Proceedings of the Twenty-Sixth RAMP Symposium

Proximity search heuristics for Mixed Integer Programs

Martina Fischetti[1*], Matteo Fischetti[2†], and Michele Monaci[2‡]

Abstract Large-Neighborhood Search heuristics for general Mixed-Integer Programs (MIPs) such as local branching and RINS define a neighborhood of the current incumbent by introducing invalid constraints into the MIP formulation, and use a black-box MIP solver to optimize the restricted problem. Proximity search is an alternative approach, still aimed at improving a given feasible solution: instead of modifying the constraints of the MIP at hand with the aim of reducing search space, one modifies the objective function to make the search easier. More specifically, proximity search replaces the objective function by a "proximity" one, with the goal of enhancing the heuristic behavior of the MIP black-box solver. The approach can be quite effective in quickly improving a given starting feasible solution, at least when the landscape of feasible solutions is not too irregular. In the present paper we first review the general proximity search paradigm, and then apply it to turbine layout optimization in a wind farm context. Computational results on very large scale instances prove the practical viability of the overall approach.

Keywords Mixed Integer Linear Programming, Heuristics, Wind Farm Optimization.

1. Introduction

What is the role of the objective function in optimization? This apparently naive question has a first obvious answer: the objective function defines the criterion to select the final optimal solution. However, the objective function plays a crucial role also to guide the search towards the optimal solution within a solution algorithm, and is not clear whether one should necessarily use the same function for both purposes. This is particularly true when the main goal of the search is to quickly provide a sequence of improved feasible solutions. The proximity search approach introduced in [5] is a refinement heuristic that replaces the objective function of the given model by a "proximity" one, with the goal of enhancing the heuristic behavior of the solver at hand.

Proximity search focuses on a generic 0-1 Mixed Integer (possibly nonlinear) Program (MIP, for short) of the form

$$\min f(x) \tag{1}$$

$$g(x) \leq 0 \tag{2}$$

1 Department of Electronic Systems, University of Aalborg, Denmark
2 Department of Information Engineering, University of Padova, Italy
∗ E-mail address: martina.fischetti@gmail.com
† E-mail address: matteo.fischetti@unipd.it
‡ E-mail address: michele.monaci@unipd.it

$$x_j \in \{0,1\} \quad \forall j \in J \tag{3}$$

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 the method, in the following we assume that both f and g are convex functions with the property that dropping the integrality condition in (3) leads to a polynomially solvable relaxation.

Proximity search starts with a feasible solution $\tilde{x}$, and adds 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 one is 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 paper is organized as follows. The proximity search approach is outlined in Section 2. Three different implementations of the basic approach are sketched in Section 3. Our optimal turbine location problem is described in Section 4, and heuristically solved by using the proximity search engine.

The present paper is based on [5], where the proximity search approach is introduced and computationally evaluated on some relevant classes of MIPs, and on the first author's master thesis [4] and the follow-up paper [6] where the optimal turbine location problem is studied.

2. Proximity search

The basic proximity search approach is sketched in Figure 1 below.

`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 θ

   until *an overall termination condition is reached*;

Figure 1 The basic Proximity Search algorithm (from [5])

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

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 Number of integer components in the LP relaxation solution at the proximity-search root node, for various values of the cutoff parameter θ (from [5]).

solution can be found by a fast ad-hoc heuristic, and the 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 θ is defined and a new constraint is added that is valid for all solutions improving over $\tilde{x}$.

At Step 3, the Hamming distance between x and $\tilde{x}$ is used, computed according to (5). Step 4 invokes the black-box MIP solver to hopefully find a new incumbent x^* with $f(x^*) \leq f(\tilde{x}) - \theta$. 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 θ can typically be fulfilled with just local adjustments. This is instrumental for the success of the 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 increased number of integer components in the solution of the convex relaxation is illustrated in Table 1 (from [5]) 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 θ, the effect on the LP-solution time is striking, and the number of integer components increases dramatically—from 682 ($\theta = 121$) to 1,622 ($\theta = 10$) and 2,181 ($\theta = 1$).

If no new solution x^* is found at Step 4 (possibly because the MIP solver was aborted before convergence), one proceeds directly to Step 6 where tolerance θ is reduced. Of course, if the MIP solver proved infeasibility for the given θ, one has 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 Δ, and/or by modifying the cutoff tolerance θ.

3. Proximity search implementations

In this section we sketch three implementations of the basic proximity search method, as described in [5].

All three implementations start with a given solution $\tilde{x}$, replace the original objective function $f(x)$ by a proximity one $\Delta(x, \cdot)$, and use a cutoff constraint to force the detection of improved solutions.

In the first implementation "without recentering" (see Subsection 3.1), $\Delta(x, \cdot)$ remains centered on the very first solution $\tilde{x}$, and the cutoff constraint is modified on the fly inside the MIP solver.

In the second implementation "with recentering" (Subsection 3.2), the MIP solver is aborted as soon as the first improved solution x' (say) is found, and is reapplied from scratch after replacing $\tilde{x}$ by x' both in the objective $\Delta(x, \cdot)$ and in the cutoff constraint.

The third implementation "with an incumbent" (Subsection 3.3) is a variant of the second one where the cutoff constraint is imposed in a soft way that makes the choice $x = \tilde{x}$ feasible—though highly penalized. In this way the current solution $\tilde{x}$ can be used to initialize the modified-MIP incumbent.

3.1. Proximity search without recentering

In this version, one assumes 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 first internally recorded in a user's data structure. Then, a new global cut $f(x) \leq f(\hat{x}) - \theta$ is added to the model so as make $\hat{x}$ infeasible—thus preventing the solver to update its own incumbent.

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" used 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 one never updates the MIP incumbent explicitly, propagation and variable-fixing schemes—as well as refinement heuristic—are never applied.

3.2. 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.

A different implementation that uses the MIP solver as a black box (with no callbacks) is outlined next, that just restarts it as soon as a new $\tilde{x}$ if found.

In the new implementation, called "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, one invokes the MIP solver as a black box, in its default mode and without any callback, and aborts 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 one can replace $\tilde{x}$ with the new solution and repeat (without changing θ) 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 θ-optimality of the incumbent $\tilde{x}$ or hits the time limit.

3.3. Proximity search with an incumbent

Both implementations above prevent the MIP solver to update its internal incumbent, so powerful refinement heuristics such as RINS [2] are never activated. To avoid this drawback, the following simple variant of the "proximity search with recentering" can be used. The cutoff constraint (4) is just replaced with its "soft version"

$$f(x) \leq f(\tilde{x}) - \theta + z \tag{6}$$

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

$$\Delta(x, \tilde{x}) + Mz \tag{7}$$

where M is a large positive value compared to the feasible values of Δ. 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 θ-improving solution has been found.

4. Application: optimal turbine location

Green energy became a topic of great interest in the last years. Indeed, every year the de-

mand for energy is increasing and the old resources, fossil fuels in particular, are becoming rare and pollutant. As a result, many countries have ambitious plans regarding green energy production, and lots of money and energies are spent in wind energy research.

The way a turbine produces power is essentially by transferring kinetic energy from the wind to the blades, which causes a decrease of the wind speed immediately behind the rotor—if another turbine is placed downstream, a reduction in the incoming wind speed—and hence a reduction in power production—is experienced. This phenomenon is known as "turbine wake". It is estimated in that in large offshore wind farms, the average power loss due to turbine wakes is around 10-20% of the total energy production. It is then obvious that power production can increase significantly if the farm layout is designed so as to reduce the effect of turbine wake as much as possible.

4.1. Choosing the MIP model

A basic MIP model from the literature is first outlined, that focuses on turbine proximity constraints and on the wake effect; see, e.g., [3]. We consider the following constraints:

a)a minimum and maximum number of turbines that can be built is given;
b)there should be a minimal separation distance of between two turbines to ensure that the blades do not physically clash (turbine proximity constraints);
c)if two turbines are installed, their interference will cause a loss in the power production that depends on their relative position and on wind conditions.

Let V denote the set of possible positions for a turbine, called "sites" in what follows, and let

- I_{ij} be the interference (loss of power) experienced by site j when a turbine is installed at site i, with $I_{jj} = 0$ for all $j \in V$;
- P_i be the power that a turbine would produce if built (alone) at site i;
- N_{MIN} and N_{MAX} be the minimum and maximum number of turbines that can be built, respectively;
- D_{MIN} be the minimum distance between two turbines;
- $dist(i, j)$ be the symmetric distance between sites i and j.

In addition, let $G_I = (V, E_I)$ denote the incompatibility graph with

$$E_I = \{[i, j] : i, j \in V,\, dist(i, j) < D_{MIN},\, j > i\}$$

and let $n := |V|$ denote the total number of sites.

In the model, two sets of binary variables are defined for each $i, j \in V$:

$$x_i = \begin{cases} 1 & \text{if a turbine is built at site } i \in V; \\ 0 & \text{otherwise} \end{cases}$$

$$z_{ij} = \begin{cases} 1 & \text{if two turbines are built at both sites } i \in V \text{ and } j \in V; \\ 0 & \text{otherwise} \end{cases}$$

The model then reads:

$$\max z = \sum_{i \in V} P_i x_i - \sum_{i \in V} \sum_{j \in V} I_{ij} z_{ij} \tag{8}$$

$$\text{s.t.} \quad N_{MIN} \leq \sum_{i \in V} x_i \leq N_{MAX} \tag{9}$$

$$x_i + x_j \leq 1 \qquad \forall [i,j] \in E_I \tag{10}$$

$$x_i + x_j - 1 \leq z_{ij} \qquad \forall i,j \in V, j > i \tag{11}$$

$$x_i \in \{0,1\} \qquad \forall i \in V \tag{12}$$

$$z_{ij} \in \{0,1\} \qquad \forall i,j \in V \tag{13}$$

Objective function (8) maximizes the total power production by taking interference losses I_{ij} into account. Constraints (11) force $z_{ij} = 1$ whenever $x_i = x_j = 1$; because of the objective function, this is in fact equivalent to setting $z_{ij} = x_i x_j$. Constraints (10) model pairwise site incompatibility, and can be strengthened to their clique counterpart

$$\textstyle\sum_{h \in Q} x_h \leq 1 \qquad \forall Q \in \mathcal{Q} \tag{14}$$

where $\mathcal{Q}$ is a family of maximal cliques of G_I, such that every edge in E_I is contained in at least one member of $\mathcal{Q}$.

The definition of the turbine power vector (P_i) and of interference matrix (I_{ij}) depends on the wind scenario considered, that greatly varies in time. Using statistical data, one can in fact collect a large number K of wind scenarios k, each associated with a pair (P^k, I^k) and with a probability π_k. Using that data, one can write a straightforward Stochastic Programming variant of the previous model where only the objective function needs to be modified into

$$z = \sum_{k=1}^{K} \pi_k \, (\sum_{i \in V} P_i^k x_i - \sum_{i \in V} \sum_{j \in V} I_{ij}^k z_{ij}) \tag{15}$$

while all constraints stay unchanged as they only involve "first-stage" variables x and z. It is therefore sufficient to define

$$P_i := \sum_{k=1}^{K} \pi_k P_i^k \qquad \forall i \in V \tag{16}$$

$$I_{ij} := \sum_{k=1}^{K} \pi_k I_{ij}^k \qquad \forall i,j \in V \tag{17}$$

to obtain the same model (8)–(13) as before.

Though alternative proposals are available in the literature (see, e.g., [1]), we decided to stick to the model above for two main reasons: (i) the model is quite standard and well understood by practitioners, (ii) as already mentioned, a suitable definition of the input data allows one to easily take different wind scenarios into account, whereas more sophisticated models would lead to really huge stochastic programming variants.

While (8)–(13) turns out to be a reasonable model when just a few sites have to be considered (say $n \approx 100$), it becomes hopeless when $n \geq 1,000$ because of the huge number of

variables and constraints involved, that grows quadratically with n. Therefore, when facing instances with several thousand sites an alternative (possibly weaker) model is required, where interference can be handled by a number of variables and constraints that grows just linearly with n. The model below is a compact reformulation of model (8)–(13) that follows a recipe of Glover [9] which is widely used, e.g., in the Quadratic Assignment Problem [7]. The original objective function (to be maximized)

$$\sum_{i \in V} P_i x_i - \sum_{i \in V} (\sum_{j \in V} I_{ij} x_j)\, x_i \tag{18}$$

is restated as

$$\sum_{i \in V} (P_i x_i - w_i) \tag{19}$$

where

$$w_i := (\sum_{j \in V} I_{ij} x_j)\, x_i = \begin{cases} \sum_{j \in V} I_{ij} x_j & \text{if } x_i = 1; \\ 0 & \text{if } x_i = 0. \end{cases}$$

denotes the total interference caused by site i. Our compact model then reads

$$\max z = \sum_{i \in V} (P_i x_i - w_i) \tag{20}$$

$$\text{s.t.} \quad N_{MIN} \leq \sum_{i \in V} x_i \leq N_{MAX} \tag{21}$$

$$x_i + x_j \leq 1 \qquad \forall [i,j] \in E_I \tag{22}$$

$$\sum_{j \in V} I_{ij} x_j \leq w_i + M_i (1 - x_i) \qquad \forall i \in V \tag{23}$$

$$x_i \in \{0,1\} \qquad \forall i \in V \tag{24}$$

$$w_i \geq 0 \qquad \forall i \in V \tag{25}$$

where the big-M term $M_i >> 0$ is used to deactivate constraint (23) in case $x_i = 0$. In our implementation, we set

$$M_i = \sum_{\substack{j \in V \\ [i,j] \notin E_I}} I_{ij}$$

and did not try to reduce it even further. Indeed, modern MIP solvers are able to strengthen the input constraints in their preprocessing phase, so the big-M values provided in the model are automatically reduced by the solver by taking the whole model into account, thus easing the modeling task.

As to (22), we decided not to improve them to their clique form (14), for two main reasons: (i) the number of cliques can be exceedingly large, and (ii) a family of cliques is automatically generated during preprocessing by the MIP solver, which can control their number and can handle them very efficiently. As we have no additional information about which cliques should be generated explicitly, we preferred to leave clique management to the solver itself.

4.2. Designing a proximity search heuristic

We now address how to improve a given feasible solution $(\tilde{x}, \tilde{w})$ by exploiting MIP model (20)–(25). One standard option would be to just use $(\tilde{x}, \tilde{w})$ to initialize the incumbent solution

of the MIP solver, and to run it in its default mode. However, it is common experience that this strategy is unlikely to produce improved solution within acceptable computing times, the main so if the underlying MIP model is very large and the formulation is weak—as it happens in out context. So, we preferred to address a different use of the MIP solver, to be applied to "search a neighborhood" of $(\tilde{x}, \tilde{w})$. In particular we focused on the proximity search strategy described in the previous sections, that seems particularly suited for models involving big-M constraints.

The approach can be cast into the so-called *MIP-and-refine* framework recently investigated in [8], and works as shown in Figure 2. The approach makes also use of very fast (1- and 2-opt) heuristics; see [4, 6] for details.

Step 1. apply ad-hoc heuristics (iterated 1-opt) to get a first incumbent $\tilde{x}$;
Step 2. apply quick ad-hoc refinement heuristics (few iterations of iterated 1- and 2-opt) to possibly improve $\tilde{x}$;
Step 3. if $n > \overline{N}$, randomly remove points $i \in V$ with $\tilde{x}_i = 0$ so as to reduce the number of candidate sites (in our tests, $\overline{N} = 2,000$ was used);
Step 4. build a MIP model for the resulting subproblem and apply proximity search to refine $\tilde{x}$ until the very first improved solution is found (or time limit is reached);
Step 5. if the time limit has not been reached, repeat from Step 2.

Figure 2 Our overall heuristic framework

Two different MIP models are used to feed the proximity-search heuristic at Step 4. During the first part of the computation, we use a simplified MIP model obtained from (20)–(25) by removing all interference constraints (23), thus obtaining a much easier relaxation. A short time limit (60 sec.s, in our tests) is imposed for each call of proximity search when this simplified model is solved. In this way we aggressively drive the solution $\tilde{x}$ to increase the number of built turbines, without being bothered by interference considerations and only taking pairwise incompatibility (22) into account. This approach quickly finds better and better solutions (even in terms of the true profit), until either (i) no additional turbine can be built, or (ii) the addition of new turbines does in fact reduce the true profit associated to the new solution. In this situation we switch to the complete model (20)–(25) with all interference constraints, which is used in all next executions of Step 4. Note that the simplified model is only used at Step 4, while all other steps of the procedure always use the true objective function that takes interference into full account.

4.3. Computational experiments

The following alternative solution approaches were implemented in C language, some of which using the commercial MIP solver IBM ILOG Cplex 12.5.1 [11]; because of the big-M's involved in the models, all Cplex's codes use zero as integrality tolerance (`CPX_PARAM_EPINT` = 0.0).

a)`proxy`: the MIP-and-refine heuristic, as outlined in the previous section, using Cplex with the following aggressive parameter tuning: all cuts deactivated, `CPX_PARAM_RINSHEUR` =

1, `CPX_PARAM_POLISHAFTERTIME` = 0.0, `CPX_PARAM_INTSOLLIM` = 2;

b) `cpx_def`: the application of IBM ILOG Cplex 12.5.1 in its default setting, starting from the same heuristic solution $\tilde{x}$ found by `proxy` after the first execution of Step 2 of Figure 2;

c) `cpx_heu`: same as `cpx_def`, with the following internal tuning intended to improve Cplex's heuristic performance: all cuts deactivated, `CPX_PARAM_RINSHEUR` = 100, `CPX_PARAM_POLISHAFTERTIME` = 20% of the total time limit;

d) `loc_sea`: a simple local-search procedure not based on any MIP solver, that just loops on Steps 2 of Figure 2 and randomly removes installed turbines from the current best solution after 10,000 iterations without improvement of the incumbent.

In our view, `loc_sea` is representative of a clever but not oversophisticated metaheuristic, as typically implemented by practitioners, while `cpx_def` and `cpx_heu` represent a standard way of exploiting a MIP model once a good feasible solution is known.

Our testbed refers to an offshore $3,000 \times 3,000$ (m) square with $D_{MIN} = 400$ (m) minimum turbine separation, with no limit on the number of turbines to be built (i.e., $N_{MIN} = 0$ and $N_{MAX} = +\infty$). Turbines are all of Siemens SWT-2.3-93 type (diameter 93m), producing a power of 0.0 MW for wind speed up to 3 m/s, of 2.3 MW for wind speed greater than or equal to 16 m/s, and intermediate values for winds in range 3-16 m/s according to a nonlinear function [14]. Pairwise interference (in MW) was computed using Jensen's model [12], by averaging 250,000+ real-world wind samplings grouped into about 500 macro-scenarios. A pairwise average interference of 0.01 MW or less is treated as zero. The reader is referred to [4] for details.

Five classes of medium-to-large problems with n ranging from 1,000 to 20,000 have been generated. For each class, 10 instances have been considered by generating n uniformly random points in the $3,000 \times 3,000$ square. In what follows, reported computing times are in CPU sec.s of an Intel Xeon E3-1220 V2 quad-core PC with 16GB of RAM.

Computational results on our instances are given in Table 4.3, where each entry refers to the performance of a given algorithm at a given time limit. In particular, the left part of the table reports, for each algorithm and time limit, the *number of wins*, i.e, the number of instances for which a certain algorithm produced the best solution at the given time limit (ties allowed).

According to the table, `proxy` outperforms all competitors by a large amount for medium to large instances. As expected, `cpx_heu` performs better for instances with $n = 1,000$ as it is allowed to explore a large number of enumeration nodes for the original model and objective function. Note that `loc_sea` has a good performance for short time limits and/or for large instances, thus confirming its effectiveness, whereas `cpx_heu` is significantly better than `loc_sea` only for small instances and large time limits.

A different performance measure is given in the right-hand side part of Table 4.3, where each entry gives the *average optimality ratio*, i.e., the average value of the ratio between the solution produced by an algorithm (on a given instance at a given time limit) and the best solution known for that instance—the closer to one the better. It should be observed that

		number of wins				optimality ratio			
n	Time limit (s)	`proxy`	`cpx_def`	`cpx_heu`	`loc_sea`	`proxy`	`cpx_def`	`cpx_heu`	`loc_sea`
1,000	60	6	1	3	0	0.994	0.983	0.987	0.916
	300	4	2	4	0	0.997	0.991	0.998	0.922
	600	7	3	7	0	0.997	0.992	0.997	0.932
	900	5	2	3	0	0.998	0.993	0.996	0.935
	1,200	5	1	5	0	0.998	0.992	0.997	0.939
	1,800	5	1	4	0	0.998	0.992	0.996	0.942
	3,600	4	2	5	0	0.998	0.995	0.997	0.943
5,000	60	9	6	6	5	0.909	0.901	0.901	0.904
	300	10	0	0	0	0.992	0.908	0.908	0.925
	600	10	0	10	0	0.994	0.908	0.994	0.935
	900	10	0	0	0	0.994	0.908	0.908	0.936
	1,200	10	0	0	0	0.994	0.908	0.925	0.939
	1,800	9	0	1	0	0.996	0.908	0.971	0.946
	3,600	5	0	5	0	0.996	0.932	0.994	0.948
10,000	60	9	9	8	10	0.914	0.913	0.914	0.914
	300	10	2	2	2	0.967	0.927	0.927	0.936
	600	10	0	10	0	0.998	0.928	0.998	0.944
	900	10	0	0	0	1.000	0.928	0.928	0.948
	1,200	10	0	0	0	1.000	0.928	0.928	0.951
	1,800	10	0	0	0	1.000	0.928	0.928	0.957
	3,600	9	0	0	1	1.000	0.928	0.928	0.964
15,000	60	9	10	9	9	0.909	0.912	0.911	0.909
	300	10	8	7	8	0.943	0.937	0.935	0.937
	600	10	0	10	0	0.992	0.939	0.992	0.942
	900	10	0	0	0	1.000	0.939	0.939	0.956
	1,200	9	0	0	1	1.000	0.939	0.939	0.959
	1,800	9	0	0	1	1.000	0.939	0.939	0.965
	3,600	9	0	0	1	1.000	0.939	0.939	0.972
20,000	60	9	9	9	10	0.901	0.902	0.901	0.902
	300	10	8	10	10	0.933	0.933	0.933	0.933
	600	9	0	9	1	0.956	0.935	0.956	0.941
	900	10	0	0	0	0.978	0.935	0.935	0.945
	1,200	10	0	0	0	0.991	0.935	0.935	0.950
	1,800	10	0	0	0	0.999	0.935	0.935	0.963
	3,600	9	0	0	0	1.000	0.935	0.935	0.971
ALL	60	42	35	35	34	0.925	0.922	0.922	0.909
	300	44	20	23	20	0.966	0.939	0.940	0.930
	600	46	3	46	1	0.987	0.941	0.987	0.938
	900	45	2	3	0	0.994	0.941	0.941	0.944
	1,200	44	1	5	1	0.997	0.940	0.945	0.947
	1,800	43	1	5	1	0.999	0.940	0.954	0.955
	3,600	36	2	10	2	0.999	0.946	0.959	0.959

Table 2 Number of times each algorithm finds the best solution within the time limit (wins), and optimality ratio with respect to the best known solution—the larger the better (from [6]).

an improvement of just 1% has a very significant economical impact due to the very large profits involved in the wind farm context. The results show that `proxy` is always able to produce solutions that are quite close to the best one. As before, `loc_sea` is competitive for large instances when a very small computing time is allowed, whereas `cpx_def` and `cpx_heu` exhibit a good performance only for small instances, and are dominated even by `loc_sea` for larger ones.

Acknowledgements

This research was supported by the University of Padova (Progetto di Ateneo "Exploiting randomness in Mixed Integer Linear Programming"), and by MiUR, Italy (PRIN project

"Mixed-Integer Nonlinear Optimization: Approaches and Applications").

References

[1] R. Archer, G. Nates, S. Donovan, H. Waterer: Wind turbine interference in a wind farm layout optimization mixed integer linear programming model. Wind Engineering **35, no.2**, 165–178 (2011)

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

[3] S. Donovan: Wind farm optimization. In: Proceedings of the 40th Annual ORSNZ Conference, pp. 196–205 (2005)

[4] Martina Fischetti: Mixed-integer models and algorithms for wind farm layout optimization. Master's thesis, University of Padova and Aalborg (2014)

[5] Matteo Fischetti, M. Monaci: Proximity search for 0-1 mixed-integer convex programming. Tech. rep., DEI, University of Padova (submitted) (2012)

[6] Martina Fischetti, M. Monaci: Proximity search heuristics for wind farm optimal layout. Tech. rep., DEI, University of Padova (submitted) (2014)

[7] Matteo Fischetti, M. Monaci, D. Salvagnin: Three ideas for the quadratic assignment problem. Operations Research **60**(4), 954–964 (2012)

[8] Matteo Fischetti, G. Sartor, A. Zanette: A MIP-and-refine matheuristic for smart grid energy management. International Transactions in Operational Research 1–11 (2013).

[9] F. Glover: Improved linear integer programming formulations of nonlinear integer problems. Management Science **22**, 455–460 (1975)

[10] F. Glover: Tabu search: A tutorial. Interfaces **20**(4), 74–94 (1990)

[11] IBM ILOG CPLEX: Optimization Studio (2013). `http://www.cplex.com`

[12] N. Jensen: A note on wind generator interaction. Tech. rep., Technical Report Riso-M-2411(EN), Riso National Laboratory, Roskilde, Denmark (1983)

[13] N. Mladenovic, P. Hansen: Variable neighborhood search. Computers & OR **24**(11), 1097–1100 (1997)

[14] Siemens AG: SWT-2.3-93 Turbine, Technical Specifications. `http://www.energy.siemens.com`