Metaheuristics for the Minimum Gap Graph Partitioning Problem

Maurizio Bruglieri

Dipartimento di Design, Politecnico di Milano

Roberto Cordone*

Dipartimento di Informatica, Università degli Studi di Milano

Abstract

The *Minimum Gap Graph Partitioning Problem* (*MGGPP*) consists in partitioning a vertex-weighted undirected graph into a given number of connected subgraphs with the minimum difference between the largest and the smallest weight in each subgraph. We propose a two-level Tabu Search algorithm and an Adaptive Large Neighborhood Search algorithm to solve the *MGGPP* in reasonable time on instances with up to about 23 000 vertices. The quality of the heuristic solutions is assessed comparing them with the solutions of a polynomially solvable combinatorial relaxation.

Keywords: Graph partitioning, Tabu Search, Adaptive Large Neighborhood Search

1. Introduction

Let $G = (V, E)$ be an undirected graph, with $|V| = n$ vertices and $|E| = m$ edges, let $p \in \mathbb{N}$ with $1 < p < n$ and let w be a *weight vector* which associates a weight $w_v \in \mathbb{N}$ with each vertex $v \in V$. For each vertex subset $U \subseteq V$, we denote by $m_U = \min_{u \in U} w_u$ and $M_U = \max_{u \in U} w_u$ the minimum and maximum value of the weights in U, respectively. We denote by *gap* the difference $\gamma_U =$

*Corresponding author

Email addresses: maurizio.bruglieri@polimi.it (Maurizio Bruglieri), roberto.cordone@unimi.it (Roberto Cordone)

$M_U - m_U$, that corresponds to the maximum difference between the weights of the vertices in U.

The *Minimum Gap Graph Partitioning Problem* (*MGGPP*) consists in finding a partition of G into p vertex-disjoint connected subgraphs $G_r = (V_r, E_r)$, with $r = 1, \ldots, p$. Here, we consider the *nondegenerate* problem (*MGGPPnd*), in which all subgraphs must have at least two vertices, and we focus on the *min-sum* version that minimizes the sum of all gaps, i.e., $\sum_{r=1}^{p} \gamma_{V_r}$.

The *MGGPP* is motivated by applications in different fields. First, the standard approach to manage large Water Distribution Networks is to sectorize them into subnetworks called District Metered Areas (*DMA*s) (Paola et al., 2014). This allows to localize leakages more accurately, by monitoring the input and the output discharges for each district. Moreover, it also achieves a better management of pressure through valves and turbines that produce energy, reducing the amount of existing losses and limiting the occurrence of new damages. The optimal design of *DMA*s takes into account, among other objectives, the minimization of the difference between the required heads within the *DMA*s. The purpose is to establish a unique target pressure value in each *DMA*, and consequently, to achieve an efficient pressure regulation also in networks with strong variations in ground elevation (Gomes et al., 2015). If the network is represented as a graph where the edges correspond to pipes, the vertices to their intersections and the weight of a vertex to the ground elevation of an intersection, the *MGGPP* models the search for an optimal sectorization.

A second possible application concerns the levelling of farmlands, that is an important foundation of modern ground irrigation systems, as it improves the uniformity of irrigation and soil salt distribution, thus controlling weeds, saving water and increasing yield (Xiao et al., 2015). In general, it might be impractical or too expensive to flatten a sloping land as a whole. In this case, it is necessary to divide the land into parcels, and build a flat terrace on each parcel by suitable earthworks. Choosing parcels with a small ground elevation difference helps to reduce the amount of ground to be moved, and therefore the cost associated with the following earthworks. Height is not the only significant property in land partitioning: for example, the problem of partitioning an agricultural field into the minimum number of rectangular zones with an upper bound on the variance of a suitable soil property is proposed by Albornoz et al. (2020), in a multiobjective and stochastic framework. The *MGGPP* allows to model the division of a land into parcels with a limited difference in height, or any other land attribute: the vertices of the graph correspond to sampled locations in the land, the weights to their heights, while the edges link adjacent locations.

In this paper we propose a two-level Tabu Search and an Adaptive Large Neighborhood Search (*ALNS*) metaheuristic to solve the *MGGPP* in reasonable time also on real-world instances with up to 23 000 vertices. The former approach is characterized by the clustering of vertices into macro-vertices that are handled as single objects during the search: many small macro-vertices or few large macro-vertices according to the granularity of the clustering. This has been made for two reasons. First, moving a single vertex from a component to another one is often unfeasible, because the resulting components of the partition can be disconnected. Second, moving a single vertex may yield a solution with the same total gap, because the gap of a component changes only if the vertex moved has the minimum or the maximum weight. On the other hand, the *ALNS* approach is based on the alternated application of removal and insertion heuristics to destroy and rebuild part of the solution so as to avoid the limitations of reassigning single vertices. Finally, in order to evaluate the quality of the solutions found, we compute a combinatorial bound by relaxing the connectivity constraint on the subgraphs and applying the polynomial procedure presented by Bruglieri et al. (2018).

The paper is organized as follows: Section 2 surveys the related literature; Section 3 introduces a Mixed Integer Linear Programming (*MILP*) formulation of the *MGGPP*; Section 4 describes the Tabu Search metaheuristic and its two-level extension *2L-TS*; Section 5 describes the *ALNS* algorithm; Section 6 shows the numerical results obtained on instances of different size and topology. Finally, we draw some conclusions in Section 7.

2. Literature

The *MGGPP* has been introduced by Bruglieri and Cordone (2016), where it is proved to be $\mathcal{NP}$-hard and a couple of its special cases are investigated. Other special cases have been characterized by Bruglieri et al. (2018), while a basic Tabu Search heuristic procedure (without the multi-level extension) is tested on small random instances in Bruglieri et al. (2017).

The *MGGPP* clearly falls within the large field of graph partitioning problems (Bader et al., 2013; Bichot and Siarry, 2013), but it is distinguished by a rather uncommon objective function. Most graph partitioning problems, in fact, optimize the cost of the edges linking different subgraphs of the partition (see for example the classical *Minimum k-Cut Problem* (Goldschmidt and Hochbaum, 1988; Krauthgamer et al., 2009)), whereas the *MGGPP* does not consider any edge cost. The graph partitioning problems whose definition includes vertex weights usually take into account the *total* weight of the vertices in each subgraph, and require it to be as uniform as possible across different subgraphs. This is obtained either by minimizing the difference between the total weights of different subgraphs (Chlebikova, 1996) or by imposing bounds on each of them (Ito et al., 2012, 2007). For example, Lucertini et al. (1993) consider the problem of partitioning a vertex-weighted path into subpaths, such that the total weight of every subpath lies between two given values.

Some similarity with the *MGGPP* can be found in the sectorization problem more recently proposed by Tang et al. (2014), where a graph must be divided into districts and the total weight of each district should differ as little as possible from a given reference value. Lari et al. (2016) address the problem of partitioning a graph into components each including a given special vertex (*center*), so that the sum of the assignment costs of the other vertices to the centers is minimized. This work also considers uniform partitioning problems, whose goal is to balance as much as possible the total cost or the total weight of the components. Finally, the *Minimum-Diameter Partitioning Problem* (*MDPP*) introduced by Hubert (1974) partitions a set of objects into a given number of clusters, such that the largest dissimilarity between objects in the same cluster is minimized. If the objects have weights and the dissimilarity between two objects is defined as the difference of their weights, one obtains the *MGGPP* on a complete graph. The two problems, therefore, share a common case. This case can be solved in polynomial time, as discussed by Hochbaum (2019) together with other clustering problems dealing with different objective functions.

3. Mixed Integer Linear Programming formulation

In this section we introduce a Mixed Integer Linear Programming (*MILP*) formulation of the *MGGPP*, where a feasible solution is represented as a spanning tree on an auxiliary directed graph $\tilde{G} = (V \cup \{r\}, A)$, being r a super-root node and A a set of arcs including both arcs (i, j) and (j, i) for each edge $[i, j] \in E$, and an arc (i, r) for each vertex $i \in V$. Each subtree appended to r identifies a connected subgraph of the solution (i.e., a component of the partition of G), and the predecessor of r is the minimum weight node in each subtree. Therefore, the resulting model is a multi-commodity flow formulation based on three families of variables: binary flow variables y_{ijk}, for all $(i, j) \in A$ and $k \in V$, equal to 1 if the flow of commodity k passes along arc (i, j), 0 otherwise; binary arc variables x_{ij}, for all $(i, j) \in A$, equal to 1 if arc (i, j) belongs to the spanning tree, 0 otherwise; real variables γ_i, for all $i \in V$, equal to the gap of the subgraph linked to r through node i, 0 if i is not linked to r.

$$\min \sum_{i \in V} \gamma_i \tag{1}$$

$$\sum_{(i,j) \in \delta^+(i)} y_{ijk} = \sum_{(i,j) \in \delta^-(i)} y_{jik} \quad \forall i \in V, k \in V : i \neq k \tag{2}$$

$$\sum_{(i,j) \in \delta^+(i)} y_{ijk} = 1 \quad \forall i \in V, k \in V : i = k \tag{3}$$

$$\sum_{i \in V} y_{irk} = 1 \quad \forall k \in V \tag{4}$$

$$\sum_{(i,j) \in \delta^+(i)} x_{ij} = 1 \quad \forall i \in V \tag{5}$$

$$x_{ij} + x_{ji} \leq 1 \quad \forall (i,j) \in A \tag{6}$$

$$y_{ijk} \leq x_{ij} \quad (i,j) \in A, k \in V \tag{7}$$

$$y_{irk} = 0 \quad \forall i \in V, k \in V : w_k < w_i \tag{8}$$

$$w_k y_{irk} - w_i x_{ir} \leq \gamma_i \quad \forall i \in V, k \in V \tag{9}$$

$$\sum_{(j,i) \in \delta^-(i)} x_{ji} \geq x_{ir} \quad \forall i \in V \tag{10}$$

$$\sum_{i \in V} x_{ir} = p \tag{11}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{12}$$

$$y_{ijk} \in \{0,1\} \quad \forall (i,j) \in A, k \in V \tag{13}$$

where $\delta^+(i)$ and $\delta^-(i)$ denote the forward and the backward star of vertex i, respectively.

The objective function (1), to be minimized, is the sum of the gaps of each subgraph in the solution. Constraints (2) guarantee the flow conservation for each commodity k in each node $i \neq k$; (3) ensure that the total flow of each commodity k emitted by node k is 1; (4) impose that for each commodity k the super-root r can be reached from one and only one node $i \in V$. Constraints (5) guarantee that for each node $i \in V$ one and only one arc of its forward star is selected. Constraints (6) prevent the two-cycles. Consistency constraints (7) impose that no commodity k can flow through arc (i,j) if this is not in the solution. Constraints (8) guarantee that the last node that sends commodity k directly to the super-root r has weight lower than w_k, and thus is the minimum weight node of the subgraph. Constraints (9) enforce variables γ_i to model the gap of the subgraph linked to r through node i and (10) ensure that no subgraph in the solution is a singleton. Finally, constraint (11) guarantees that the solution is made up of p subgraphs and constraints (12) and (13) specify the nature of the variables.

4. Tabu Search

This procedure consists of two phases: the first builds an initial solution through a *construction procedure*, while the second improves it by *Tabu Search* (Glover, 1986). The construction procedure is inspired by Prim's algorithm for the minimum spanning tree problem (Prim, 1957) and it returns a spanning forest of p trees identifying a feasible solution for the *MGGPP*. The procedure is detailed in Algorithm 1. First, it builds a forest, extracting for p times an edge with the minimum weight difference, avoiding the edges adjacent to the previously selected ones. Then, adjacent edges are iteratively selected and appended to one of the p trees, so as to keep the minimum total gap and not to merge different trees, until the forest spans the whole graph.

Tabu Search is a well known metaheuristic approach introduced by Glover (1986) to enhance the performance of local search. Our implementation for the *MGGPP* exploits a very simple neighborhood, based on moving a single vertex from a subgraph of the current solution to another one. This neighborhood consists of at most $n(p-1)$ solutions. The feasibility requires three conditions: 1) the moved vertex must not be an *articulation vertex* for the original subgraph[1]; 2) the destination subgraph must be adjacent to the vertex; 3) the original subgraph must include at least three vertices. The feasible moves can be identified in $O(m)$ time for the whole neighborhood, thanks to the identification of the

[1] An articulation vertex is a vertex whose removal disconnects the graph it belongs to.

Algorithm 1 Constructive procedure

```
Order the edges $[i, j] \in E$ by non decreasing values of the gaps $|w_i - w_j|$;
Let $X := \emptyset$;
while $|X| \leq p$ do
    Let $[i, j] := extract\_first(E \setminus X)$;
    if $[i, j]$ is not adjacent to any vertex in $X$ then
        Let $X := X \cup \{[i, j]\}$;
    end if
end while
while (there are vertices uncovered by $X$) do
    for each edge $e$ in the cut generated by the vertices covered by $X$ do
        Compute the total gap obtained if $e$ were added to $X$;
    end for
    Select the edge $e^*$ that minimizes the total gap;
    Let $X := X \cup \{e^*\}$;
end while
```

articulation vertices and of the subgraphs adjacent to each vertex.

The *landscape* (Stadler, 1996) of the *MGGPP* is characterized by large *plateaus*, that is, subsets of feasible solutions with the same objective function value. This is challenging for any neighbourhood-based procedure, because it may hide the most promising search directions. Therefore, we consider four criteria that are optimized in a lexicographic way in order to discriminate the feasible solutions in the plateau and to drive the search in a less miopic direction. The first criterium is the original objective function f. The second and the third one allow a partial look-ahead, considering not only the impact of a single move, but also its combination with the next ones. Indeed, although a move may have no immediate effect on the objective function, the subsequent transfer of a second vertex from the same subgraph could allow a strong improvement. Consequently, the second criterium to evaluate the transfer of a vertex v from its current subgraph G_s to a different one is defined as the better of the following two values: (i) the gap of G_s after removing both v and the vertex with the maximum weight; (ii) the gap of G_s after removing both v and the vertex with the minimum weight. If v coincides with either the maximum

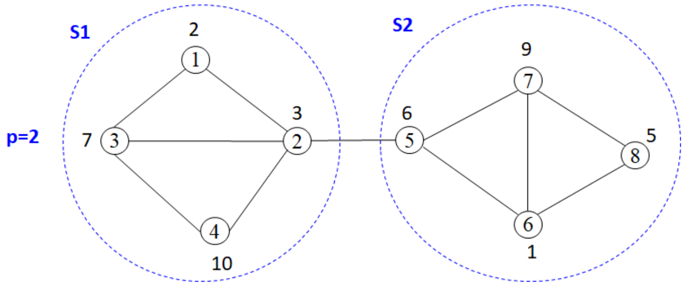

Figure 1: Example of the application of the lexicographic optimization to escape from plateaus

weight vertex or the minimum one, the previous measures consider the second maximum or minimum weight vertex. The third criterium does the same, but moves two vertices in addition to v, corresponding to (i) the two maximum weights, (ii) the two minimum ones, or (iii) the maximum and the minimum one. Finally, the fourth criterium is the index of the most recent iteration in which the vertex v has been transferred in the past. This aims to favour the oldest moves upon the most recent ones. See an example in Figure 1: the two only feasible moves transfer, respectively, vertex 2 from subgraph S1 to S2 and vertex 5 from subgraph S2 to S1. Both moves yield solutions with a total gap equal to 16. Applying the second criterium, the gap of S1 after moving $v = 2$ and the maximum weight vertex (4) is 5, whereas its gap after moving $v = 2$ and the minimum weight vertex (1) is 3: the better of the two values is 3. On the other hand, the gap of S2 after moving $v = 5$ and the maximum weight vertex (7) is 4; its gap after moving $v = 5$ and the minimum weight vertex (6) is again 4. The lexicographic function therefore selects the first move. In this way, at the next iteration the best move consists in moving also vertex 1 to S2, thus leading to the optimal solution.

The Tabu Search mechanism adopted uses as an attribute the inclusion of a vertex v in a subgraph G_s for $s = 1, \dots, p$. A suitable matrix L stores the most recent iteration l_{vs} in which vertex v has left G_s. A move that transfers vertex v into G_s is tabu until iteration $l_{vs} + l_{in}$, where l_{in} is a parametric value known as *tabu tenure*. At the beginning, all elements of matrix L are set to a very large negative value ($l_{vs} = -\infty$), so that all moves are nontabu. In order to intensify or diversify the search, the tenure parameter changes in an adaptive way based on the value assumed by the objective in the previous iteration: l_{in} decreases by 1 if the objective has improved, increases by 1 otherwise, remaining inside a prescribed range $\{l_{\min}, ..., l_{\max}\}$. The rationale is to intensify the search in more promising regions of the solution space and to diversify it in less promising ones.

4.1. Two-level Tabu Search

In spite of the lexicographic approach and the Tabu Search mechanism, plateaus remain a challenging feature of the problem. A well-known approach to deal with problems in which a graph must be partitioned into connected components is to apply a coarsening procedure to merge subsets of adjacent vertices into macro-vertices that can be moved as a whole. This is often done hierarchically, that is repeatedly applying the coarsening procedure to generate more and more aggregated graphs, that are given as inputs to a partitioning procedure (Walshaw, 2004). The solution obtained at each coarsening level can be transformed back into a solution of the original problem by simply replacing each macro-vertex with the corresponding subset of vertices. In fact, the components obtained are connected subgraphs of macro-vertices and each macro-vertex induces a connected subgraph on the original graph G.

Algorithm 2 Two-level Tabu Search (2L-TS)

```
for $\tilde{n} = \lfloor n/2 \rfloor, \lfloor n/2 \rfloor - 1, \ldots, p+1$ do
    Apply Algorithm 1 and then the Tabu Search both with $p = \tilde{n}$;    ▷ Building phase
    Contract each subgraph into a macro-vertex to obtain an aggregated graph $\tilde{G}$;
    Apply the Tabu Search to $\tilde{G}$ allowing singletons and considering a range weight
        $[w_v^{\min}, w_v^{\max}]$ associated with each macro-vertex $v$;    ▷ Improving phase
    Map the solution obtained onto the original graph $G$;
    Apply the Tabu Search to this solution in $G$;    ▷ Refinement phase
end for
return The best solution obtained
```

As it is not easy to identify the correct level of aggregation depending on the features of the single instance, we decided to exploit the Tabu Search procedure itself as a coarsening procedure while limiting the approach to a single coarsening phase. In fact, the components of a good solution to the *MGGPP* tend to have small gaps, and this makes them promising candidates for the role of macro-vertices. This gives rise to the two-level Tabu Search (*2L-TS*) whose pseudocode is reported in Algorithm 2. The algorithm considers every feasible number $\tilde{n}$ of macro-vertices, starting from $\tilde{n} = \lfloor n/2 \rfloor$ macro-vertices made of two vertices each (except one macro-vertex of three vertices, if n is odd), down to $p+1$ macro-vertices. For each value of $\tilde{n}$, the algorithm goes through three phases. The *building phase* applies the Prim-like constructive and the Tabu Search procedure to the original graph, with the number of subgraphs set to $\tilde{n}$, instead of p. The solution of this phase is converted into an aggregated graph, contracting each subgraph into a macro-vertex. The *improving phase* applies the constructive and the Tabu Search procedure to the aggregated graph, and returns a partition into p components. The Tabu Search applied in this phase differs from the one described in the previous section for two aspects. First, it allows singleton components, because it works on macro-vertices that correspond by construction to subsets of at least two vertices. Second, each macro-vertex is associated with a range of weights $\left[w_v^{\min}, w_v^{\max}\right]$, instead of a single weight w_v, where $w_v^{\min}$ and $w_v^{\max}$ are the minimum and the maximum vertex weight in the subgraph corresponding to vertex v. The original Tabu Search can be trivially extended to handle both these aspects. Indeed, if a subgraph is made up by k macro-vertices $v_1, v_2, \ldots, v_k$, its gap is straightforwardly given by $\max_{i=1,\ldots,k} w_{v_i}^{max} - \min_{i=1,\ldots,k} w_{v_i}^{min}$. The solution of the improving phase corresponds to a solution on the original graph. The *refinement phase* applies the original Tabu Search procedure to this solution on the original graph, in a final attempt to further improve it.

5. Adaptive Large Neighborhood Search

The *ALNS* is a metaheuristic introduced by Røpke and Pisinger (2006). It has been successfully applied to several problems, as surveyed by Pisinger and Røpke (2010), mainly in the field of vehicle routing. The applications to other contexts are less numerous: see Cordeau et al. (2010) for the scheduling of technicians and tasks in a telecommunications company, Muller (2009) for the resource-constrained project scheduling problem and Muller et al. (2012) for the lot-sizing problem. To the best of our knowledge, this is the first application of *ALNS* to a graph partitioning problem.

ALNS is an extension of Large Neighborhood Search (*LNS*). The basic idea of *LNS*, introduced by Shaw (1998), is to search in large neighborhoods, which may contain more and potentially better solutions compared to small neighborhoods. The neighborhood of a solution is defined implicitly by a destroy method, i.e., a *removal heuristic*, and a repair method, i.e., an *insertion heuristic*. The former disrupts part of the current solution while the latter rebuilds the destroyed solution. In *ALNS*, several removal heuristics and insertion heuristics are employed. In each iteration, the removal/insertion heuristics to be applied to the current solution are selected according to their *scores*, which are adaptively adjusted based on the performance of these heuristics in the previous iterations: the heuristics that have performed better in the previous iterations have larger scores and are more likely to be chosen in the current one.

Algorithm 3 reports the pseudocode of our implementation of *ALNS*. First, we initialize probability P_{RH} for each removal heuristic $RH \in \mathcal{RH}$ and probability P_{IH} for each insertion heuristic $IH \in \mathcal{IH}$, using a uniform probability distribution (lines 1–2). The initial solution X_0 is computed with the Prim-like constructive heuristic (line 3). The algorithm runs as long as the maximum number of iterations or the maximum execution time have not been reached (line 7). We randomly select a removal heuristic and an insertion heuristic according to the current values of the probabilities (lines 8 and 9) and we apply them to the current solution X_{cur}, obtaining the new solution X_{new} (line 10).

To accept a solution, we use the same criterion defined in the Simulated Annealing algorithm. If the total gap of the new solution, $\gamma(X_{new})$, is smaller than that of the current one, $\gamma(X_{cur})$, then the new solution is accepted (lines 11–12). In the opposite case (lines 13–19), the new solution is accepted with a probability equal to $e^{(-(\gamma(X_{new})-\gamma(X_{cur}))/\gamma(X_0)\tau)}$, where τ denotes the current value of a *temperature* parameter that starts at a suitable value τ_0 (line 5) and decreases at each iteration through a *cooling rate* $h \in [0, 1]$ (line 23).

Algorithm 3 Pseudocode for the *ALNS* procedure

```
1: Set $P_{RH} := 1/|\mathcal{RH}|$ for each removal heuristic $RH \in \mathcal{RH}$;
2: Set $P_{IH} := 1/|\mathcal{IH}|$ for each insertion heuristic $IH \in \mathcal{IH}$;
3: Generate an initial solution $X^0$ by applying the Prim-like constructive heuristic;
4: Set $X_{best} := X_{cur} := X^0$;
5: Set $\tau := \tau^0$;
6: Set $N := 0$;
7: while the termination condition is not satisfied do
8:     Select a removal heuristic $RH \in \mathcal{RH}$ with probability $P_{RH}$;
9:     Select an insertion heuristic $IH \in \mathcal{IH}$ with probability $P_{IH}$;
10:    Let $X_{new}$ be the solution obtained applying $RH$ to $X_{cur}$ and $IH$ to the result;
11:    if $\gamma(X_{new}) < \gamma(X_{cur})$ then
12:        Set $X_{cur} := X_{new}$ ;
13:    else
14:        Set $\nu := e^{(-(\gamma(X_{new})-\gamma(X_{cur}))/\gamma(X^0)\tau)}$;
15:        Generate a random number $\epsilon \in [0, 1]$;
16:        if $\epsilon < \nu$ then
17:            Set $X_{cur} := X_{new}$ ;
18:        end if
19:    end if
20:    if $\gamma(X_{cur}) < \gamma(X_{best})$ then
21:        Set $X_{best} := X_{cur}$;
22:    end if
23:    Set $\tau := h\tau$;
24:    Update $P_{RH}$ and $P_{IH}$ using $AWAP$;
25:    Let $N := N + 1$;
26: end while
```

5.1. Removal heuristics

In our implementation of the *ALNS* for the *MGGPP*, we consider three families of removal heuristics:

1. *Random removal*: heuristic RH1 randomly eliminates a fraction q of the n vertices from the current solution. In order to ensure that the insertion procedure will be able to rebuild a feasible solution, we compute the connected components for each cluster from which vertices have been removed and keep only the component of maximum cardinality (and minimum gap, in case of ties). If all components of a cluster are singletons, we remove all the vertices. In general, therefore, the number of removed vertices is $\geq q\,n$.

2. *Worst cluster removal*: heuristic RH2 removes all the vertices of the cluster with the maximum gap. We also consider heuristic RH3, which randomly selects either the cluster with the second maximum gap or that with the third maximum gap, and removes all its vertices.

3. *Shaw removal*: this heuristic, introduced by Shaw (1998), removes vertices that are somewhat similar, on the basis of a specific relatedness indicator, since we expect it to be easier to create better solutions by reassigning similar vertices. Given two vertices i and j, we define their *relatedness* $R(i,j)$ as a convex combination of their weight difference, $|w_i - w_j|$, and of the number of edges, l_{ij}, of the minimum cardinality path between them:
$$R(i,j) = \omega \frac{|w_i - w_j|}{\Gamma} + (1-\omega)\frac{l_{ij}}{L}$$
where $\omega \in [0,1]$ tunes the relative weight of the two components, $\Gamma = \max_{i,j \in V} |w_i - w_j|$, and $L = \max_{i,j \in V} l_{ij}$. We obtain three heuristics of this kind that we call RH4, RH5 and RH6 considering the following values of weight, respectively: $\omega = 0$, $\omega = 0.5$, and $\omega = 1$.

We report the detailed pseudocode of the Shaw removal heuristic, based on that of Røpke and Pisinger (2006), in Algorithm 4. Starting from a random

Algorithm 4 Pseudocode for the Shaw removal heuristic

```
1: Let $\overline{v}$ be a vertex randomly selected from $V$;
2: Set $S := \{\overline{v}\}$;
3: while $|S| < q\,n$ do
4:     Let $v$ be a vertex randomly selected from $S$;
5:     Let $\alpha$ be an array containing all vertices of $V \setminus S$ and $n_\alpha = |V \setminus S|$;
6:     Sort $\alpha$ in such a way that $i < j \Rightarrow R(v, \alpha[i]) \leq R(v, \alpha[j])$;
7:     Choose a random number $y$ from the interval [0,1);
8:     Set $S := S \cup \{\alpha[\lfloor y^\rho n_\alpha \rfloor]\}$;
9: end while
10: Remove the vertices of $S$ from $X$;
```

vertex $\overline{v}$, the procedure removes a total number of $q\,n$ vertices from the current solution X. They are selected choosing one of the previously removed vertices at random, v, sorting the vertices that still belong to the solution by increasing relatedness to v and selecting one of them with a biased random extraction. The real number $\rho \geq 1$ tunes the randomness in the selection. In fact, $\rho = 1$ corresponds to a completely random removal algorithm, whereas very high values of ρ let y^ρ tend to 0, so that line 8 selects the first vertex of array α, i.e., the most related to v.

5.2. Insertion heuristics

We consider four insertion heuristics, IH1-IH4, to reinsert the removed vertices back into the current solution X. The heuristic IH1 applies the Prim-like procedure: for each removed vertex that can be feasibly added to one of the current subgraphs of X (that is, the adjacent ones), it computes the resulting total gap; then, it adds the best one until all the removed vertices are inserted. Heuristic IH2 is a randomized version of the same procedure: instead of choosing the best adjacent vertex, it randomly selects one of the k best vertices, where k is set to one third of the total number of adjacent vertices. Heuristic IH3 does the same, with a candidate list including two thirds of the adjacent removed vertices. Finally, IH4 adds an adjacent vertex at random.

5.3. Adaptive Weight Adjustment Procedure

The Adaptive Weight Adjustment Procedure (*AWAP*), used at line 24 of Algorithm 3, allows to adjust automatically the probabilities of using the removal and insertion heuristics on the basis of their success in the previous iterations. Its pseudocode is shown in Algorithm 5. A *score* s_H is associated with each heuristic $H \in RH \cup IH$, in order to measure how well the heuristic has performed recently. A counter θ_H tracks the number of applications of each heuristic. The whole search is divided into a number of *segments*, that is sequences of NS iterations of the *ALNS*. At the beginning of each segment, when the remainder of the division of iteration index N by NS is equal to zero (line 1), all scores and counters are initialized to zero. At each iteration, the counters of the removal and of the insertion heuristic applied are increased by 1 and the corresponding scores are increased by one of three possible values, depending on the quality of the solution achieved. If a new overall best solution is found (line 4), the two heuristics are rewarded with an increase of σ_1. If the solution found has not been visited before and it is accepted by the *ALNS* (line 7), then the heuristics are rewarded with an increase of σ_2 (line 7) or σ_3 (line 10), according to the fact that the new solution is better or worse than the previous one. To check whether a solution has never been visited before, we can memorize all the new solutions accepted in a hash table. Notice that, as we shall see in Section 6, the experimental results suggest to accept only improving solutions, so that the case of line 10 will never occur and the hash table is not actually required.

At the end of each segment, when $mod(N, NS) = NS - 1$, the probability of each heuristic is updated with a convex combination of the previous probability and the average score s_H/θ_H achieved applying the heuristic (see line 18). The *reaction factor*, r, controls how quickly the weight-adjustment algorithm reacts to changes in the effectiveness of the heuristics: if $r = 0$, the scores are not used and thus the probabilities do not change; if $r = 1$, only the scores obtained in the last segment determine the new probabilities. We introduce a slight modification in the original formula of Røpke and Pisinger (2006) in order to address the relative order of magnitude of probabilities and average scores: first, we normalize the average scores s_H/θ_H before combining them; second, we renormalize the probabilities after the update. In both cases, we proceed separately for the removal and for the insertion heuristics. This allows a much easier tuning of the scores.

Algorithm 5 Pseudocode for the *AWAP*

```
if $mod(N, NS) = 0$ then
    Set $s_H := 0$, $\theta_H := 0$ for each $H \in \mathcal{RH} \cup \mathcal{IH}$;
end if
if applying $RH$ and $IH$ yields a new best known solution then
    Set $s_{RH} := s_{RH} + \sigma_1$;
    Set $s_{IH} := s_{IH} + \sigma_1$;
else if applying $RH$ and $IH$ yields a solution that has never been accepted and is better
  than the current one then
    Set $s_{RH} := s_{RH} + \sigma_2$;
    Set $s_{IH} := s_{IH} + \sigma_2$;
else if applying $RH$ and $IH$ yields a solution that has never been accepted, is not better
  than the current one, but has been accepted then
    Set $s_{RH} := s_{RH} + \sigma_3$;
    Set $s_{IH} := s_{IH} + \sigma_3$;
end if
Set $\theta_{RH} := \theta_{RH} + 1$;
Set $\theta_{IH} := \theta_{IH} + 1$;
if $mod(N, NS) = NS - 1$ then        ▷ Update the probabilities at the end of each segment
    for each $H \in \mathcal{RH} \cup \mathcal{IH}$ do
        Set $P_H := (1 - r)\, P_H + r\, Norm(\frac{s_H}{\theta_H})$;
    end for
    for each $H \in \mathcal{RH} \cup \mathcal{IH}$ do
        Set $P_H := Norm(P_H)$;
    end for
end if
```

6. Computational experiments

Our experimental campaign was performed coding all algorithms in C, compiling them with `gcc 7.3.1` and running them on an Intel Xeon E5-2620 2.1GHz server with 16GB of RAM.

We built instances with five different sizes (n ranging from 100 to 500 by steps of 100 vertices) and three different structures with respect to the density and topology: random graphs with $m = 2n(n-1)/3$ edges and $m = n(n-1)/3$ edges, and planar graphs obtained with a greedy triangulation of n points uniformly distributed at random in a square. In the following, we will denote these three classes as *dense*, *sparse* and *planar* instances. For each of them, three subclasses were obtained setting the number of subgraphs p equal to $\ln(n)$, $\sqrt{n}$ or $n/\ln(n)$ (rounded to the closest integer), in order to obtain solutions with few large subgraphs, balanced subgraphs, or many small subgraphs, respectively. The vertex weights are integer numbers uniformly distributed in $\{1, \ldots, n\}$. For each combination we generated 5 instances, thus obtaining a benchmark set of $5 \cdot 3 \cdot 3 \cdot 5 = 225$ instances of the *MGGPP*. This set was used to tune the parameters of the two algorithms and, when necessary, it will be denoted as the *tuning* set.

A second set of 225 instances with the same structure, but different random seeds, was generated to evaluate the performance of the algorithms after the tuning phase, in order to avoid the risk of overfitting. This will be denoted as the *evaluation* set. Finally, a third set of *real-world* instances was drawn from the literature. This was derived from three different sources. First, 28 instances were drawn from the 10th DIMACS challenge (Bader et al., 2014): they are connected unidirected graphs ranging from 62 to 22 963 vertices. As the original graphs are unweighted, we generated vertex weights in the same way used for our benchmarks (a uniform distribution in $\{1, ..., n\}$). Then, we considered 6 hydraulic networks introduced by Wang et al. (2014), that range from 272 to 12 527 vertices, whose weights represent elevations. Finally, we considered 3 agricultural networks. These represent grids of sampled locations on fields. They have 616, 741 and 862 vertices, respectively, and their weights also represent the elevations of the corresponding points. For each of these 37 graphs, we considered the usual three different values for the number of subgraphs p (that is $\ln(n)$, $\sqrt{n}$ or $n/\ln(n)$, rounded to the closest integer). The first and second column of Table 1 list the sources and the names of the 37 considered graphs. The following two columns contain the number of vertices n and edges m, and the last three the cardinality p of the partition required in each of the three derived instances. The third benchmark set, therefore, consists of 111 instances. All of the benchmark instances are available at `https://homes.di.unimi.it/cordone/research/research.html`.

6.1. Indicators of the solution quality

The quality of a heuristic solution is usually expressed by the percentage gap between its value z and the optimum z^*. However, the *MGGPP* is strongly $\mathcal{NP}$-hard (Bruglieri and Cordone, 2016), and Formulation (1)-(13) allows to compute the optimum with a general-purpose solver only for very small instances ($n \leq 20$) (Bruglieri et al., 2017). We therefore have to replace z^* with estimates. A lower bound $z_{LB} \leq z^*$ is computed neglecting the connectivity constraint and then solving the problem on the resulting complete graph with the polynomial procedure described by Bruglieri et al. (2018). An upper bound $z_{UB} \geq z^*$ is provided by the best solution found during the whole experimental campaign (i.e., including also the results found in the parameter tuning phases). Correspondingly, we measure the quality of solutions with two indicators: i) the gap $\delta_{UB}(z) = (z - z_{UB})/z_{UB}$ with respect to the upper bound z_{UB}; ii) the gap $\delta_{LB}(z) = (z - z_{LB})/z_{LB}$ with respect to the lower bound z_{LB}. The former underestimates the actual gap between z and z^*, while the latter overestimates it. Our experiments show that both estimates are very tight on the dense instances with few subgraphs[2]. Moving to sparser instances and larger values of p, the lower bound strongly degrades because the solution of the combinatorial relaxation mostly consists of disconnected subgraphs, while the upper bound becomes less robust as different parameter tunings often yield different results.

[2] Sometimes, the two bounds even coincide; notice that, however, in most of these cases the bounding procedure provides an unfeasible partition, though with the same total gap as the optimal solution.

		n	m		p	
Source	Instance			$\ln n$	$\sqrt{n}$	$n/\ln n$
Hydraulic	MOD	272	317	6	16	49
	d-town	407	459	6	20	68
	BIN	447	454	6	21	73
	wolf-initial-fig	1786	1995	7	42	239
	EXN	1893	2467	8	44	251
	3_BWSN_Network_2	12527	14831	9	112	1328
Agriculture	cellplot3	616	1172	6	25	96
	cellplot2	741	1422	7	27	112
	cellplot1	862	1664	7	29	128
DIMACS (clustering)	dolphins	62	318	4	8	15
	lesmis	77	254	4	9	18
	polbooks	105	882	5	10	23
	adjnoun	112	850	5	11	24
	football	115	1226	5	11	24
	jazz	198	2742	5	14	37
	celegansneural	297	2148	6	17	52
	celegans_metabolic	453	2025	6	21	74
	email	1133	5451	7	34	161
	power	4941	13188	9	70	581
	PGPgiantcompo	10680	24316	9	103	1151
	as-22july06	22963	96872	10	152	2287
DIMACS (Walshaw)	add20	2395	7462	8	49	308
	data	2851	15093	8	53	358
	3elt	4720	13722	8	69	558
	uk	4824	6837	8	69	569
	add32	4960	9462	9	70	583
	bcsstk33	8738	291583	9	93	963
	whitaker3	9800	28989	9	99	1066
	crack	10240	30380	9	101	1109
	wing_nodal	10937	75488	9	105	1176
	fe_4elt2	11143	32818	9	106	1196
	vibrobox	12238	165250	9	111	1300
	4elt	15606	45878	10	125	1616
	fe_sphere	16386	49152	10	128	1689
	cti	16840	48232	10	130	1730
	memplus	17758	54196	10	133	1815
	cs4	22499	43858	10	150	2245

Table 1: Main features of the real-world benchmark instances

	p		
δ	$\ln n$	$\sqrt{n}$	$n/\ln n$
Dense	0.12%	0.59%	5.57%
Sparse	0.29%	1.90%	28.85%
Planar	11.39%	40.03%	296.83%

Table 2: Relative difference $\delta = (z_{UB} - z_{LB})/z_{LB}$ between the best known value z_{UB} and the combinatorial lower bound z_{LB} for the nine classes of instances: the difference strongly increases as the graph becomes sparser and the number of subgraphs increases

Table 2 shows how the density of the graph and the cardinality of the partition influence the distance between the two bounds, measured by the relative difference $\delta = (z_{UB} - z_{LB})/z_{LB}$.

6.2. *Tuning of the* 2L-TS *and the* ALNS

Both the *2L-TS* and the *ALNS* have been tuned on the dedicated benchmark instances. The tuning process is described in more detail in Appendix A and Appendix B, because it is rather long, but also provides useful insights for the development of the algorithms. As for the *2L-TS*, these experiments showed that the tenure should depend on the size of the graph n and on the cardinality p of the partition. The use of a multilevel objective also proved useful. However, only the aggregation mechanism was able to reach satisfactory results. In Table 3 we report the best values for the *2L-TS* parameters.

As for the *ALNS*, the fraction of removed vertices q (both in the random and in the Shaw removal heuristic) and the randomness ρ coefficient of the Shaw removal heuristic exhibit a rather large range of values that are not statistically different. On the other hand, the best probability to accept a worsening solution proves zero, which strongly simplifies the algorithm tuning, as several other parameters can be ignored (e.g., σ_1 and σ_2). The number of iterations NS before updating the learning parameters has been set to apply a sufficient number of times each insertion and removal heuristic. The reaction factor r is drawn from

Parameter	Value
$[l_{\min}, l_{\max}]$	$[0.75\sqrt{np}, \sqrt{np}]$
(α_b, α_e)	$(0.25, 0.5)$

Table 3: Best parameter values for the *2L-TS*

Parameter	Value
q (random)	0.10
q (Shaw)	0.15
ρ	50
τ_0	0
NS	1 000
r	0.1

Table 4: Best parameter values for the *ALNS*

the literature (Røpke and Pisinger, 2006). Table 4 summarizes the final values adopted for the parameters of the *ALNS*.

6.3. Robustness of the ALNS

While the *2L-TS* is deterministic, the *ALNS* applies random moves. This means that the above reported results might exhibit random fluctuations and the effectiveness of the algorithm could sometimes be the result of a lucky choice. It is therefore interesting to evaluate how robust these results are with respect to the choice of the random seed. In order to investigate this point, we have chosen 9 representative instances (namely, the first instance with 500 vertices for each of the three graph topologies and three cardinalities). Then, we have run the algorithm with 30 different values of the random seed, setting the time limit to five minutes, as in the previous experiments. Figure 2 shows the empirical distribution function of the percentage gap with respect to the upper bound, δ_{UB}. In particular, the behaviour of the fraction of runs with $\delta_{UB} \leq \delta$ is plotted as a function of δ. The dotted black line represents the value of δ_{UB} obtained by the starting run of ALNS. We can observe that very often, in all the runs, a value of δ_{UB} close to 0 is obtained (e.g., in instances $n500plap1i1$ and $n500plap2i1$

$\delta_{UB} = 0$ for every run). Only in three instances the $ALNS$ finds values of δ_{UB} not close to 0: for instance $n500d06p3i1$, δ_{UB} is close to 2% in all the 30 runs, and this is also the value obtained with the original seed value; for instance $n500d03p3i1$, δ_{UB} varies between 1% and 4% over the 30 runs; for instance $n500plap3i1$, δ_{UB} varies between 0 and 9%. Therefore, these experiments allow to confirm the robustness of $ALNS$ and its capability to detect good quality solutions.

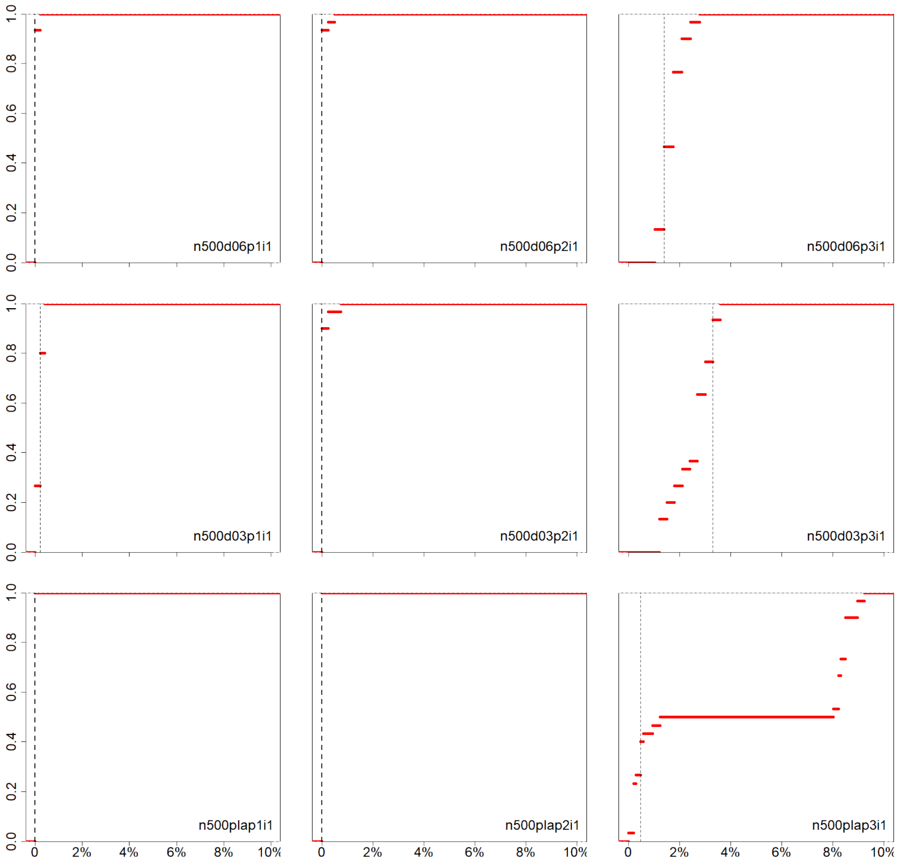

Figure 2: Empirical distribution function of the gap with respect to the upper bound over 30 runs of the $ALNS$ on 9 representative instances with 500 vertices: the upper line corresponds to planar instances, the central line to sparse instances, the lower line to dense instances; the number of subgraphs increases from left to right

6.4. Comparison between the 2L-TS *and the* ALNS

Detailed results for the *2L-TS* and the *ALNS* on the single instances can be found at `https://homes.di.unimi.it/cordone/research/research.html`. Table 5 reports the average values of δ_{LB} and δ_{UB} on the whole benchmark. There are three blocks of rows, for planar, sparse and dense graphs, respectively. Each block consists of three rows, corresponding to different values of the number of subgraphs p ($\ln(n)$, $\sqrt{n}$ and $n/\ln(n)$). We emphasize in boldface the better between the values obtained by the two algorithms. In most cases, the *ALNS* outperforms the *2L-TS*: this occurs when the number of subgraphs is larger ($p = \sqrt{n}$ or $p = n/\ln(n)$). When this number is smaller, the *2L-TS* is slightly better than the *ALNS*. A possible explanation of this different performance could be that the removal and insertion heuristics of *ALNS* become less effective due to the low number of subgraphs, since most removed vertices are simply reinserted in their original position. The last row of the table provides the overall average values, according to which the performance of the *ALNS* is superior. This is confirmed by the number of instances in which it finds the overall best known result: 157 versus 94, out of 225. Since both algorithms find the best known result in 74 cases, there are 83 best known results found by the *ALNS* approach that are not detected by the *2L-TS*, while the opposite occurs only for 20 instances.

Figures 3 and 4 show the behavior of δ_{UB} varying the size of the instances, for the *2L-TS* and the *ALNS*, respectively. The general behavior of the *2L-TS* consists in an increase in δ_{UB} as the size of the instance becomes larger. The *ALNS* has a less clear behavior, since for some classes δ_{UB} decreases as the instance size grows when the number of subgraphs p is small, whereas it tends to increase in the other cases. In fact, these are the instances where the *2L-TS* performs slightly better than the *ALNS*.

These results refer to the *tuning* benchmark set, which of course could be affected by overfitting. To investigate whether this is actually the case, we have also run the two algorithms on the *evaluation* benchmark set. Figure 5 provides the empirical distribution function of the gap between the result

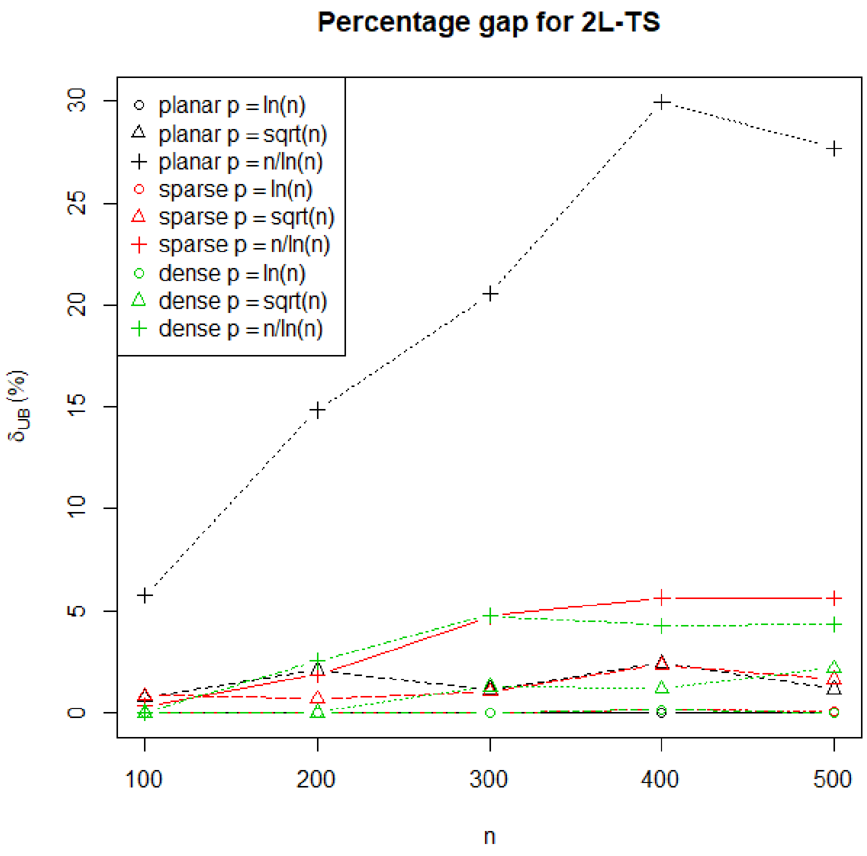

Figure 3: Average behavior of δ_{UB} over different sizes of the instances for *2L-TS*

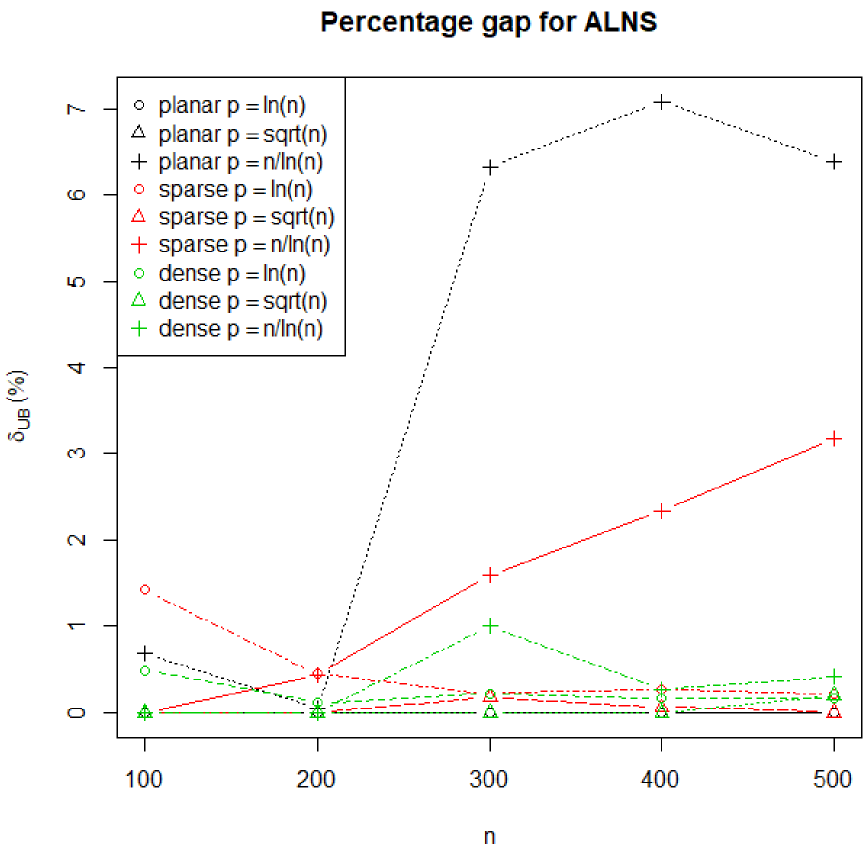

Figure 4: Average behavior of δ_{UB} over different sizes of the instances for *ALNS*

		2L-TS		*ALNS*	
	p	δ_{UB}	δ_{LB}	δ_{UB}	δ_{LB}
Planar	$\ln(n)$	**0.00%**	**11.39%**	**0.00%**	**11.39%**
	$\sqrt{n}$	1.49%	42.11%	**0.00%**	**40.03%**
	$n/\ln(n)$	19.75%	380.01%	**4.11%**	**315.11%**
Sparse	$\ln(n)$	**0.04%**	**0.33%**	0.52%	0.82%
	$\sqrt{n}$	1.29%	3.22%	**0.05%**	**1.95%**
	$n/\ln(n)$	3.63%	33.39%	**1.51%**	**30.73%**
Dense	$\ln(n)$	**0.02%**	**0.14%**	0.23%	0.35%
	$\sqrt{n}$	0.93%	1.46%	**0.04%**	**0.57%**
	$n/\ln(n)$	3.16%	8.85%	**0.34%**	**5.92%**
Average		3.37%	53.44%	**0.75%**	**45.21%**

Table 5: Average gap with respect to the best known upper and lower bounds of the results obtained by the *2L-TS* and the *ALNS* on different classes of instances

of each algorithm and the lower bound, for the tuning benchmark and the evaluation benchmark, respectively. These diagrams are conceptually similar to the so called *performance profiles* (Dolan and Moré, 2002), which provide the empirical distribution function of the ratio between the result of each algorithm and the best result among the algorithms considered in the comparison. Since the lower bound is computed with a procedure independent from any heuristic and the two benchmarks have the same structure, we consider the lower bound as a proper reference to obtain a fair comparison between different algorithms on different benchmarks.

We can observe that *ALNS* dominates the *2L-TS* both in the *tuning* benchmark and in the *evaluation* benchmark. Moreover, the diagrams of the two algorithms are very similar in the two benchmarks and the gaps look even slightly smaller in the *evaluation* one. This suggests that there is no overfitting.

Finally, we discuss the performance of the two algorithms on the real-world benchmark instances. Overall, the *ALNS* finds the best known value for 91

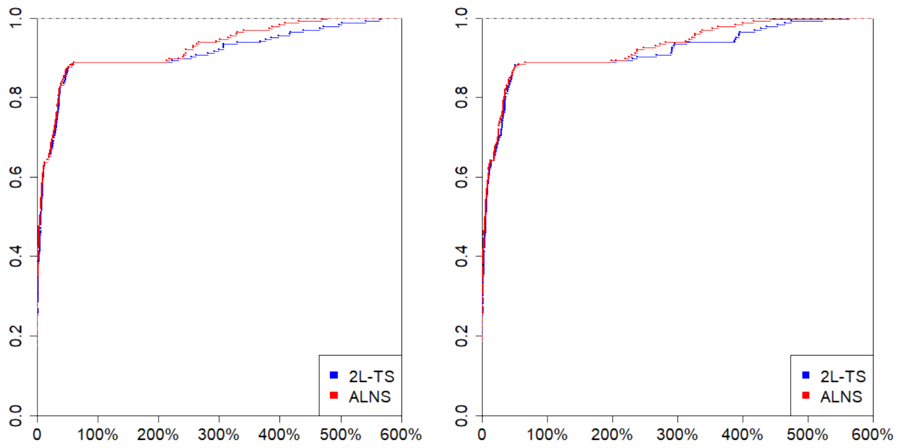

Figure 5: Distribution function of the percentage gap with respect to the lower bound (i.e., fraction of instances with $\delta_{LB} \leq \delta$) for *2L-TS* and *ALNS* applied to the tuning benchmark (on the left) and the evaluation benchmark (on the right)

instances out of 111 (82%), while the *2L-TS* does the same for 41 instances (37%). Figure 6 reports the performance profiles of the two algorithms for the 111 real-world instances divided into the four classes: hydraulic, agricultural, DIMACS-clustering and DIMACS-Walshaw. This diagram shows the fraction of instances where each algorithm obtains a value having ratio $\leq \eta$ with the better of the two. We can observe that the two metaheuristics are roughly comparable for the hydraulic instances. The *ALNS* is better for the agricultural and the Walshaw benchmark. For the clustering instances, the *ALNS* has more best results, but there is a small number of instances for which the *2L-TS* is strongly better. Since all the 20 real-world instances in which this happens are among the largest ones and they are all sparse, this behaviour could depend on a better scaling of the *2L-TS*; in fact, for such instances each iteration of the *2L-TS* takes linear time, as opposed to the quadratic time of the *ALNS* (see Figures A.10 and B.16).

To investigate whether this behaviour can be due, instead, to randomisation, we selected the 6 largest instances in which *2L-TS* performs better than *ALNS* and we ran 30 times the latter with different random seeds. Figure 7 shows the distribution of η. In particular, the dashed vertical line denotes the solution value obtained for the starting seed and we can see that it falls not far from the median of the solution values obtained with the other runs, and that all are worse than that of *2L-TS*. This suggests that the previous result is due to scaling rather than randomisation issues.

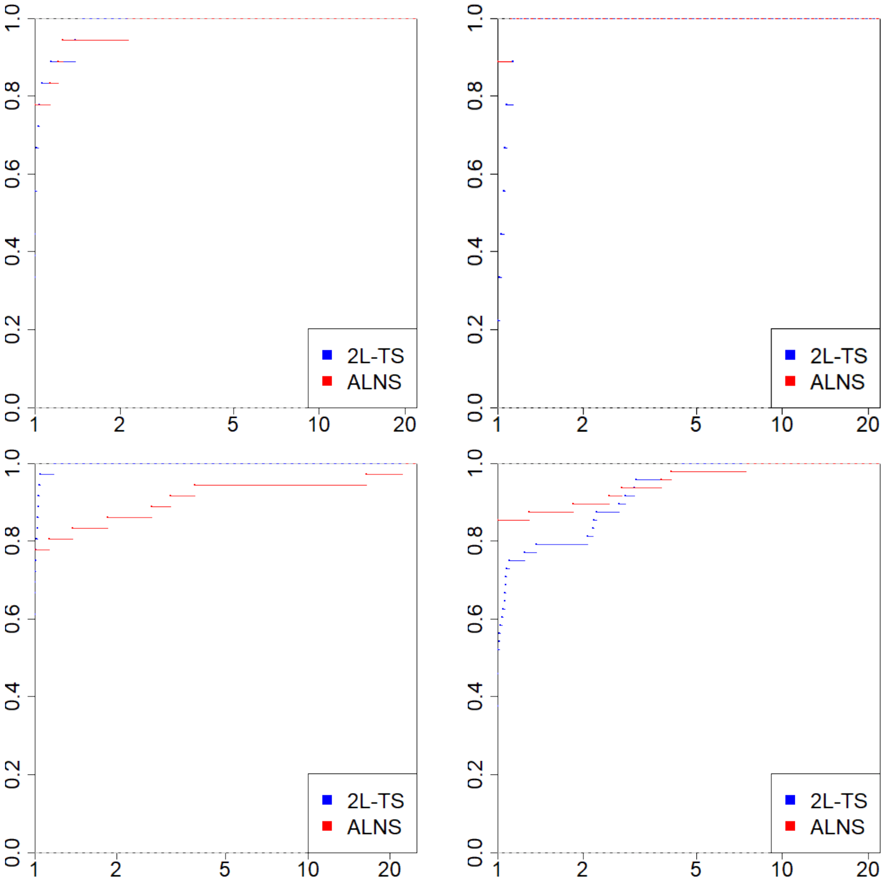

Figure 6: Performance profiles of *2L-TS* and *ALNS*, i.e., fraction of instances for which the result z of each algorithm is not worse than η times the better of the two results: $z \leq \eta \min(z_{\mathrm{2L-TS}}, z_{\mathrm{ALNS}})$ on the four real-world benchmark classes: hydraulic (upper left), agricultural (upper right), clustering (lower left) and Walshaw (lower right)

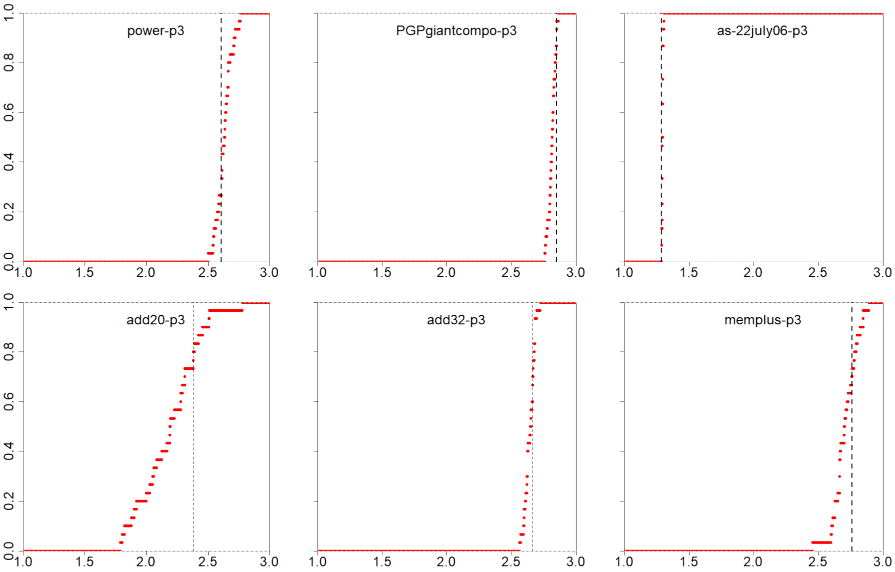

Figure 7: Empirical distribution function of the ratio η with respect to the best known value over 30 runs of the *ALNS* on 6 larger DIMACS instances (clustering on the upper line, Walshaw on the lower line). The number of vertices increases from left to right.

7. Conclusions

In this paper, we investigate the *Minimum Gap Graph Partitioning Problem* (*MGGPP*), that is a partitioning problem on vertex-weighted graphs, aiming to produce connected subgraphs with a narrow internal distribution of weights. We tackle the problem with two competing approaches. Due to the misleading nature of the objective function and the strong limitation imposed by the connectivity constraint, a basic Tabu Search algorithm exchanging vertices between subgraphs proves little effective. We therefore enhance it with a two-level approach that builds groups of vertices and moves them jointly as macro-vertices. On the other hand, we develop an Adaptive Large Neighbourhood Search algorithm that alternatively removes vertices with different strategies and reinserts them with greedy randomized procedures, and progressively adapts the frequency of each strategy based on the quality of the results obtained. The adaptive tuning mechanism and the experimental advantage to accept only improving solutions allow to use a particularly small number of parameters. Computational experiments have been performed on a large benchmark set of instances of different size (from 100 to 500 vertices), topology and density (planar, sparse and dense graphs), and cardinality of the partition ($p = \ln(n)$, $p = \sqrt{n}$ and $p = n/\ln(n)$), and on a benchmark set of real-world graphs (from 60 to 23 000 vertices) deriving from hydraulic and agriculture applications and from the 10th DIMACS challenge on graph partitioning. In general, the *ALNS* algorithm outperforms the *2L-TS* algorithm, since the former finds the overall best known result in 157 cases versus 94 of the tuning benchmark set, in 168 versus 96 of the evaluation benchmark set, and in 91 versus 41 of the real-world instances. The *2L-TS* algorithm, however, appears to scale better for the sparser instances as the size increases. With respect to a combinatorial lower bound, the solution quality of *ALNS* is very good on dense graphs and on sparse ones for small and medium values of p (i.e., $\ln(n)$ and $\sqrt{n}$), since in these cases the average gap with the lower bound is smaller than 1%, whereas it becomes almost 6% on dense graphs for $p = n/\ln(n)$ and it is very high on planar graphs. This is probably due to the fact that the lower bound is very weak on planar graphs since it gets tighter as the graph becomes denser, providing exactly the optimum for complete graphs. For this reason, our future work will concern the development of methods to find tighter bounds also for sparser graphs and high values of p. For instance this could be obtained considering the relaxation of different formulations of the problem or stronger combinatorial bounding techniques.

References

References

Albornoz, V., Véliz, M., Ortega, R., Ortíz-Araya, V., 2020. Integrated versus hierarchical approach for zone delineation and crop planning under uncertainty. Annals of Operational Research 286, 617—-634.

Bader, D.A., Kappes, A., Meyerhenke, H., Sanders, P., Schulz, C., Wagner, D. (Eds.), 2014. In Encyclopedia of Social Network Analysis and Mining. Springer.

Bader, D.A., Meyerhenke, H., Sanders, P., Wagner, D. (Eds.), 2013. Graph Partitioning and Graph Clustering. volume 588 of *Contemporary Mathematics*. American Mathematical Society, Providence, Rhode Island.

Bichot, C.E., Siarry, P. (Eds.), 2013. Graph Partitioning. Wiley-ISTE.

Bruglieri, M., Cordone, R., 2016. Partitioning a graph into minimum gap components. Electronic Notes in Discrete Mathematics 55, 33–36.

Bruglieri, M., Cordone, R., Caurio, V., 2017. A metaheuristic for the minimum gap graph partitioning problem, in: Proceedings of the 15th Cologne-Twente Workshop on Graphs and Combinatorial Optimization, pp. 23–26.

Bruglieri, M., Cordone, R., Lari, I., Ricca, F., Scozzari, A., 2018. A metaheuristic for the minimum gap graph partitioning problem, in: Proceedings of the 16th Cologne-Twente Workshop on Graphs and Combinatorial Optimization, pp. 107–110.

Chlebikova, J., 1996. Approximability of the maximally balanced connected partition problem in graphs. Information Processing Letters 60, 225–230.

Cordeau, J.F., Laporte, G., Pasin, F., S., R., 2010. Scheduling technicians and tasks in a telecommunications company. Journal of Scheduling 13, 393–409.

Dolan, E.D., Moré, J.J., 2002. Benchmarking optimization software with performance profiles. Mathematical Programming, Series A 91, 201–213.

Dunn, O.J., 1961. Multiple comparisons among means. Journal of the American Statistical Association 56, 52–64.

García, S., Fernández, A., Luengo, J., Herrera, F., 2010. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. Information Sciences 180, 2044—-2064.

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13, 533–549.

Glover, F.W., Laguna, M., 1997. Tabu Search. Kluwer.

Goldschmidt, O., Hochbaum, D.S., 1988. Polynomial algorithms for the k-cut problem, in: Proceedings of the 29th annual IEEE Symposium on Foundations of Computer Sciences (*FOCS*), pp. 444–451.

Gomes, R., Sousa, J., Muranho, J., Marques, A.S., 2015. Different design criteria for district metered areas in water distribution networks. Procedia Engineering 119, 1221–1230.

Hochbaum, D., 2019. Algorithms and complexity of range clustering. Networks 73, 170–186.

Hubert, L.J., 1974. Some applications of graph theory to clustering. Psychometrika 39, 283–309. doi:10.1007/BF02291704.

Ito, T., Nishizeki, T., Schröder, M., Uno, T., Zhou, X., 2012. Partitioning a weighted tree into subtrees with weights in a given range. Algorithmica 62, 823–841.

Ito, T., Zhou, X., Nishizeki, T., 2007. Partitioning a weighted graph to connected subgraphs of almost uniform size. IEICE Transactions on Information and Systems E90-D, 449–456.

Krauthgamer, R., Naor, J., Schwartz, R., 2009. Partitioning graphs into balanced components, in: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA. pp. 942–949.

Lari, I., Ricca, F., Puerto, J., Scozzari, A., 2016. Partitioning a graph into connected components with fixed centers and optimizing cost-based objective functions or equipartition criteria. Networks 67, 69–81. doi:10.1002/net.21661.

Lucertini, M., Perl, Y., Simeone, B., 1993. Most uniform path partitioning and its use in image processing. Discrete Applied Mathematics 42, 227–256.

Muller, L.F., 2009. An adaptive large neighborhood search algorithm for the resource-constrained project scheduling problem, in: Proceedings of the VIII Metaheuristics International Conference (MIC), Hamburg, Germany, pp. 1–10.

Muller, L.F., Spoorendonk, S., Pisinger, D., 2012. A hybrid adaptive large neighborhood search heuristic for lot-sizing with setup times. European Journal of Operational Research 218, 614–623. doi:https://doi.org/10.1016/j.ejor.2011.11.036.

Paola, F.D., Fontana, N., Galdiero, E., Giugni, M., degli Uberti, G.S., Vitaletti, M., 2014. Optimal design of district metered areas in water distribution networks. Procedia Engineering 70, 449–457.

Pisinger, D., Røpke, S., 2010. Large neighborhood search, in: Gendreau, M., Potvin, J.Y. (Eds.), Handbook of Metaheuristics. Springer, pp. 399–420.

Prim, R.C., 1957. Shortest connection networks and some generalizations. Bell System Technical Journas 36, 1389–1401.

Røpke, S., Pisinger, D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation science 40, 455–472.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems, in: International Conference on Principles and Practice of Constraint Programming, Springer. pp. 417–431.

Stadler, P.F., 1996. Landscapes and their correlation functions. Journal of Mathematical Chemistry 20, 1–45.

Tang, X., Soukhal, A., T'kindt, V., 2014. Preprocessing for a map sectorization problem by means of mathematical programming. Annals of Operations Research 222, 551–569.

Walshaw, C., 2004. Multilevel refinement for combinatorial optimisation problems. Annals of Operations Research 131, 325–372.

Wang, Q., Guidolin, M., Savic, D., Kapelan, Z., 2014. Two-objective design of benchmark problems of a water distribution system via MOEAs: Towards the best-known approximation of the true pareto front. Journal of Water Resources Planning and Management 141, 04014060–1–14.

Wilcoxon, F., 1945. Individual comparisons by ranking methods. Biometrics 1, 80–83.

Xiao, L., Hongpeng, L., Dongling, N., Yan, W., Gang, L., 2015. Optimization of GNSS-controlled land leveling system and related experiments. Transactions of the Chinese Society of Agricultural Engineering 31, 48–55.

Appendix A. Tuning of the two-level Tabu Search algorithm

This appendix reports detailed information on the tuning of the two-level Tabu Search algorithm, and the motivations of its main design choices. All the following analyses, therefore, refer to the 225 instances of the *tuning* benchmark.

Tabu tenure. First, we considered different ranges of values for the tabu tenure of the basic Tabu Search procedure. Preliminary investigations showed that the use of a fixed tenure is strongly ineffective. On the one hand, large instances with many subgraphs require large tenures to avoid cyclic behaviors; on the other hand, small instances with few subgraphs require small tenures to avoid forbidding all moves. Inspired by a remark by Glover and Laguna (1997), we decided to adopt a tenure proportional to the square root of the number of attributes, that is $\sqrt{np}$. We therefore let the tabu tenure vary in the range $[l_{\min}, l_{\max}]$, setting $l_{\min} = \alpha_{\min}\sqrt{np}$ and $l_{\max} = \alpha_{\max}\sqrt{np}$, where the coefficients $\alpha_{\min}$ and $\alpha_{\max}$ assume all pairs of values in $\{0.5, 0.75, 1, 1.25, 1.5\}$ such that $\alpha_{\min} \leq \alpha_{\max}$. This allows to consider both fixed tenures (when $\alpha_{\min} = \alpha_{\max}$) and variable tenures, with a more or less wide range of variation. The Tabu Search procedure was then run on the benchmark instances for $T = 10\,000$ iterations with no aggregation of vertices into macro-vertices.

Table A.6 reports the results: rows and columns correspond to the possible values of $\alpha_{\min}$ and $\alpha_{\max}$, respectively. Each cell reports the average values of δ_{UB} and δ_{LB} obtained by the corresponding configuration. Both estimates of the gap exhibit a smooth profile, with a minimum value (marked in bold) surrounded by gradually worsening results as $\alpha_{\min}$ and $\alpha_{\max}$ move farther away. Unfortunately, the optimal tunings for δ_{UB} and δ_{LB} are different. Since this is mainly due to the planar instances, where the quality of the lower bound is particularly bad, we adopted the first configuration ($\alpha_{\min} = 0.75$ and $\alpha_{\max} = 1$) and compared its results with those of the other ones with Wilcoxon's signed rank test (Wilcoxon, 1945) applying the Bonferroni correction (Dunn, 1961) to account for the family-wise error, i.e. the possible false positives deriving from the multiple pairwise comparisons thus performed. The cells shaded in grey

$\alpha_{\min} \backslash \alpha_{\max}$	0.50	0.75	1.00	1.25	1.50
0.50	34.05% 137.48%	33.35% 134.24%	33.58% 134.93%	33.99% 136.24%	34.17% 136.61%
0.75	- -	31.14% 125.45%	**30.59%** 123.71%	31.32% 125.76%	31.51% 126.02%
1.00	- -	- -	30.73% **122.75%**	30.84% 123.69%	31.83% 126.78%
1.25	- -	- -	- -	31.37% 124.45%	32.55% 128.30%
1.50	- -	- -	- -	- -	33.34% 131.87%

Table A.6: Average gap with respect to upper and lower bounds of the results obtained after 10 000 iterations of the basic Tabu Search with a tenure varying in $\{\alpha_{\min}\sqrt{np}, \ldots, \alpha_{\max}\sqrt{np}\}$

indicate the configurations for which the difference with respect to the chosen one is not statistically significant, i. e., the adjusted P-value (García et al., 2010) of the test is $\geq 5\%$.

Figure A.8 shows the *performance profiles* (Dolan and Moré, 2002) of the basic Tabu Search with different ranges of the tenure parameter. For each pair of values of α_{min} and α_{max} in the range $\{0.5, 0.75, 1, 1.25, 1.50\}$, a profile plots the fraction of instances where the result was not larger than η times the best known value z_{UB} as a function of the coefficient η[3]. We can observe that several diagrams strongly overlap, in agreement with Table A.6, where several pairs α_{min}-α_{max} do not produce significantly different results. However, the pair $\alpha_{min} = 0.75$ and $\alpha_{max} = 1.00$ provides a diagram (emphasized in black) that dominates many of the others and performs best on average.

Overall, the quality of these results looks disappointing, motivating first a

[3] The classical performance profiles use the best result over the configurations tested as a reference; we consider the overall best known result as more significant.

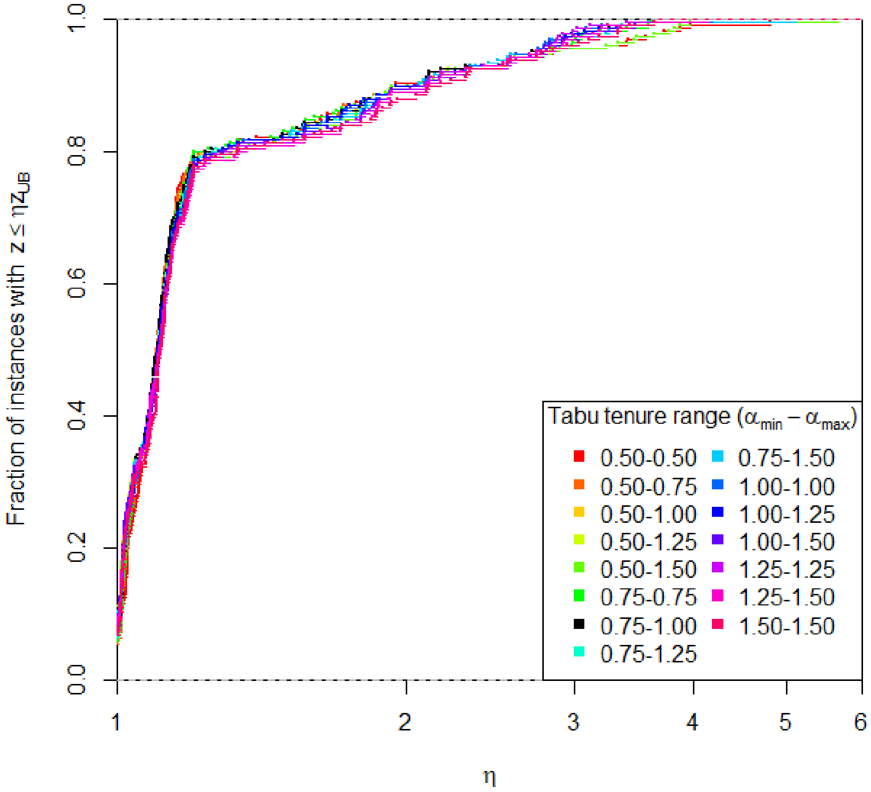

Figure A.8: Performance profiles of the Tabu Search algorithm with different tenure ranges $[\alpha_{\min}\sqrt{np}, \alpha_{\max}\sqrt{np}]$

	Basic	Intermediate	Full-fledged
δ_{UB}	41.08%	30.99%	30.59%
δ_{LB}	140.35%	123.83%	123.71%

Table A.7: Average gap with respect to the upper and lower bounds of the results obtained after 10 000 iterations of the basic (actual objective), the intermediate (three-level objective) and the full-fledged Tabu Search (three-level objective and recency)

deeper analysis of the algorithm, then the search for alternative, more effective mechanisms.

Plateau moves. Our second batch of experiments focused on the use of the lexicographic objective function to discriminate between alternative moves. We compared the basic version of the Tabu Search algorithm, considering only the actual objective function, an intermediate version, which includes also the second and third look-ahead criteria, and the full-fledged version, which also breaks ties choosing the less recent move. All three versions were run for 10 000 iterations to determine whether the additional information exploited to choose among moves is actually useful. Table A.7 reports the average values of the gap estimates δ_{UB} and δ_{LB}: the full-fledged version proves better than the intermediate one, which dominates the basic version. Wilcoxon's test with the Bonferroni correction suggests that, in fact, the basic version is significantly dominated by the intermediate one ($P = 7.39 \cdot 10^{-13}$) and this is weakly dominated by the full-fledged one ($P = 1.79\%$). In the following we adopt the full-fledged version.

Figure A.9 shows the performance profiles of the three variants of Tabu Search described in Section 4. We can observe that the diagram of the full-fledged TS strongly dominates the diagram of the basic TS and slightly dominates the one of the intermediate TS.

Computational time of the Tabu Search for different classes of instances. Figure A.10 reports the average computational time required to perform 10 000

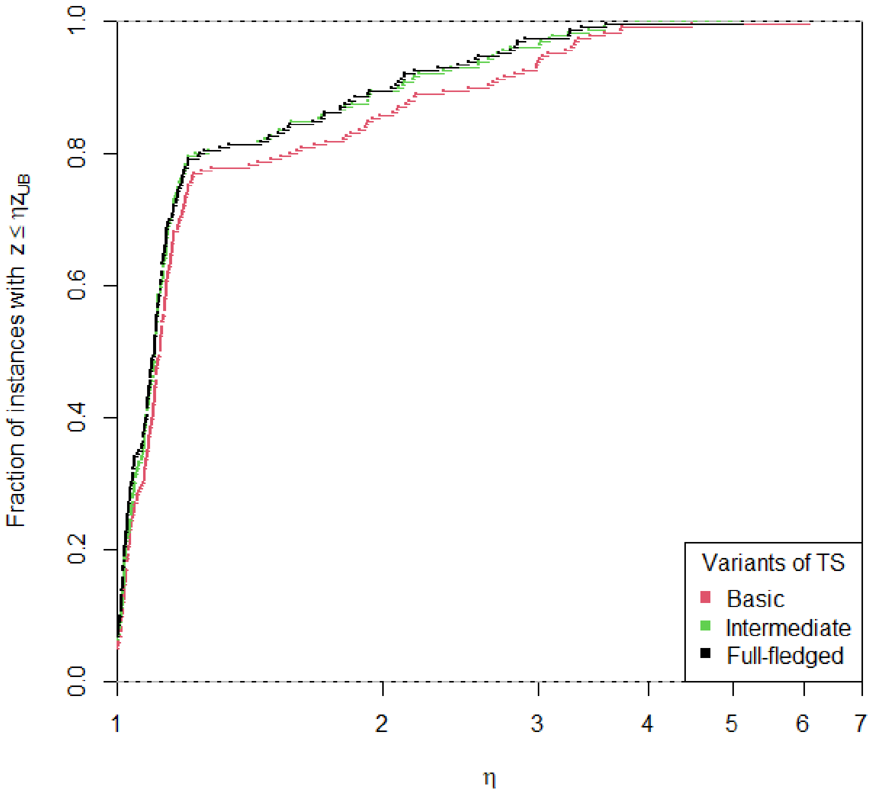

Figure A.9: Performance profiles of the three Tabu Search variants concerning the objective function

Tabu Search iterations on each group of 5 benchmark instances with the same size n, topology (planar, sparse and dense) and value of p ($\ln n$, $\sqrt{n}$ and $n/\ln n$). A least-square interpolating profile is drawn for each class of instances to estimate the dependence of the computational time on the size of the instance. Quite clearly, the density of the graph strongly influences the computational time (denser instances take more time), while the overall number of clusters has a weaker, and less consistent, effect (more clusters, but in some cases also less clusters, imply more time).

The interpolation suggests that for planar instances the computational time is slightly less than linear in n, especially for large numbers of clusters. In these instances, in fact, each vertex tends to have a constant number of adjacent vertices. Since some vertices cannot be moved, because they would disconnect a subgraph or they are surrounded by vertices of the same subgraph, the overall number of neighbor solutions increases slightly less than linearly with n. For the sparse and dense instances, on the contrary, each vertex has on average a linear number of adjacent vertices, part of them belonging to different subgraphs, and the computational time increases quadratically with the number of vertices.

Restart. Even with the best tuning of the tabu tenure, the basic Tabu Search procedure is clearly unable to obtain good solutions in short time. We have then evaluated the possible improvements obtained by increasing the computational time and restarting the search when it appears to be unpromising. Since the most challenging instances are the planar ones and the complexity of the neighbourhood exploration is nearly linear on these instances, we have set an overall running time of $n/100$ minutes (i.e., from one to five minutes according to the size), and we have imposed a restart from a random solution after different numbers of nonimproving iterations, specifically $T_{ni} \in \left\{10^2, 10^3, 10^4, 10^5, 10^6\right\}$, where the first value corresponds to a very frequent restart, with a search phase possibly shorter than the tabu tenure, and the last value to at most 10 restarts during the whole computation. Table A.8 reports the results, as usual providing the average gaps with respect to the upper and lower bound. As in the previ-

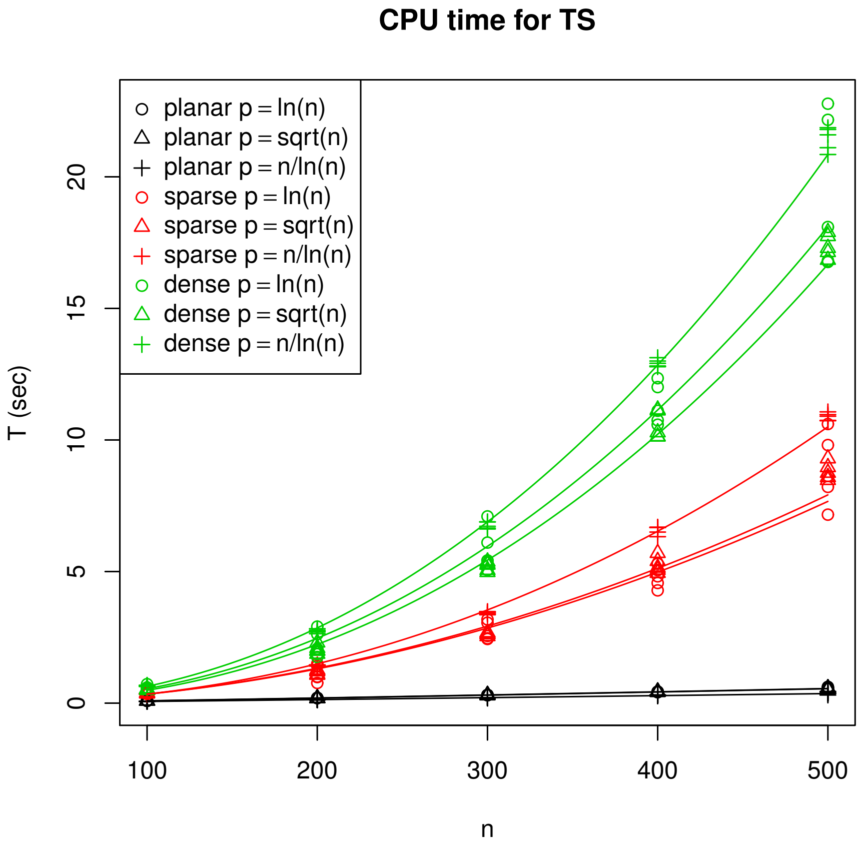

Figure A.10: Average computational time (in seconds) required to perform 10 000 iterations of the Tabu Search algorithm on instances of the nine classes

T_{ni}	100	1 000	10 000	100 000	1000 000
δ_{UB}	19.63%	18.08%	**16.80%**	19.65%	22.28%
δ_{LB}	104.06%	98.53%	**93.28%**	96.69%	99.83%

Table A.8: Average gap with respect to the best known upper and lower bounds of the results obtained by the Tabu Search, with a restart after different numbers T_{ni} of nonimproving iterations

ous tables, the best result is bolded: it corresponds to a restart of the search after 10 000 nonimproving iterations. The other tunings are significantly worse according to Wilcoxon's test with the Bonferroni correction ($P = 1.29\%$ for $T_{ni} = 1,000$ and $P < 10^{-5}$ for the other values).

Figure A.11 shows the performance profiles of the Tabu Search with different restart frequencies. The best one is obtained for $T_{ni} = 10\,000$, in agreement with Table A.8.

Though better, the results are still not satisfactory and call for a change in the structure of the algorithm.

Two-level Tabu Search. The bad performance of the basic Tabu Search procedure mainly concerns the planar instances: the overall 16.80% average gap actually combines a 5.67% average gap on the sparse and dense instances and a 39.20% gap on the planar ones. This is the effect of two main limitations of the local search: first, the objective function, even if enhanced by the lexicographic objective, is unable to clearly discriminate between promising and unpromising moves; second, in planar graphs it is difficult to move vertices from cluster to cluster while keeping them connected. Random restarts can move the search to different regions of the solution space, but does not allow to explore it effectively. The search can be strongly improved by moving suitably identified macro-vertices, as discussed in Section 4.1. We have therefore divided the the overall computational time of $n/100$ minutes into $n/2 - p$ equal intervals of length $\tau = n/(100(n/2-p))$, one for each possible value of the number of macro-vertices $\tilde{p}$ (from $p+1$ to $n/2$). Then, we have divided each interval into three

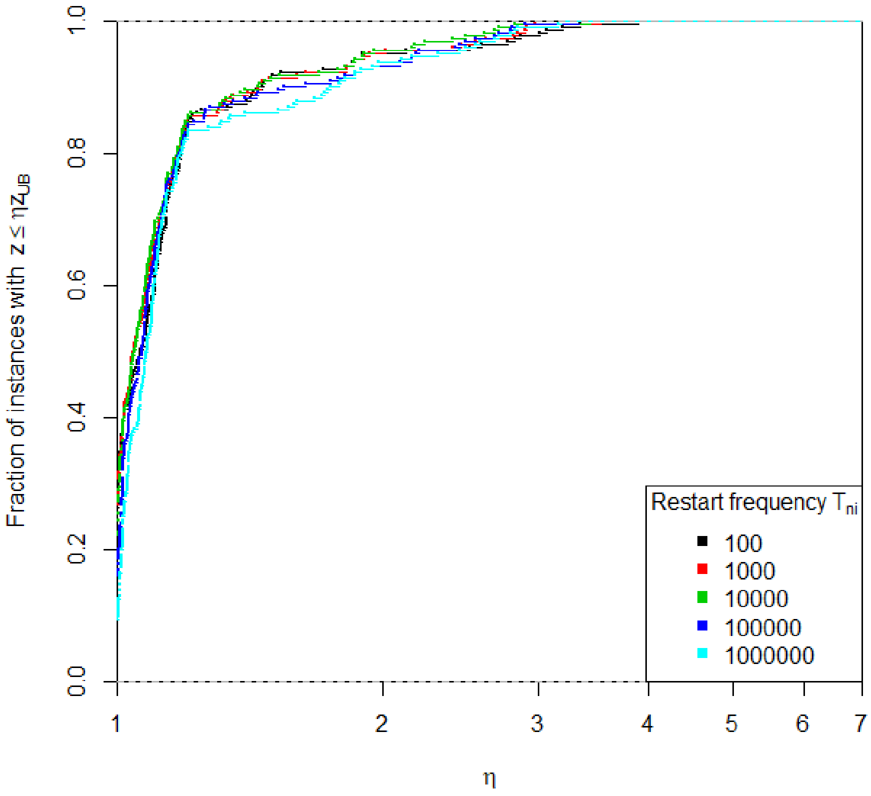

Figure A.11: Performance profiles of the Tabu Search algorithm with different restart frequencies

sequential phases. The building phase applies for $\alpha_b\tau$ minutes the constructive and Tabu Search procedures to the original graph in order to build $\tilde{p}$ clusters, that are converted into macro-vertices of an aggregated graph. Then, the exchange phase applies for $\alpha_e\tau$ minutes the greedy and Tabu Search procedures to partition the aggregated graph into p clusters. Finally, a refinement phase applies for $(1-\alpha_b-\alpha_e)\tau$ minutes the Tabu Search procedure to the original graph, so as to refine the p clusters obtained from the exchange phase. We have generated 15 configurations extracting the values of the two coefficients α_b and α_e from $\{0, 0.25, 0.5, 0.75, 1\}$ so that $\alpha_b + \alpha_e \leq 1$. When $\alpha_b = 0$ (respectively, $\alpha_e = 0$), the building (respectively, the exchange) phase applies only the greedy procedure; when $\alpha_b+\alpha_e = 1$, the refinement phase is skipped. The Tabu Search procedure is always applied with the best tabu tenure tuning identified above.

Table A.9 reports the results of the *2L-TS*: rows and columns correspond to the possible values of α_b and α_e, respectively. Each cell reports the average values of δ_{UB} and δ_{LB} obtained by the corresponding configuration. The best configuration with respect to both indicators is $\alpha_b = 0.25$ and $\alpha_e = 0.50$, but the configurations with $\alpha_b = 0.50$ and $\alpha_e = 0.25$, and with $\alpha_b = 0.25$ and $\alpha_e = 0.25$, are nearly as effective and the difference is not statistically significant (these cells, in fact, are shaded in grey in the table). From these results it is reasonable to conclude that all three phases of the algorithm (building good macro-vertices, collecting them into good subgraphs and refining the solution thus obtained) have an important impact on the final result, and that any "balanced" distribution of the computational time among them yields an acceptable behaviour. These results are also confirmed by the performance profiles plotted in Figure A.12.

The quality of the results is clearly much better than that of the basic Tabu Search, even with a restart mechanism (notice that the number of iterations, $\lfloor n/2 \rfloor - p$, is of the same magnitude of the best number of random restarts) We conclude that the aggregation into macro-vertices actually plays a role in building an effective search.

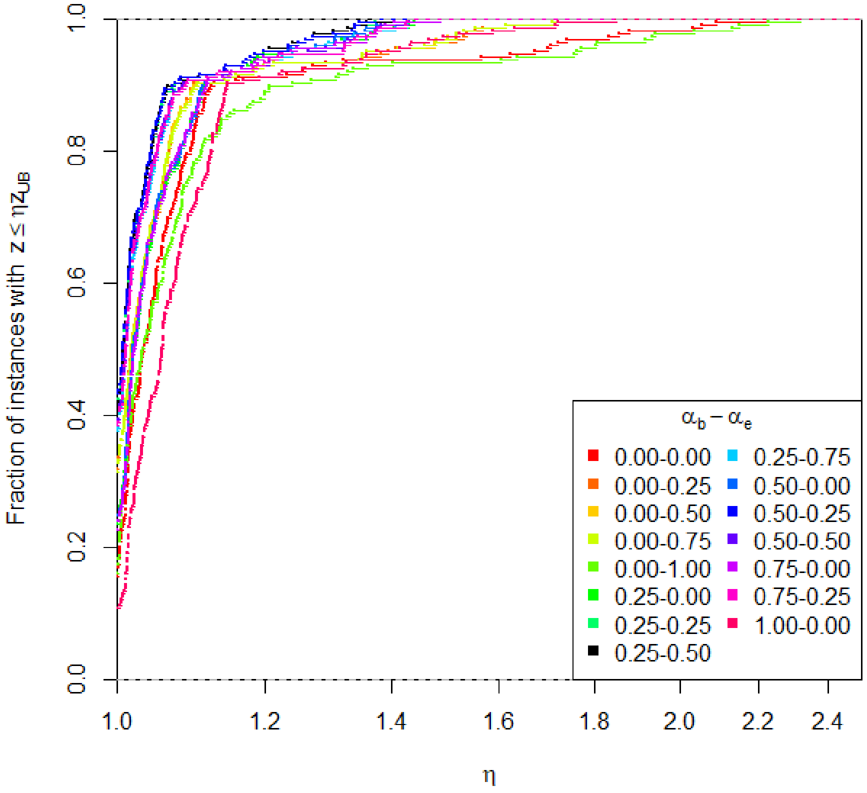

Figure A.12: Performance profiles of the two-level Tabu Search algorithm with different distributions of the computational effort among the building, exchange and refinement phase

$\alpha_b \backslash \alpha_e$	0.00	0.25	0.50	0.75	1.00
0.00	9.44% 72.63%	6.12% 62.14%	6.05% 61.87%	6.09% 62.02%	11.19% 78.44%
0.25	5.26% 57.06%	3.39% 53.49%	**3.37%** **53.44%**	4.16% 55.92%	- -
0.50	5.11% 56.43%	3.37% 53.48%	4.13% 55.91%	- -	- -
0.75	5.28% 57.00%	4.21% 56.12%	- -	- -	- -
1.00	9.17% 66.24%	- -	- -	- -	- -

Table A.9: Average gap with respect to upper and lower bounds of the results obtained in $n/100$ minutes by the two-level Tabu Search with different distributions of the running time among the building, exchange and refinement phase

Appendix B. Tuning of the *ALNS* algorithm

The *ALNS* algorithm has a large number of parameters, as it combines several different insertion and removal procedures. In our case, the insertion procedures have no parameters: each of them is completely characterized by the given cardinality of the list of candidate vertices. Our first round of experiments, therefore, was focused on the removal procedures.

Tuning of the removal parameters. The random removal heuristic RH1 is characterized by the fraction q of vertices removed. In order to tune it, we have provisionally disabled the other removal heuristics and assigned a fixed uniform probability to each of the insertion heuristics. For the sake of simplicity, we have also set to 0 the parameter τ_0 that controls solution acceptance, so that all nonimproving solutions are rejected. Consequently, the cooling parameter h becomes inactive and the results only depend on q. We have considered all values of q in $\{0.05, 0.10, 0.15, 0.20\}$. Notice that q strongly affects the time required

q	0.05	0.10	0.15	0.20
δ_{UB}	9.34%	**7.27%**	8.49%	8.28%
δ_{LB}	60.99%	**57.21%**	62.07%	62.78%

Table B.10: Average gap with respect to the best known upper and lower bounds of the results obtained in $n/1\,000$ minutes by the *ALNS* removing different fractions of the vertices, uniformly chosen at random

by each iteration of the algorithm, because removing more vertices slows down both the removal and the insertion heuristic. In order to make a fair comparison, with stable results, but in a reasonable time, we set a time limit of $n/1\,000$ minutes (from six to thirty seconds), that is comparable to the time used to tune the *TS* algorithm. Table B.10 reports the results of this experiment. It has the following structure: the first row provides the values tested for parameter q, the second one the percent gap with respect to the best known result and the third one the gap with respect to the lower bound. The best average gaps (reported in bold) are obtained by setting $q = 0.10$, that is the tuning adopted in all following experiments, but Wilcoxon's test with the Bonferroni correction suggests that the differences with respect to $q = 0.15$ and $q = 0.20$ are not significant (these results are shaded in grey).

Figure B.13 shows the performance profiles of the *ALNS* with different fractions q of removed vertices in the random removal heuristic. We can observe that the best performance profile is obtained for $q = 0.10$ and there are no significant differences from the others, in agreement with Table B.10.

The worst cluster removal heuristics RH2 and RH3 have no parameters to tune. The Shaw removal heuristics RH4, RH5, RH6 have two parameters: the fraction q of vertices chosen to be removed and the real parameter ρ that tunes the randomness of their choice. We have considered all pairs of values (q, ρ) with $q \in \{0.05, 0.10, 0.15, 0.20\}$ and $\rho \in \{5, 10, 20, 50, 100\}$. The experimental framework adopted is the same: $n/1\,000$ minutes of computation, $\tau_0 = 0$ and equal probabilities for the four insertion heuristics; also the three Shaw removal

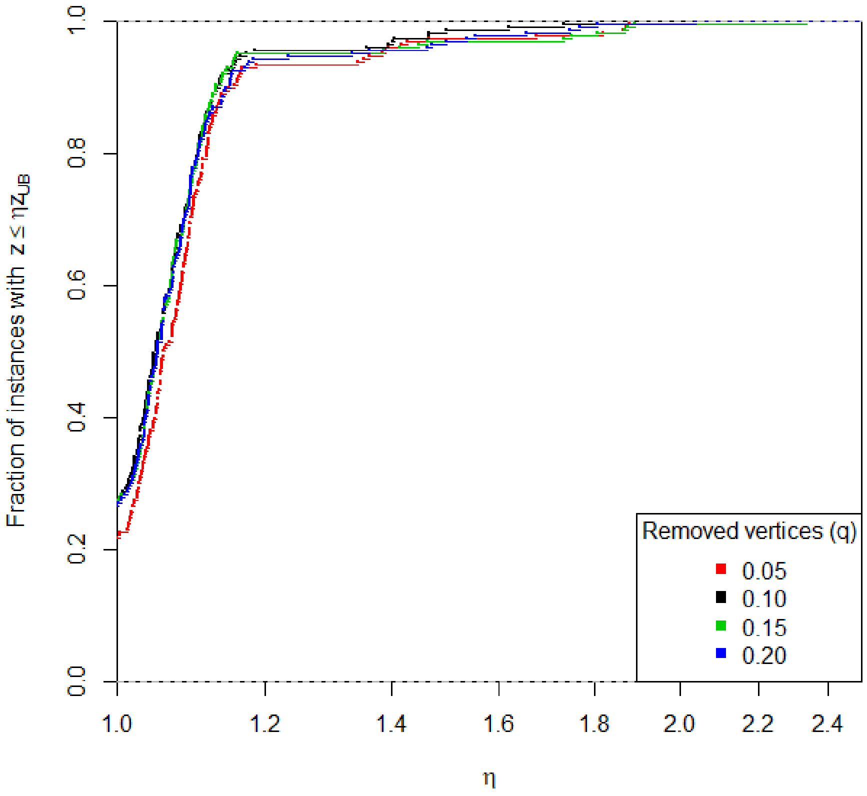

Figure B.13: Performance profiles of the *ALNS* algorithm with different fractions q of removed vertices

ρ q	5	10	20	50	100
0.05	4.39% 49.91%	4.75% 53.99%	3.38% 49.50%	4.11% 52.35%	4.37% 51.10%
0.10	4.44% 53.47%	3.61% 52.77%	2.72% 49.54%	2.51% **48.65%**	3.22% 51.37%
0.15	4.54% 54.60%	2.91% 51.10%	2.50% 50.19%	**2.42%** 49.41%	2.49% 49.53%
0.20	4.69% 57.32%	3.65% 54.77%	3.04% 52.65%	3.46% 54.08%	2.89% 51.56%

Table B.11: Average gap with respect to upper and lower bounds of the results obtained in $n/1\,000$ minutes by the *ALNS* with the Shaw removal heuristics with different fractions of removed vertices (q), chosen more or less randomly (increasing ρ reduces randomness)

heuristics have been assigned the same probability. Table B.11 reports the results: the rows correspond to the values of q and the columns to the values of ρ. The best results with respect to the upper bound are obtained setting $q = 0.15$ and $\rho = 50$, and these are the values that will be employed in all following experiments. However, there is a rather large region in the parameter space that does not differ in a statistically significant way from the chosen tuning.

Figure B.14 shows the performance profiles of the *ALNS* with different values of parameters q and ρ ruling the Shaw removal heuristic. We can observe that the best performance profile is obtained for $q = 0.15$ and $\rho = 50$ (as emphasized in black) and there are no significant differences in the others, in agreement with Table B.11.

Tuning of parameter τ_0. After determining a good value for the specific parameters of each removal heuristic, we have tuned the parameter τ_0 that controls the probability to accept a worsening solution. This parameter rules the balance between the intensification and diversification aspects of the search. We remind

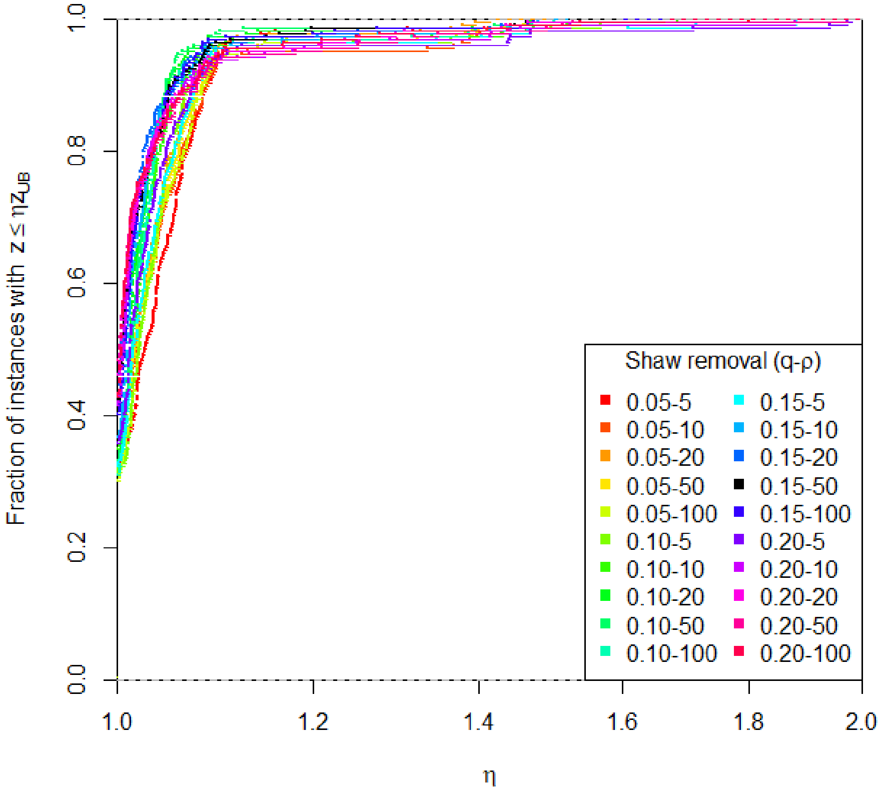

Figure B.14: Performance profiles of the ALNS algorithm with different values of the parameters ruling the Shaw removal (q and ρ)

τ_0	0.00	$\dfrac{0.05}{\ln 2}$	$\dfrac{0.10}{\ln 2}$	$\dfrac{0.15}{\ln 2}$	$\dfrac{0.20}{\ln 2}$
δ_{UB}	**1.50%**	3.36%	10.69%	18.78%	23.10%
δ_{LB}	**46.75%**	49.06%	79.51%	111.97%	128.22%

Table B.12: Average gap with respect to the best known upper and lower bounds of the results obtained by the *ALNS*, with different values of the probability to accept worsening solutions

that the *ALNS* algorithm always accepts an improving solution, whereas it accepts a solution with a relative increase of the total gap equal to δ with a probability equal to $e^{-\delta/\tau_0}$. An intuitive way to describe this mechanism is to say that solutions with a relative gap increase equal to $\tau_0 \ln 2$ have the same probability to be accepted or rejected (50%), solutions with a smaller increase are accepted more often and solutions with a larger increase are more likely to be rejected. By $\tau_0 = 0$ we mean that all nonimproving solutions are rejected. In our experiments, we have considered all values of τ_0 in $\{0.00, 0.05/\ln 2, \ldots, 0.20/\ln 2\}$, so that the relative gap increase accepted with 50% probability ranges from 0 (only improving solutions) to 0.20 (solutions with a gap 1.2 times the current one). The algorithm has been run for $n/1\,000$ minutes, as in the previous experiment, assigning the same probability to each of the four insertion heuristics and of the six removal heuristics. Table B.12 reports, as usual, the average values of δ_{UB} and δ_{LB} obtained by each configuration. The best one is $\tau_0 = 0$, that is, accepting only improving solutions; the quality of the best found result becomes gradually worse as the probability of accepting worse solutions increases. This is a not very common tuning for *ALNS*, but it has actually already been applied to other problems with objective functions characterized by *plateaus*, such as the Resource-constrained Project Scheduling Problem (Muller, 2009). According to Wilcoxon's test with the Bonferroni correction, the difference with respect to all other tunings is statistically significant. In the following, therefore, we have set $\tau_0 = 0$, which also makes it no longer necessary to tune the cooling rate h.

Figure B.15 shows the performance profiles of the ALNS with different values of τ_0 ruling the acceptance of worsening solutions. We can observe that the profile obtained for $\tau_0 = 0.00$ dominates the other ones, with the exception (but only for a small range) of $\tau_0 = 0.05/\ln 2$. This is in agreement with the results of Table B.12.

Computational time of the ALNS *for different classes of instances.* Figure B.16 reports the average computational time required to perform 1 000 *ALNS* iterations on each group of 5 benchmark instances with the same size n, topology (planar, sparse and dense) and value of p ($\ln n$, $\sqrt{n}$ and $n/\ln n$). A least-square interpolating profile is drawn for each class of instances to estimate the dependence of the computational time on the size of the instance. As for the Tabu Search algorithm, denser instances take a longer time, while the effect of the overall number of clusters is in this case unambiguous: more clusters clearly imply more time.

The interpolation suggests that the dependence of the computational time on the number of vertices is between linear and quadratic. This is consistent with the structure of the insertion mechanisms, that are Prim-like $O\left(n^2 + m\right)$ procedures, and of the removal mechanisms, that are $O\left(n\right)$ (worst and random-worst), $O\left(m\right)$ (random) or $O\left(n^2\right)$ (Shaw removal).

Parameter configuration for the adaptive mechanism. Finally, we introduce the adaptive mechanism *AWAP* described in Section 5.3. We increase the overall computational time to $n/100$ minutes, that is 10 times larger than in the previous experiment and equal to the time used in the final test on the Multilevel Tabu Search algorithm. Since this time limit allows to perform millions of iterations on all the instances of the benchmark, we set the number of iterations for each segment to $NS = 1\,000$. This guarantees both the generation of a large number of segments and several applications of the $4 \cdot 6 = 24$ possible combinations of insertion and removal heuristics, in each segment. Setting $\tau_0 = 0$ avoids tuning the scores σ_2 and σ_3, because the corresponding cases never occur: only solutions that improve the best known result are accepted. As for σ_1, this score has been set to 1, so that the ratio s_H/θ_H used to update the probability of each heuristic H is equal to the frequency with which that heuristic has returned an

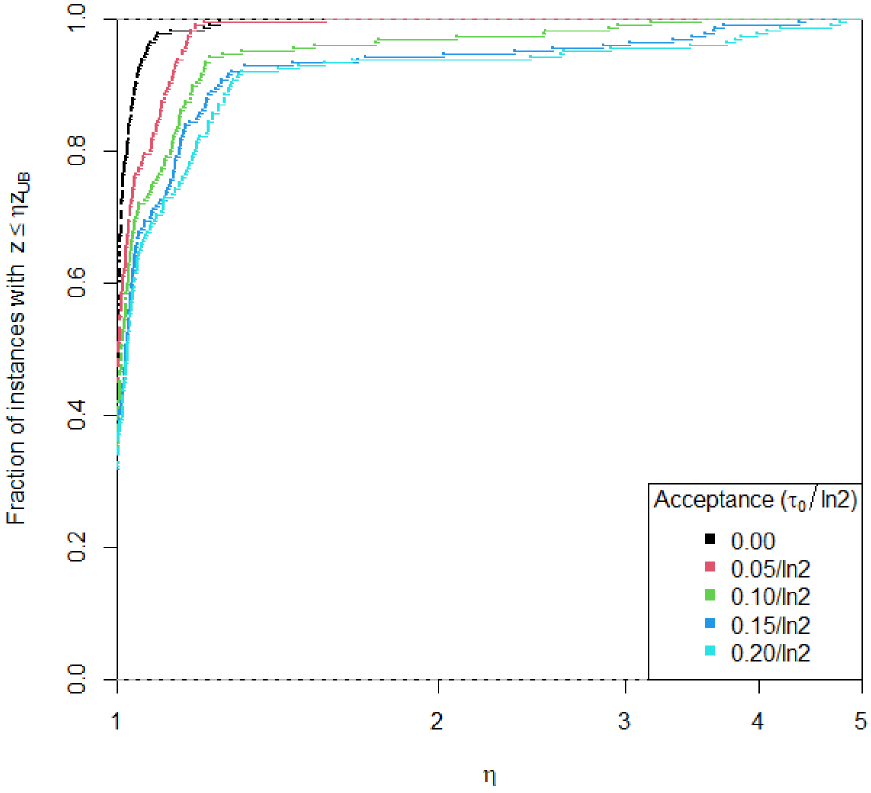

Figure B.15: Performance profiles of the ALNS algorithm with different values of the parameter τ_0 ruling the acceptance of worsening solutions

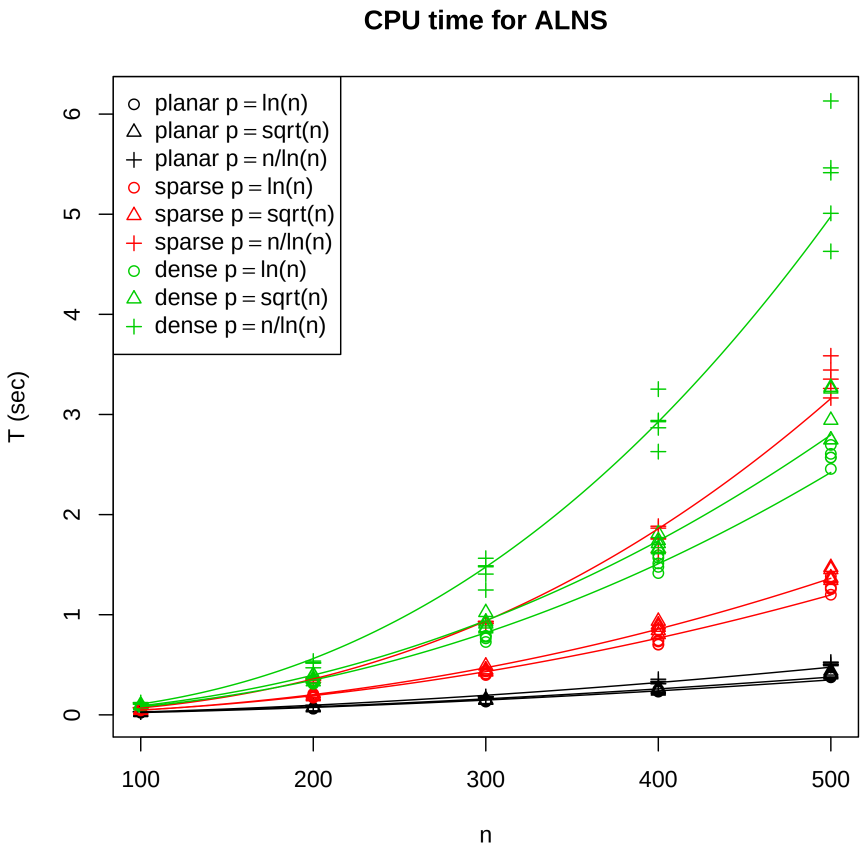

Figure B.16: Average computational time (in seconds) required to perform 1 000 iterations of the *ALNS* algorithm (with equal probability for each combination of insertion and removal procedures) on instances of the nine classes

improved solution in the last segment. The only remaining parameter is the reaction factor, that is set to $r = 0.1$ following Røpke and Pisinger (2006).