J^2$$

A path following heuristic

1. run a black-box solver on the original 0-1 MIP, until a "reasonably good" feasible solution $\tilde{x}$ is found;

repeat

2. explicitly add the *cutoff constraint* $f(x) \leq f(\tilde{x}) - \theta$ to the MIP model, where $\theta > 0$ is a given parameter;
3. replace the objective function $f(x)$ by a new "proximity" one, say $\Delta(x, \tilde{x})$;
4. run the MIP solver on the new model until a termination condition is reached, and let x^* be the best feasible solution found;
5. **if** $J \subset N$ **then** refine x^* by solving the convex program $x^* := \text{argmin}\{f(x) : g(x) \leq 0,\ x_j = x_j^*\ \forall j \in J\}$;
6. recenter $\Delta(x, \cdot)$ by setting $\tilde{x} := x^*$, and/or update θ

until *an overall termination condition is reached*;

return $\tilde{x}$

Relaxation grip

- Effect of the cutoff constr. for various values of parameter θ on MIPLIB2010 instance **ramos3** (root node relaxation)

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
Δ-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 θ.

Related approaches

- Exploiting locality in optimization is of course not a new idea
  - Augmented Lagrangian
  - Primal-proximal heuristic for discrete opt. (Daniilidis & Lemarechal ‘05)
  - Can be seen as dual version of local branching
  - Feasibility Pump can be viewed as a proximal method (Boland et al. ‘12)
  - …

- However we observe that (as far as we know):
  - the approach was never analyzed computationally in previous papers
  - the method was not previously embedded in any MIP solver
  - the method has PROs and CONs that deserve investigation

Possible implementations

- The way a computational idea is actually implemented (not just coded) matters

- Computational experience shows how difficult is to evaluate the real impact of a new idea, mainly when hybrid versions are considered and several parameters need be tuned → the so-called **Frankenstein effect**

- **Stay clean: in our analysis, we deliberately avoided considering hybrid versions of proximity search (mixing objective functions, using RINS-like fixing, etc.), though we guess they can be more successful than the basic version we analyzed**

Proximity search without recentering

Each time a feasible solution x* is found

- record it
- update the right-hand side of the cutoff constraint (this makes x* infeasible, so the solver incumbent is never defined)
- continue without changing the objective function

PROs:

- a single tree is explored, that eventually proves the optimality of the incumbent (modulo the theta-tolerance)

CONs:

- callbacks need to be implanted in the solver (gray-box) → some features can be turned off automatically
- the proximity function remains "centered" on the first solution

Proximity search with recentering

As soon as a feasible solution *x** (say) is found, abort the solver and

- Update the right-hand side of the cutoff constraint
- Redefine the objective function as $\Delta(x, x^*)$
- Re-run the solver from scratch

CONs:

- several overlapping trees are explored (wasting computing time)
- the root node is solved several times → time-consuming cuts should be turned off, or computed at once and stored?

PROs:

- Easily coded (no callbacks)
- proximity function automatically "recentered" on the incumbent

Proximity search with incumbent

- Both methods above work **without an incumbent** (as soon a better integer sol. is found, we cut it off) → powerful internal tools of the black-box solver (including RINS) are never activated

- Easy workaround: **soft** cutoff constraint (slack *z* with BIGM penalty)

$$\begin{aligned} \textit{min} \quad & \Delta(x, \tilde{x}) + Mz \\ & f(x) \leq f(\tilde{x}) - \theta + z \\ & \ldots \end{aligned}$$

- Hence any subMIP can be warm-started with the (high-cost but) feasible integer sol. $\tilde{x}$

Faster than the LP relaxation?

Example: very hard set-covering instance **ramos**3, initial solution of value 267

Cplex (default):

- initial LP relaxation: 43 sec.s, root node took 98 sec.s
- first improved sol. at node 10, after 1,163 sec.s: value 255, distance=470

Proximity search without recentering:

- initial LP relaxation: 0.03 sec.s
- end of root node, after 0.11 sec.s: sol. value 265, distance=3
- value 241 after 156 sec.s (200 nodes)

Proximity search with recentering:

- most calls require no branching at all
- value 261 after 1 sec., value 237 after 75 sec.s.

Proximity search with incumbent:

- value 232 after 131 sec.s, value 229 after 596 sec.s.

Computational tests

- We do not expect proximity search will work well in all cases…

  … because its **primal nature** can lead to a sequence of slightly-improved feasible solutions [cfr. Primal vs. Dual simplex]

  Three classes of 0-1 MIPs have been considered:

  – 49 hard **set covering** from the literature (MIPLIB 2010, railways)
  – 21 hard **network design** instances (SNDlib)
  – 60 MIPs with convex-quadratic constraints (**classification** instances related to SVM with ramp loss)

Compared heuristics

- **Proximity sear**ch vs. **Cplex** in different variants (all based on IBM ILOG Cplex 12.4)

proxy_norec
cplex_def
cplex_heu
cplex_no_cuts
cplex_gui_div
proxy_incum
proxy_rec
locBra_orig
locBra_aggr
cplex_polish

- All runs on an Intel i5-750 CPU running at 2.67GHz (single-thread mode)

Some plots

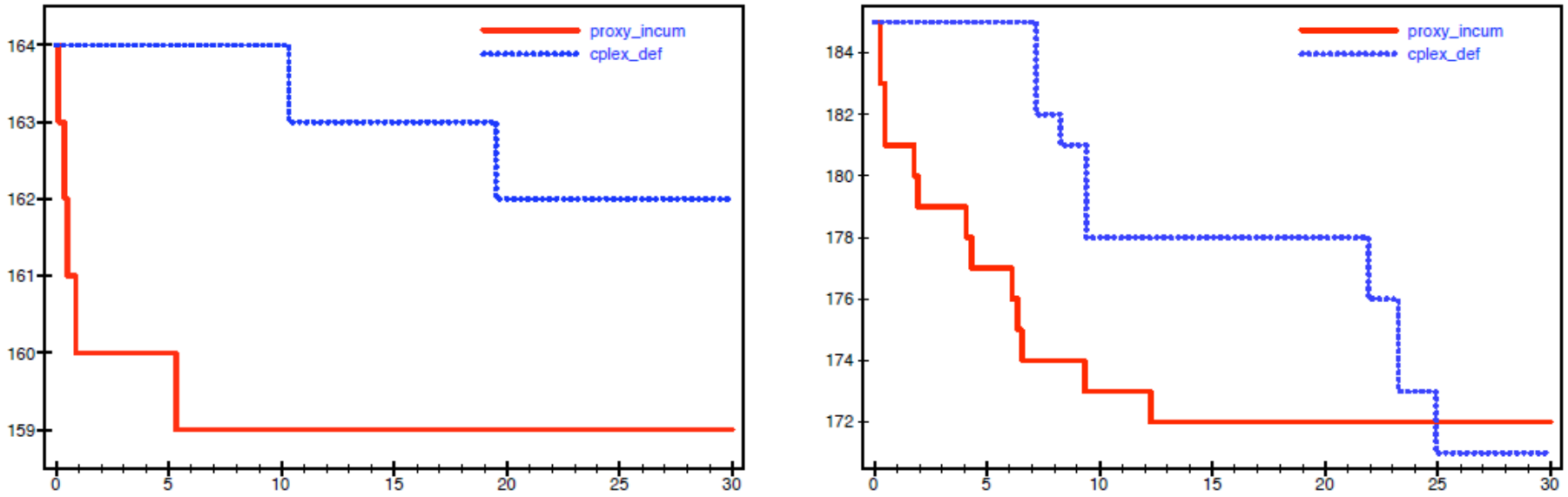


Figure 3 Solution updatings for set covering instances `neos-1616732` (left) and `scpnrg4` (right) over time; the lower the better.

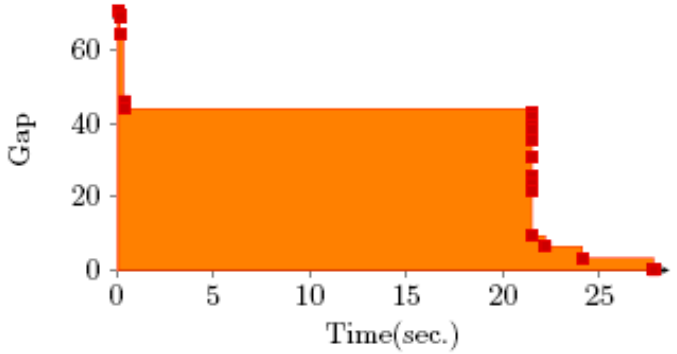
Comparision metric

- Trade-off between computing time and heuristic solution quality

- We used the **primal integral** measure recently proposed by

T. Achterberg, T. Berthold, and G. Hendel. Rounding and propagation heuristics for mixed integer programming. Technical report, ZIB 11-29, 2011.

where the history of the incumbent updates is plotted over time until a certain timelimit, and the relative-gap integral *P(t)* till time *t* is taken as performance measure (the smaller the better)



Cumulative figures

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

Primal integrals after 5, 10, …, 1200 sec.s (the lower the better)

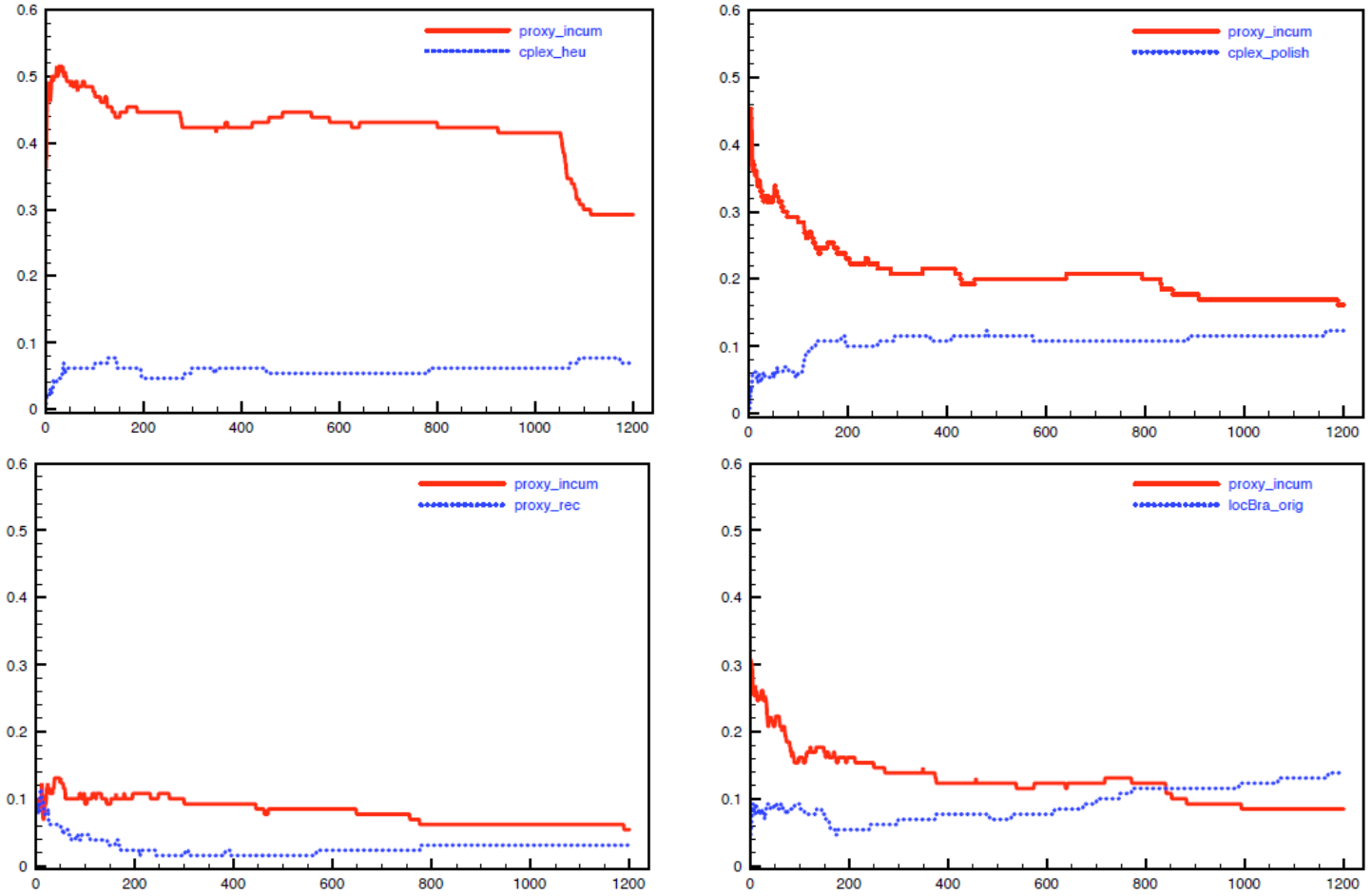
Network design instances

	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

Pairwise comparisons



Probability of being 1% better than the competitor (the higher the better)

Conclusions

- The objective function has a **strong impact** in search and can be used to improve the heuristic behavior of a black-box solver
- Even in a proof-of-concept implementation, proximity search proved quite successful in quickly improving the initial heuristic solution
- Proximity search has a **primal** nature, and is likely to be effective when improved solutions exist which are not too far (in terms of binary variables to be flipped) from the current one
- Worth to be implemented in open-source/commercial MIP solvers? Already available in COIN-OR CBC and in GLPK 4.51 …