On some Optimization Problems on Dynamic Networks

Antonio Napoletano

Dep. of Mathematics and Applications, "R. Caccioppoli"
University of Napoli, "Federico II"

Thesis submitted for the degree of
Doctor of Philosophy

Supervisor:
Professor Paola Festa

December 2017

Abstract

The basic assumption of *re-optimization* consists in the need of efficiently managing huge quantities of data in order to reduce the waste of resources, both in terms of space and time. Re-optimization refers to a series of computational strategies through which new problem instances are tackled analyzing similar, previously solved, problems, reusing existing useful information stored in memory from past computations. Its natural collocation is in the context of dynamic problems, with these latter accounting for a large share of the themes of interest in the multifaceted scenario of combinatorial optimization, with notable regard to recent applications. Dynamic frameworks are topic of research in classical and new problems spanning from routing, scheduling, shortest paths, graph drawing and many others.

Concerning our specific theme of investigation, we focused on the dynamical characteristics of two problems defined on networks: re-optimization of shortest paths and incremental graph drawing. For the former, we proposed a novel exact algorithm based on an auction approach, while for the latter, we introduced a new constrained formulation, *Constrained Incremental Graph Drawing*, and several meta-heuristics based prevalently on Tabu Search and GRASP frameworks.

Moreover, a parallel branch of our research focused on the design of new GRASP algorithms as efficient solution strategies to address further optimization problems. Specifically, in this research thread, will be presented several GRASP approaches devised to tackle intractable problems such as: the Maximum-Cut Clique, p-Center, and Minimum Cost Satisfiability.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

The common threads that link the topics which will be discussed in this thesis are the challenges arising from the study of dynamic networks and the related optimization problems. Particular emphasis will be put on the Shortest Path (SPP) and Graph Drawing Problems (GDP) and their corresponding dynamic versions. It could be superfluous to point out the great importance and the impact which the shortest path problem had in the history of the combinatorial optimization and operation research. In its basic formulation, the objective is to determine the minimum cost path from a given origin node to a destination node in a directed weighted network. One of the reasons of great interest in studying this problem is motivated by the fact that many optimization problems require the solution of the SPP as sub-task. Some examples of these problems are Maximum-Flow Minimum-Cost Problems [1], Vehicle Routing Problems [121], and several other variants of the SPP, spanning from problems on time-dependent networks [96, 97] to general constrained SPP [28, 29, 40]. An immediate extension of the SPP is the Shortest Path Tree Problem (SPTP), where it is required to find all the shortest paths starting from an origin node by considering as destination all the other nodes in the network.

Concerning the purely dynamic aspect of the problem, re-optimizing shortest paths on dynamic networks consists in solving a sequence of shortest path problems, where each problem differs only slightly from the previous one, because the origin node has been changed, some arcs have been removed from the network, or the cost of a subset of arcs has been modified. Each problem could be simply solved from scratch, independently from the previous one, by using either a *label-correcting* or a *label-setting* shortest path algorithm. Nevertheless, a clever way to approach it is to design ad hoc algorithms that efficiently exploit informations resulting from previous computations. This type of problems appears in a wide variety of contexts and application settings, including logistics [72, 128], telecommunications [118, 104], transportation, urban traffic [20, 100], and transit planning [1, 78]. Although the SPP and SPTP together with their variants have been widely studied in literature, the re-optimization version, Re-optimization SPP (R-SPP) and Re-optimization SPTP (R-SPTP), gathered much less scientific efforts over the years. A detailed list of the approaches proposed for the re-optimization of shortest paths is surveyed in [43]. One of the first efficient strategies was proposed by Gallo [55], where the R-SPTP is addressed when either the origin node is changed, or when the cost of only one arc is updated. Another relevant work was introduced by Florian et al. [51], which presented a dual approach for the R-SPTP, when the origin node changes. Later, Ramalingam and Reps [107] elaborated an algorithm for the R-SPTP when the weight of one arc is increased, while Pallottino and Scutellá [101] addressed the problem of recomputing the shortest path tree when a new cost is assigned to a subset of arcs. On the same lines Buriol et al. [13] developed two different algorithms for the R-SPP and R-SPTP to manage the case when the cost of one arc is either increased or decreased. Others proposals to tackle the problem when a single arc or a set of these is updated can be found in Chan and Yang [17], D'Andrea et al. [35], Nannicini et al. [96]. According to best of our knowledge, and considering the list of works just surveyed, several works were proposed to tackle the re-optimization problem when the weight of the arcs changes, while much less were addressed to the case of origin change.

In Chapter 2, the R-SPTP in the case of origin change will be introduced. Moreover, a new approach, proposed in Festa et al. [50], will be presented. The algorithm tries to update an initial optimal solution for a given SPTP in order to obtain a new optimal solution for a new problem which requires to find another shortest path tree starting from a new origin node. The process which leads to the construction of the new solution will act through a sequence of operations, called extensions, by using a strongly polynomial auction algorithm based on the virtual source concept, described in [15].

The second part of this manuscript, Chapter 3, will be focused on the Graph Drawing Problem. Drawing a graph or network plays a role of fundamental importance in many information visualization problems, like models in software, information engineering, project scheduling and many others. For example, flows charts, state transition diagrams, entity-relationships diagrams, and Petri nets are all cases of graph drawing. Conventionally, the nodes of a graph are represented with boxes or circles (containing text information) and the arcs with lines between the nodes.

Graph drawing is an extensively studied area of research, especially in recent years, and a considerable number of works were presented in this direction, refer to Battista et al. [3] for a detailed survey. Nevertheless, the establishment of a uniform criterion to evaluate the quality of a drawing is a much debated topic. Currently, the hypothesis that seems to be the most supported from the scientific community, in order to evaluate the quality of a drawing, is the number of crossing arcs in the layout of the graph.The hardness of achieving a clear graph drawing with this evaluation criteria is easily relatable to the proof of Garey and Johnson, that showed how the minimization of the number of crossing in a graph is a NP-complete problem. As we can observe in Figure 1.1, according to the crossing-minimization criterion, Figure 1.1(b) is considered better than Figure 1.1(a), in fact the correlations between the nodes are clearer and structural information can be retrieved in a simpler way. For our purpose we will consider only hierarchical directed acyclic graphs (HDAG),

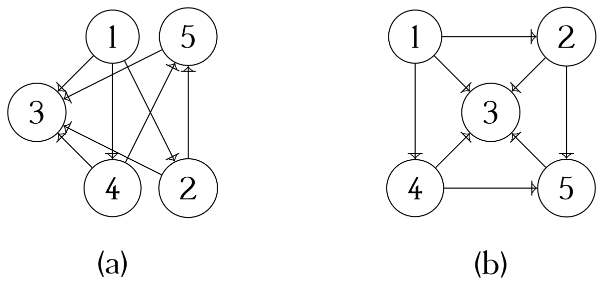


Fig. 1.1 Different representations of the same graph.

anyway this will not cause a loss of generality since each directed acyclic graph (DAG) can be transformed into a HDAG through a sequence of steps as demonstrated in [57]. One of the most established method to build an HDAG, starting from a DAG, consists in disposing the nodes in layers in such a way that all the arcs point in the same direction, after this step, if there are some arcs which connect nodes that are not in contiguous layers, then artificial nodes can be added in order to generate a graph will meet the desired requirements. Even in this case, [57] proves that the problem of minimizing the number of crossing arcs in a HDAG with only two layers is NP-complete. In Figure 1.2 is depicted an application of the method described above.

The dynamic version of the GDP is known in literature with the name of Incremental Graph Drawing Problem (IGDP), it consists in adding a set of new nodes and corresponding arcs to the original HDAG. Referring to the nodes and arcs of the original HDAG as original nodes and original arcs, the addition of the new nodes

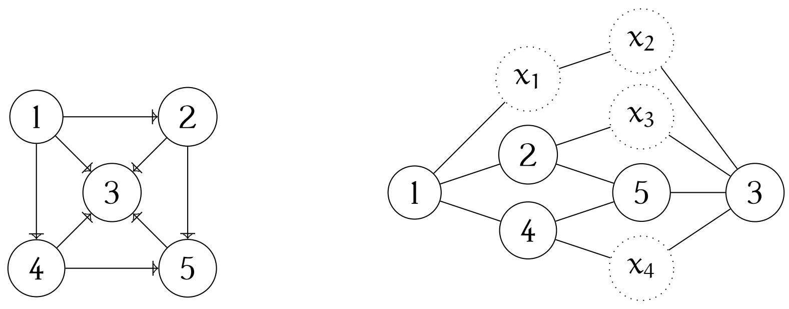


Fig. 1.2 Transformation from DAG to HDAG.

must not alter the relative positions of the original ones during the process of minimization of the number of crossing arcs. As in the case of the SPP, also in this circumstance the static version of the problem has been studied extensively, while the dynamic one received much less attention. In fact, the only works proposed for the IGDP were [82], limited to graph with only two layers , and [115], which considers also graphs with more layers. In Figure 1.3 it is shown an example of IGDP.

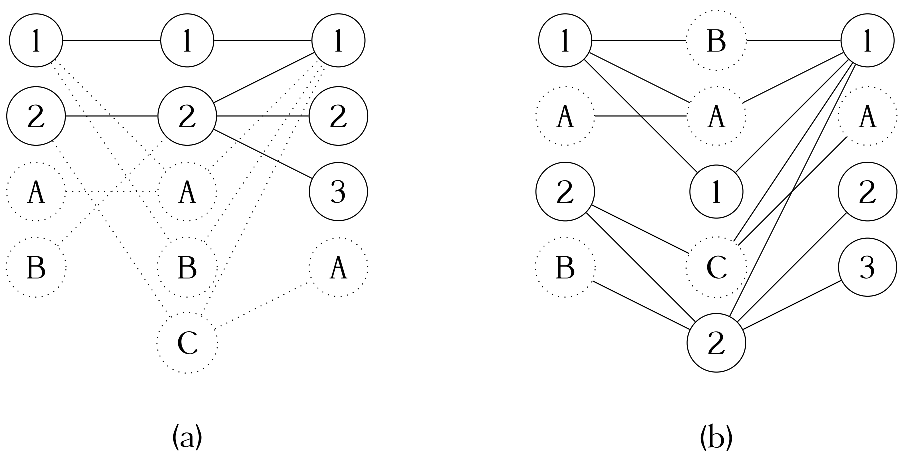


Fig. 1.3 Example of Incremental Graph Drawing Problem. The new nodes introduced in the HDAG contains characters and are depicted dotted, while the original nodes contain numbers. Figure(b) represents the optimal solution after the insertion of the new nodes, the number of crossings is 2.

The problem of layouting dynamic graphs, thus graphs which evolve over the time, introduces an additional aesthetic criterion known as *preservation of the mental map* [92]. Each node in a graph appears with some graphical attributes, like position, color, size, shape and others. The concept of mental map is used in behavioral geography and psychology to describe the process through to which an agent is able to acquire and maintain information, related to the surrounding environment, and how through this he can plan activities such as setting up connections between several points or going along path already traveled. It's clear how several evolution of a graph could lead to a disruption of the agent's mental map. In fact, it is sufficient to consider the example in Figure 1.4 to evaluate how a single evolution of the graph can bring substantial changes to the initial mental map.

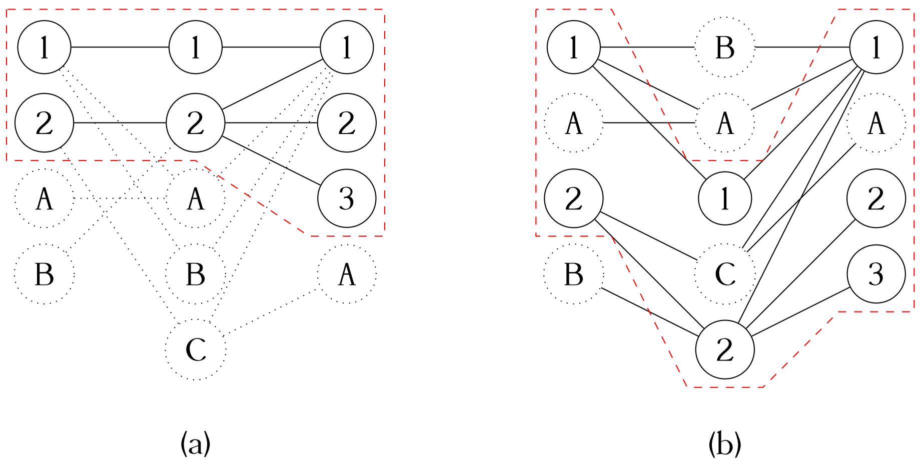


Fig. 1.4 How the mental map changes.

With in mind the compromise between the two aesthetic criteria described above, i.e. minimizing the number of crossing arcs and preserving the mental map, we proposed in [98] a new version of the IGDP, called Constrained-IGDP (C-IGDP), where the incremental version of the problem is enriched by an additional constraint which limits the displacement of the original nodes within a certain range K, where K is defined as *position constraint*. In Figure 1.5 is depicted an example of C-IGDP. The comparison between Figure 1.4 and Figure 1.5 shows how the position constraint considered in the C-IGDP lets the algorithm attain a solution that better preserves the original disposition of the original nodes, and thus the mental map, although the number of arc crossed in the new drawing increases.

For the C-IGDP, we developed several resolution strategies, based on GRASP [39, 47, 48] and Tabu Search frameworks [60–63]. In particular, we designed three constructive phases for the GRASP with a common local search, and a Tabu-like algorithm. The Tabu algorithm adopts a constructive phase based on memory and an improvement phase, which follows the principles of a Tabu Search paradigm. Both the construction and the local search, in this last approach, have the main purpose

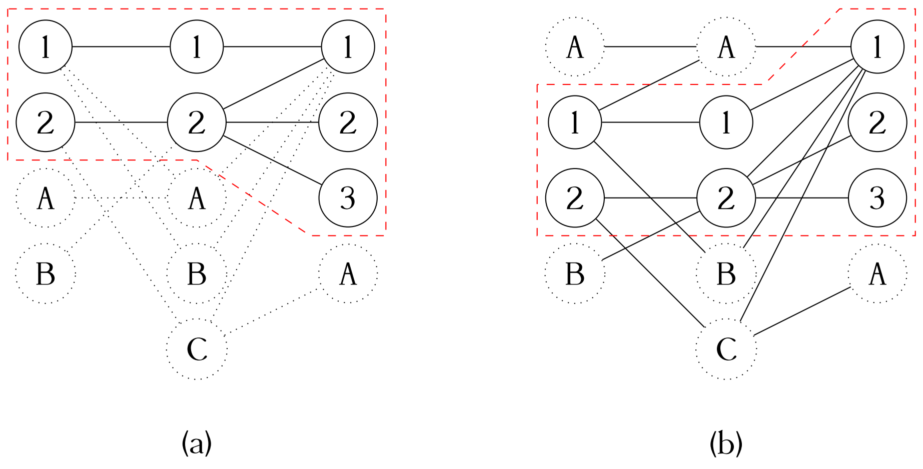


Fig. 1.5 C-IGDP example, the position constraint is given by $K = 1$, then the original nodes can be displaced at most of one position compared to the original one. The number of crossings in the Figure (b) is 7.

of guaranteeing a degree of diversification during the exploration of the solution space. Finally, for all the algorithms a post-optimization phase with `Path-Relinking` [76, 64, 110] has been implemented.

In the third part, Chapter 4, we survey and analyze other optimization problems addressed and resolved with remarkable results by designing ad-hoc algorithms based on the GRASP framework:

- p-Center Problem (p-CP): we hybridized a greedy adaptive constructive phase with a smart and fast two-criteria local search based on the concept *critical node*, [44, 45];
- Minimum Cost Satisfiability Problem (MinCost-SAT): we implemented a GRASP with an experimental stopping rule criterion, based on the probabilistic analysis of the solution values collected by the algorithm during its execution, [38]
- Maximum Cut-Clique Problem (MCCP): for this after the constructive phase we consider an improvement one founded on the *Two Phased Local Search* introduced in [106], [42].

Finally, in the concluding part, Chapter 5, the benefits obtained applying our original methods for the several problems considered will be summarized, and the possible future developments will be discussed.

Chapter 2

Re-optimization of Shortest Paths

In this chapter the mathematical formulations for the SPP and its variants are given, and the best know techniques for these problems will be analyzed . Then, it will be introduced the Re-optimization framework for the SPP with particular attention to the Re-optimization Shortest Path Tree Problem (R-SPTP). For the latter we presented in [41, 43] an ad-hoc algorithm, a smart and fast dual-based algorithm which adopts an auction strategy.

2.1 Mathematical Formulations

All the problems which we will formulate rely on the following notation. Let $G = (V, A)$ be a connected directed graph, where

- $V = \{1, 2, \dots, n\}$ is a set of nodes;
- $A \subseteq \{\, (i, j) \in V \times V \mid i, j \in V \wedge i \neq j \,\}$ is a set of m arcs;
- $c\colon A \to \mathbb{R}^+$ is a function that assigns a non-negative cost c_{ij} to each arc $(i, j) \in A$.

A path P is a sequence of nodes $P = \{i_1, i_2, \dots, i_k\}$ such that $(i_l, i_{l+1}) \in A$, for all $l = 1, \dots, k-1$. If the nodes $i_l, i_2, \dots, i_k$ are distinct, the sequence $P = \{i_1, i_2, \dots, i_k\}$ is called a simple path. The cost $z(P)$ of a path $P = \{i_1, i_2, \dots, i_k\}$ is defined as the sum of the costs of its arcs, that is, $z(P) = \sum_{l=1}^{k-1} c_{i_l i_{l+1}}$. Furthermore, for each $i = 1, \dots, n$, let

- $FS(i) = \{\, j \in V \mid (i, j) \in A \,\}$ be the *forward star* of node i;
- $BS(i) = \{\, j \in V \mid (j, i) \in A \,\}$ be the *backward star* of node i.

2.1.1 Shortest Path Point-to-Point Problem (P2P)

The problem consists in finding a shortest path $P^* = (v_1, v_2, \dots, v_h)$ from a source node $v_1 = s$ to a destination node $v_h = t$, with $s, t \in V$. Introducing m Boolean decision variables, x_{ij}, $\forall\ (i,j) \in A$, such that:

$$x_{ij} = \begin{cases} 1, & \text{if } (i,j) \text{ belongs to } P^*, \\ 0, & \text{otherwise,} \end{cases}$$

the mathematical formulation of the (P2P) problem is the following:

$$\begin{array}{lll} \text{(P2P)} & z = \min \sum\limits_{(i,j) \in A} w_{ij} x_{ij} & \\ \text{subject to:} & & \\ \text{(P2P-1)} & \sum\limits_{j \in BS(i)} x_{ji} - \sum\limits_{j \in FS(i)} x_{ij} = b_i, & \forall i \in V \\ \text{(P2P-2)} & x_{ij} \in \{0, 1\}, & \forall (i,j) \in A, \end{array}$$

with $b_i = -1$ for $i = s$, $b_i = 1$ for $i = t$, and $b_i = 0$ otherwise.

2.1.2 Shortest Path Tree Problem (SPTP)

Given an origin node $r \in V$, the aim of the SPTP is to find a shortest (of minimum cost) path from r to all other nodes in V. The SPTP admits the following linear programming formulation:

$$\begin{array}{lll} \text{(SPTP)} & \min \sum\limits_{(i,j) \in A} c_{ij} x_{ij} & \\ \text{subject to:} & & \\ & \sum\limits_{(j,i) \in BS(i)} x_{ji} - \sum\limits_{(i,j) \in FS(i)} x_{ij} = b_i, & \forall\ i \in V, \\ & x_{ij} \geq 0, & \forall\ (i,j) \in A, \end{array}$$

where x_{ij} is the flow on the arc (i,j) and the right hand side $b_i = -n + 1$, for $i = r$ and $b_i = 1$, for $i \neq r$.

2.1.3 Dual-Shortest Path Tree Problem (D-SPTP)

The linear programming formulation of the dual shortest path tree problem (D-SPTP) for a given origin node r is the following:

$$\text{(D-SPTP)} \quad \max\ (-n+1)\, y_r - \sum_{j \neq r} y_j$$

subject to:

$$y_j - y_i \leq c_{ij}, \qquad \forall\ (i,j) \in A,$$

where y_i is the dual variable associated with the node i, called *potential* of i. Without loss of generality, we can set $y_r = 0$; in this case, the optimal dual solution y_i represents the minimum path cost from r to i.

2.2 An Overview of the Best-Known Approaches

A first successful attempt to solve the SPP was originally proposed by Ford Jr [52], Ford Jr and Fulkerson [53], although the most famous algorithm to solve P2P and SPTP is a labeling method proposed by Dijkstra [32], whose pseudo-code is reported in Algorithm 2.1. Let $s \in V$ be the source node in a graph G, to find a shortest path from s to each other $v \in V$, $i \neq s$, Dijkstra's algorithm maintains and updates for each node $v \in V$:

- `dist[v]`, the distance of v from the source node s;
- `pred[v]`, the predecessor of the node v in the incumbent path from s to v.

In addition, the following sets are used: S and Q, that are the sets of visited and unvisited nodes, respectively. The algorithm starts with an initialization phase (lines 2-5), where the vectors `pred`, `dist` and the sets S and Q are initialized. Afterwards, while set Q is nonempty, the algorithm selects an unvisited node v, relaxes all the edges in $FS(v)$, and insert v in S. The relaxation operation is described in lines 10-12. If the cost function c is non-negative, the algorithm always terminates with the correct shortest path distances stored in `dist[]`, and shortest path tree in `pred[]`. The following theorem holds:

Theorem 1. *If the cost function w is non-negative, then Dijkstra's algorithm visits nodes in non-decreasing order of their distances from the source, and visits each node at most once.*

```
Dijkstra $(G = (V, A), s)$
dist[s] ← 0; pred[s] ← NULL; $S$ ← $\emptyset$;
forall $v \in V \setminus \{s\}$ do
    dist[$v$] ← $+\infty$;
    pred[$v$] ← NULL;
$Q \leftarrow V$;
while $Q \neq \emptyset$ do
    $u$ ← extract_min($Q$);
    $S$ ← $S \cup \{u\}$;
    forall $v \in FS(u)$ do
        relax($u, v$);
        if dist[$v$] > dist[$u$] + $c_{uv}$ then
            dist[$v$] ← dist[$u$] + $c_{uv}$;
            pred[$v$] ← $u$;
```

Fig. 2.1 Dijkstra's algorithm.

In the case of P2P problem, *bidirectional versions* of Dijkstra's algorithm were proposed in [22, 33, 99]. The bidirectional framework is based on the consideration that if s and t are the source and the destination node, respectively, then it is possible to run the algorithm into two opposite directions: the first from node s to t, called the *forward search*, and the latter from t to s, the *backward search*. The backward search operates on the *reverse graph*, obtained from G reversing the direction of each arc in A. The algorithm terminates when the two paths meet.

Hart et al. [70] proposed another labeling method for SPP: an informed search algorithm called A^*. It refines the Dijkstra's method, using a best first paradigm, firstly exploring sub-paths which appear to lead most quickly to the solution. The estimation of the most promising sub-paths is carried out by means of a potential function π_t. Let $\pi_t : V \rightarrow \mathbb{R}^+$ be a non-negative function, giving an estimate on the distance from each node v to t. The A^* search uses a new set L, which contains all the nodes that are relaxed at least once and whose label is not permanent. It selects a node $v \in L$ with the smallest value of $k(v) = d(s, v) + \pi_t(v)$, where $d(s, v)$ is the shortest distance from s to v.

A potential function π_t is defined to be *feasible* if $d_\pi(u, v) = d(u, v) - \pi_t(u) + \pi_t(v)$ is non-negative for each arc $(u, v) \in A$. Goldberg and Harrelson [65] showed that A^* search with a feasible non-negative potential function visits no more nodes than Dijkstra's algorithm. In order to define π_t, in the Euclidean domain, it is possible to use the canonical Euclidean distance to establish a lower bound. Such computation is carried out by means of a method based on the concept of landmarks selection [65] and the triangle inequality.

An alternative and simple approach for the SPP was proposed by Bertsekas, this algorithm is based on the auction strategy which was proposed by the same author for the assignment problem in [8]. Since our algorithmic proposal, which will be discussed in the follow of this chapter, is founded on this strategy we dedicate the following paragraph to describe the auction strategy.

2.2.1 The auction strategy

The auction strategy was firstly proposed by Bertsekas in an unpublished report [8] for the assignment problem, while the standard auction approach for the SPP was developed, by Bertsekas himself, in 1991 [9] and a survey can be found in Chapter 4 of [10]. Any auction algorithm follows a primal-dual approach and consists of the following three basic operations:

- *path extension*;
- *path contraction*;
- *price increase*.

For the SPP, given a graph $G = (V, A)$ as before defined and the origin node $r \in V$, the algorithm maintains a path P starting at r and ending at a node t, called *terminal node* and a set π of *feasible node prices*. The pair (P, π) satisfies the following conditions:

$$\pi_i \leq \pi_j + c_{ij}, \ \forall \ (i, j) \in A, \tag{2.1}$$

$$\pi_i = \pi_j + c_{ij}, \ \forall \ (i, j) \in P. \tag{2.2}$$

Conditions (2.1) are known as Price Feasibility Conditions (PFC) and (2.2) as Complementary Slackness Conditions (CSC). The algorithm starts with any pair (P, π) satisfying conditions (2.1) and (2.2) (for example, $P = \{r\}$ and $\pi_i = 0$, $\forall \ i \in V$) and proceeds in iterations, updating (P, π) into a new pair (P', π') satisfying (2.1) and (2.2). At each iteration, if there is an arc $(t, j) \in FS(t)$ such that $\pi_t = \pi_j + c_{tj}$, then the path P is extended along that arc and j becomes the new terminal node. Otherwise, no extensions are possible and the price of t is increased through the following price increase operation:

$$\pi_t = \min\{c_{tj} + \pi_j \ : \ (t, j) \in FS(t)\}. \tag{2.3}$$

```
Auction ( $G = (V, A)$, C, s )
$P \leftarrow (s)$;
$\pi(\cdot) \leftarrow 0$;
pred($\cdot$) $\leftarrow 0$;
operation $\leftarrow 1$;
switch ( operation ) do
    case 1 do
        i $\leftarrow$ get-terminal-node(P);
        if ( $\pi(i) < \min_{(i,j)\in FS(i)} \{c_{ij} + \pi(j)\}$ ) then
            operation $\leftarrow 2$;
        else
            operation $\leftarrow 3$;
    case 2 do                                              /* contract path */
        $\pi(i) \leftarrow \min_{(i,j)\in FS(i)} \{c_{ij} + \pi(j)\}$ );
        if ( $i \neq s$ ) then
            $P \leftarrow P \setminus \{i\}$;
        operation $\leftarrow 1$;
    case 3 do                                              /* extend path */
        $j_i \leftarrow \operatorname{argmin}_{(i,j)\in FS(i)} \{c_{ij} + \pi(j)\}$;
        pred($j_i$) $\leftarrow i$;
        if ( All nodes have been terminal at least once ) then
            return pred;
        operation $\leftarrow 1$;
```

Fig. 2.2 Auction algorithm for the SPTP.

If the last arc of P violates condition (2.2) due to a price increase operation, P is contracted by deleting the arc. The first time a node i becomes the terminal node, path P is a shortest path from r to i, its cost is given by $\pi_r - \pi_i$, and the *predecessor node* of node i in P is stored to define the primal solution, i.e., the shortest path tree. The algorithm terminates as soon as all nodes have become the terminal node of the path P at least once. It is easy to verify that the prices have a natural interpretation as the opposite of dual variables. The auction algorithm starts with a feasible pair (P, π) satisfying CSC and proceeds in iterations, transforming (P, π) into another pair satisfying (2.1) and (2.2). Since the opposite of node prices can be used as node dual potentials, price feasibility is equivalent to dual feasibility. At each iteration, a dual feasible solution and a primal unfeasible solution (implicitly defined by the predecessors) are available for which CSC hold. Therefore, the algorithm either constructs a new primal solution (not necessarily feasible) or a new dual feasible solution, until a primal feasible (and hence also optimal) is obtained. The typical iteration of the standard auction algorithm for the SPTP is reported in Figure 2.2.

2.3 Re-optimization Framework

In the context of SPP, previous information can be reused while tackling a problem which differs only slightly from another SPP previously solved. This occurrence can happen with one of the following changes in the network:

- the origin node has been changed;
- some nodes have been added or removed;
- some arcs have been added or removed;
- some arcs cost have been increased or decreased.

This problem can be addressed as a shortest path re-optimization problem [41], which consists in solving a sequence of shortest path problems, where the k^{th} problem marginally differs from the $(k-1)^{th}$ one.

In the first case, we say that there was a root change from the $(k-1)^{th}$ problem to the k^{th} problem, in the remaining cases we say that the graph is dynamic. Moreover, for what concerns problems on dynamic graphs, they can be classified according to the type of changes that can occur on the network. A dynamic graph is said to be fully dynamic if both insertion and deletion of either edges or nodes are allowed. In Figure 2.3, a diagram of the interplays among all possible cases of shortest path re-optimization is depicted. The root change re-optimization problem does not admit further sub-cases. For what concerns dynamic graphs, the problems can be classified in two different branches, depending on whether the changes involve nodes or arcs. The only possible changes involving a node are addition or removal. On the other hand, the arcs can be either added/removed or their cost can be increased/decreased. It is worthy to note that the case of changes involving nodes implies also arcs insertion or deletion. Without loss of generality, we can consider the input graph $G = (V, A)$ as a complete graph. Indeed, if G is not complete, for each pair of nodes i and j, such that $(i, j) \notin A$, it can be always added a dummy arc from i to j with $c_{ij} = +\infty$. This operation allows the following considerations:

- arc removal in the graph can be seen as a special case of arc cost increasing. If an arc (i, j) is deleted, then the cost c_{ij} increases to $+\infty$.
- Arc insertion can be seen as a special case of arc cost decreasing. If an arc (i, j), must be inserted, then the cost c_{ij} is decreased from $+\infty$ to the new cost k.

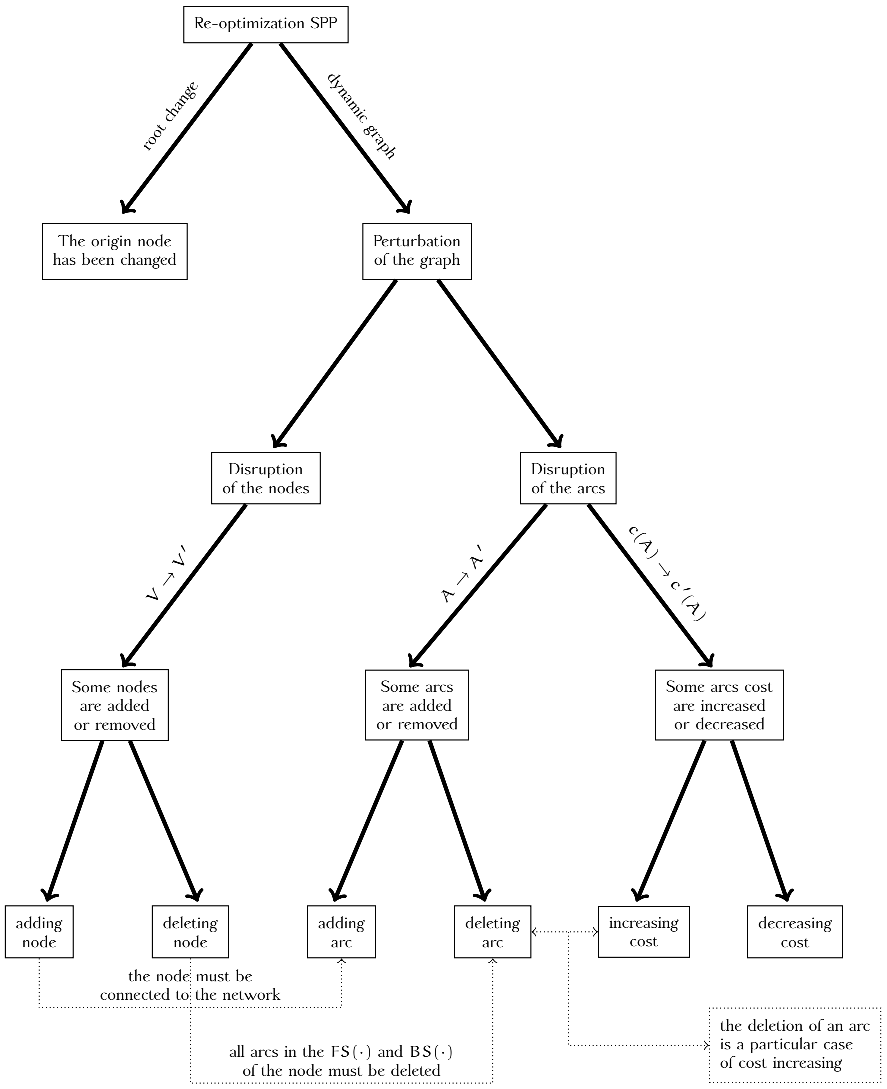

Re-optimization SPP
root change
dynamic graph
The origin node has been changed
Perturbation of the graph
Disruption of the nodes
Disruption of the arcs
$V \to V'$
$A \to A'$
$c(A) \to c'(A)$
Some nodes are added or removed
Some arcs are added or removed
Some arcs cost are increased or decreased
adding node
deleting node
adding arc
deleting arc
increasing cost
decreasing cost
the node must be connected to the network
all arcs in the FS(·) and BS(·) of the node must be deleted
the deletion of an arc is a particular case of cost increasing

Fig. 2.3 Re-optimization problems hierarchy.

2.3.1 Root-change

The purpose of this paragraph is to show how, in the case of root change for the SPTP problem, it is possible to obtain a well performing algorithm making wise use of the information stored in a SPTP previously computed. Such result relies on some remarkable theoretical properties proven by Gallo [55], starting from the assumption that a single root shortest path tree problem has been solved. Let $G = (V, A)$ be a complete directed graph, and let T_r be a shortest path tree rooted at node r, i.e., a tree that contains a shortest path from r to each node $v \in V$, $v \neq r$. Let s be a node of V, $s \neq r$, and T_s be a SPTP rooted at node s. The following propositions show how the knowledge of T_r provides useful informations on T_s. Let $T_r(h)$ denote the subtree of T_r which contains node h together with all its descendants, then

Proposition 1. $T_r(s) \subseteq T_s$ *and* $d(s, j) = d(r, j) - d(r, s)$, *for any* $j \in T_r(s)$.

Henceforth, the paths contained in the subtree of T_r rooted in s still remain optimal shortest paths from s to its descendants. This result shows how a wise handling of the old solution is likely to be the most efficient strategy, since -especially when the new root s is close to r - a consistent part of the previously optimal tree T_r will remain optimal.

As formerly stated, beyond the theoretical insight given by Proposition 1, the information provided by T_r can be employed in order to reduce the computational time needed to solve the new SPTP problem, improving the classical Dial's implementation of Dijkstra's Algorithm (DDA) [30, 31]. In Dial's implementation, in fact, one of the most time consuming tasks consists in the identification of the minimum temporary node cost, due to the high number of comparisons to be performed. This number of comparisons strongly depends on the maximum cost among the arcs, $c_{max} = \max_{(i,j)\in A} c_{ij}$. Propositions 2 and 3 show how to reduce c_{max} without changing the sets of feasible and optimal solutions.

Let $\pi_1, \pi_2, \ldots, \pi_n$ be integer numbers such that

$$l_{ij} = c_{ij} + \pi_i - \pi_j \geq 0 \quad \forall (i, j) \in A. \tag{2.4}$$

Proposition 2. *The problem of finding the SPT from* s *with arc lengths* c_{ij} *is equivalent to the problem of finding the SPT with arc lengths* l_{ij} *given by* (2.4).

Proof. Let P be a generic path from a node k to a node h of $G = (V, A)$, and let $C(P)$ and $L(P)$ be the lengths of P pertaining to the length functions c and l, respectively.

Then, it holds that

$$L(P) = \sum_{(i,j)\in P} l_{ij} = \sum_{(i,j)\in P} (c_{ij} + \pi_i - \pi_j) = C(P) + (\pi_k - \pi_h). \tag{2.5}$$

Henceforth, the lengths $C(P)$ and $L(P)$ only differ by a constant depending only on the source and the destination nodes of the path.

Ultimately, a shortest path tree T_r with respect to the arc length c_{ij} will remain optimal with arc length l_{ij}. □

A straightforward consequence of Proposition 2 is that lengths c_{ij} can be replaced by lengths l_{ij}, and once the new shortest length $d'(s,h)$, $h \in V$, is found, the value $d(s,h)$ can be obtained as follows:

$$d(s,h) = d'(s,h) - \pi_s + \pi_h. \tag{2.6}$$

The results outlined above suggest that an appropriate choice of the integers π_i might decrease the distance from the source to the farthest node, and thus also the computation time required by DDA. Indeed, when selecting $\pi_j = d(s,j)$ one has $d'(s,j) = 0$, for all $j \in V$, thus obtaining the validity of the following proposition:

Proposition 3. *Let be* $\pi_j = d(r,j)$, *for all* $j \in T_r$, *and let be the arc lengths defined as in* (2.4). *Let* h *be one of the farthest node from the origin* s, *then:*

$$d'(s,h) = d(r,s) + d(s,r). \tag{2.7}$$

From Proposition 3 it follows that if nodes r and s are close enough, the computational effort required by DDA can be strongly reduced by a cost modification of type (2.4), with the vector π given by $\pi_j = d(r,j)$, for all $j \in T_r$.

As reported in Gallo [55], in terms of Linear Programming, such new costs correspond to the reduced costs relative to a dual feasible, but primal unfeasible, basis. This interpretation of the vector $(\pi_1, \pi_2, \ldots, \pi_n)$ as a dual feasible solution for the Shortest Path Problem is due to Bazaraa and Langley [4].

Deriving a cost reduction in a similar fashion, in 1982 Gallo and Pallottino [56] devised an algorithm which outperforms both the one proposed in [55] and classical from scratch optimization techniques. This algorithm refines the classical label setting paradigm by partitioning the nodes of the graph in three distinct sets: NT, NP, and NQ. As in a classical label setting algorithm, NT and NP are the set of nodes whose labels are temporary and permanent, respectively. While, the nodes in NQ

are those nodes such that $d(s,v)$=$d(s,p(v))$. This property ensures that such nodes can be inserted straightaway in NP without further comparisons, thus speeding up the execution of the algorithm. In [56], it has been noted how in any re-optimization problem instances a large share of nodes of V is likely to be found in NQ.

The observations made by Gallo are the starting point of the work of Florian et al. [51]: the shortest path tree rooted at node r is an optimal solution to a corresponding linear program, but when a successive new source s is considered, the previous tree is a dual feasible and primal infeasible solution for the new problem. The approach proposed by Florian et al. [51] consists in the adaptation of the dual simplex method to compute the shortest paths from the new root s. It is shown in [51] that the proposed algorithm runs at most in $O(n^2)$. In order to evaluate the performances of their method, the authors tested their code on graphs representing the regional roads of the cities of Vancouver and Winnipeg, as in [55].

The experimental evaluations proposed in these works show how Gallo [55] is slightly better performing than Dijkstra's from scratch technique, meanwhile Gallo and Pallottino [56] further improves the previous results. Although, among all, the algorithm proposed by Florian et al. [51] appears to outperform the other competitive methods.

2.4 Re-optimization Auction Algorithmic Scheme: R-Auction

Given an optimal solution for the SPTP from node r to all other nodes, described by the tree $\overset{r}{T} = (\overset{r}{V}, \overset{r}{A})$, the predecessor vector $p_r \in V^{|V|}$ and the distance vector $d_r \in \mathbb{R}^{|V|}$, where $d_r(i)$ is the shortest distance from the origin node r to i, the proposed auction algorithm operates on the original graph, by considering the optimal reduced cost $\bar{c}_{ij} = c_{ij} + [d_r(i) - d_r(j)]$ associated with each arc $(i,j) \in A$.

In the case of origin change, that is, the current origin node r is replaced by a new one denoted by s, at the beginning of the algorithm the arc connecting the new origin s to its predecessor is removed and the subtree $\overset{s}{T} = (\overset{s}{V}, \overset{s}{A})$ rooted at s is taken as the initial portion of the new optimal solution. This produces a cut (V', V'') in the graph, where $V' = \overset{s}{V}$ and $V'' = \overset{r}{V} \backslash V'$.

We use two vectors $p_s \in V^{|V|}$ and $d_s \in \mathbb{R}^{|V|}$ to keep track of the predecessor nodes and the distance of each node in the optimal solution under construction, and two vectors π and w to store the potential and the weight for each node as described in [15]. In particular, at any iteration k, $w_k(i)$ is 0 if the node i has not been involved yet, or it expresses the shortest distance from the source node s to i plus the current minimum value of the function $c + \pi$ on $FS(i)$. Finally, a priority queue Q is used to store the nodes that during the algorithm become virtual source, according to the concept expressed in [15]. The complete auction algorithmic scheme for the R-SPTP is described in Figure 2.4. In lines 2-4 the above mentioned data structures are initialized and the reduced costs are computed in lines 5-6. In 7-13 the subtree rooted in $\overset{s}{T}$ is extracted from $\overset{r}{T}$, and for each node in $\overset{s}{V}$, traversing $\overset{s}{T}$ in post-order, the relative fields in d_s, p_s, π, w are updated. Finally, each node in $\overset{s}{V}$ with $FS(\cdot) \neq \emptyset$ is added in the priority queue Q used to store the nodes in non-decreasing order according to w. After `step 0` (lines 14-33), the algorithm performs a sequence of extensions from the actual solution $\overset{s}{T}$ to a node $x \in V''$. Once the extension is performed, the subtree rooted in x is added to $\overset{s}{T}$ and extracted from $\overset{r}{T}$. The process of the extension terminates when $\overset{r}{T}$ becomes empty.

An extension is performed in line 16 applying the virtual source algorithm [15], whose pseudo-code is described in Figure 2.5. This algorithm is based on a primal-dual approach and improves both the convergence and the complexity of the best known auction-like algorithm. It uses a virtual source concept based on the following consideration: when a node i is visited for the first time by any algorithm which preserves the dual feasibility conditions, then the shortest path from the source node to i is found. Therefore, the shortest path from the source to the remaining nodes can be computed by considering i as a "virtual"source. The virtual source method is very efficient, since it does not require the execution of the path contraction phase. Figure 2.6 and Figure 2.7 describe the main operations performed by our auction approach to determine the values of the vectors p_s, d_s, π, w for the first obtained sub-tree and the general sub-tree determined after the path extension phase.

2.4.1 Correctness of the `R-Auction`

Let $G = (V, A)$ be a directed graph and let $P^1, P^2, \dots, P^q$ denote a sequence of SPTP problems. We assume that the generic problem $P^{(k-1)}$, $2 \leq k \leq q$ is aimed at finding a shortest path tree rooted at r and we denote with $\overset{r}{T} = (\overset{r}{V}, \overset{r}{A})$ an optimal solution for the problem $P^{(k-1)}$, where:

- $\overset{r}{V} = V$, is the set of nodes in $\overset{r}{T}$;
- $\overset{r}{A} \subseteq A$, is the set of arcs in $\overset{r}{T}$.

The P^k problem, instead, requires to find a shortest path tree on G rooted at s, with $s \neq r$.

```
R-Auction ( $G = (V, A)$, $C$, $d_r(\cdot)$, $p_r(\cdot)$, $s$, $T_r$ )
$\pi(\cdot) \leftarrow w(\cdot) = 0$;
$d_s(\cdot) \leftarrow +\infty$;
$p_s(\cdot) \leftarrow p_r(\cdot)$;
foreach ( $(i, j) \in A$ ) do
    $\overline{c}_{ij} \leftarrow c_{ij} + (d_r(i) - d_r(j))$;
                         /* *********************** step-0 ********************* */
$T_s \leftarrow$ extractSubtree( $s$, $T_r$ );
$T_r \leftarrow T_r \setminus T_s$;
$p_s(s) \leftarrow 0$;
foreach ( $i \in T_s$ ) do                                   /* in post-order visit */
    set-tree-first( $T_s$, $s$, $i$, $d_r$, $\pi$, $w$, $\overline{C}$);
    remove-BS( $T_s$, $i$ );
    enqueue ( $Q$, $i$ );                                              /* if $FS(i) \neq \emptyset$ */
                         /* ******************** end step-0 ******************** */
operation $\leftarrow 1$;
while (true) do
    switch ( operation ) do
        case 1 do
            $(i, j) \leftarrow$ find-extension( $T_s$, $T_r$, $Q$, $\overline{C}$ )      /* extension from $T_s$ to $T_r$ */
             operation $\leftarrow 2$;
        case 2 do
            $T_j \leftarrow$ extractSubtree( $j$, $T_r$ );
            $T_s \leftarrow T_s \cup (i, j) \cup T_j$;
            $T_r \leftarrow T_r \setminus T_j$;
            operation $\leftarrow 3$;
        case 3 do
            $p_s(j) \leftarrow i$;
            $d_s(j) \leftarrow d_s(i) + c_{ij}$;
            foreach ( $q \in T_s$ ) do                       /* in post-order visit */
                set-tree-general( $T_s$, $i$, $j$, $q$, $d_r$, $\pi$, $w$, $d_s$, $\overline{C}$);
                remove-BS( $T_s$, $q$ );
                enqueue ( $Q$, $q$ );                                  /* if $FS(i) \neq \emptyset$ */
            operation $\leftarrow 4$;
        case 4 do
            if ( $T_r = \emptyset$ ) then
                return $T_s$.
            operation $\leftarrow 1$;
```

Fig. 2.4 Auction algorithm for the R-SPTP.

```
find-extension ( $T_s$, $T_r$, $Q$, $\overline{C}$ )
operation ← 1;
switch (operation) do
    case 1 do
        if ( $w(q) = +\infty \ \forall q \in Q$ ) then
            return NIL;
        $i$ ← select-min( $Q$ );
    case 2 do
        if ( $\pi(j) = +\infty \ \forall j \in FS(i)$ or $FS(i) = \emptyset$ ) then
            $\pi(i) = w(i) = +\infty$;
            if ($i \in Q$) then
                update($Q, i$);
            operation ← 1;
        if ( $\pi(i) < \min_{(i,j) \in FS(i)} \{\overline{c}_{ij} + \pi(j)\}$ ) then
            oldprice ← $\pi(i)$;
            $\pi(i) \leftarrow \min_{(i,j) \in FS(i)} \{\overline{c}_{ij} + \pi(j)\}$;
            $w(i) \leftarrow w(i) + (\pi(i) -$ oldprice$)$;
            update($Q, i$);
            operation ← 1;
            break;
        else
            $j_i \leftarrow \operatorname{argmin}_{(i,j) \in FS(i)} \{\overline{c}_{ij} + \pi(j)\}$;
            $p_s(j_i) \leftarrow i$;
            $d_s(j_i) \leftarrow w(i)$;
            $\pi(j_i) \leftarrow \min_{(j_i,q) \in FS(j_i)} \{\overline{c}_{j_i q} + \pi(q)\}$;
                                        /* if $FS(j_i) = \emptyset$ then $\pi(j_i) = +\infty$ */;
            $w(j_i) \leftarrow \pi(j_i)$;
            if ( $\pi(j_i) \neq +\infty$ ) then
                insert ($Q, i$ );
            return ($i, j_i$);
```

Fig. 2.5 General scheme of the function that performs an extension in R-Auction, based on the virtual source algorithm.

```
set-tree-first( $T_s$, $s$, $i$, $d_r$, $\pi$, $w$, $\overline{C}$)
$d_s(i) \leftarrow (d_r(i) - d_r(s)) + d_s(s)$;
$\pi(i) \leftarrow \min_{(i,j) \in FS(i)} \{\overline{c}_{ij} + \pi(j)\}$;
$w(i) \leftarrow w(s) + \pi(i)$;
```

Fig. 2.6 Procedure set-tree-first of R-Auction which sets the information of the first fragment of the optimal solution.

```
set-tree-general( $T_s$, i, j, q, $d_r$, $\pi$, $w$, $d_s$, $\overline{C}$)
$d_s(q) \leftarrow (d_r(q) - d_r(j)) + d_s(j)$;
$\pi(q) \leftarrow \min_{(q,t)\in FS(q)} \{\overline{c}_{qt} + \pi(t)\}$;
$w(q) \leftarrow w(i) + \pi(q)$;
```

Fig. 2.7 Procedure set-tree-general of R-Auction which sets the information of the generic sub-tree added to the current partial solution.

Let $\overset{s}{T}_0, \overset{s}{T}_1, \dots, \overset{s}{T}_m$ be the sequence of sub-trees built during the execution of the R-Auction, for determining an optimal solution to the problem P^k, starting from $\overset{r}{T} = (\overset{r}{V}, \overset{r}{A})$.

In particular, at the initial step (step-0), R-Auction constructs the portion of the optimal solution $\overset{s}{T}_0$. At each successive step, step-1, ..., step-m, the solution under construction will be enlarged through a sequence of extensions $(i_1, j_1), (i_2, j_2), \dots, (i_m, j_m)$ with $i_q \in \overset{s}{V}_q$ and $j_q \in V \setminus \overset{s}{V}_q$, for $q = 1, \dots, m$.

The extensions will connect a node belonging to the current partial solution to a node not yet included in the solution. Once an extension (i_q, j_q) has been carried out, the node j_q reached and the corresponding sub-tree T_{j_q} will be integrated in the current partial solution.

To prove the correctness of R-Auction, we will show that the operations executed by R-Auction are equivalent to those performed by the auction algorithm, in the sense that the extensions $(i_1, j_1), \dots, (i_m, j_m)$ produced by R-Auction starting from the solutions $\overset{s}{T}_0, \dots, \overset{s}{T}_{m-1}$ are the same produced by the classical auction algorithm (Figure 2.2) starting from the paths $P_0, \dots, P_{m-1}$.

In order to prove formally the correctness of R-Auction, it is useful to introduce the following definitions and notations:

- p_r and d_r represent the predecessors and distances vectors corresponding to the optimal solution of the problem $P^{(k-1)}$, respectively;
- $(\pi_0, w_0), (\pi_1, w_1), \dots, (\pi_m, w_m)$ denote the sequence of potentials and weights for each nodes, computed by the R-Auction;
- $G_0 = (V_0, A_0), G_1 = (V_1, A_1), \dots, G_m = (V_m, A_m)$ represent the sub-graphs induced by $\overset{s}{T}_0, \overset{s}{T}_1, \dots, \overset{s}{T}_m$;
- $\overline{C}$ is the matrix of the reduced costs computed at lines 5-6 in Figure 2.4.

To prove the correctness of R-Auction, we rely on the following theorem.

Theorem 2. *Suppose that* `R-Auction` *and the classic auction algorithm (referred in sequel to as* `Auction`*) are running on the same input graph* $G = (V, A)$*, with cost function* $\bar{c}$ *for the former and* c *for the latter. At the end of the* $(k-1)^{th}$ *extension, let* $\bar{\pi}_k$ *be the potential vector generated by* `Auction` *and* (π_k, w_k) *the potential and weight vectors generated* `R-Auction`*, respectively. Then, during the* k^{th} *extension the following conditions hold:*

$$\pi_k(i) = \bar{\pi}_k(i), \qquad \forall\, i \in \overset{r}{V}_k; \tag{2.8}$$

$$\pi_k(i) = \bar{\pi}_k(i) + [d_r(i) - d_r(j_i)], \qquad \textit{for}\ (i, j_i) = (i_k, j_k); \tag{2.9}$$

$$w_k(i) = d_s(i) + \bar{\pi}_k(i), \qquad \textit{for}\ (i, j_i) = (i_k, j_k), \tag{2.10}$$

where $j_i = \underset{(i,j) \in FS(i)}{\operatorname{argmin}} \{\bar{c}_{ij} + \pi_k(j)\}$.

Proof. We prove the thesis by induction on the number of extensions produced by the algorithms. At `step-0` (before the first extension), let $\overset{s}{T}_0 = (\overset{s}{V}_0, \overset{s}{A}_0)$ be the sub-tree extracted from $\overset{r}{T}$, where:

- $\overset{s}{V}_0 \subset \overset{r}{V}$;
- $\overset{s}{A}_0 \subset \overset{r}{A}$.

This extraction produces a cut in the tree $\overset{r}{T}$, generating two sub-trees: $\overset{s}{T}_0$ and a new tree $\overset{r}{T}_0 = (\overset{r}{V}_0, \overset{r}{A}_0)$, where:

- $\overset{r}{V}_0 = \overset{r}{V} \setminus \overset{s}{V}_0$;
- $\overset{r}{A}_0 = \overset{r}{A} \setminus \overset{s}{A}_0$;

Since $\overset{s}{T}_0$ is extracted from an optimal shortest path tree, it represents an optimal solution to SPTP defined on the sub-graph G_0 of G induced by $\overset{s}{T}_0$. Then, it is possible to show that there is a sequence of iterations of the algorithm `Auction`, which produces the same sub-solution starting from s (this solution is stored in the vector `pred` in Figure 2.2).

At `step-1` (first extension), we suppose that $P_0 = (s, \ldots, i_1)$ in `Auction`, then the algorithm will try to extend P from i_1. In more detail, the possible scenarios are as follows:

$$\text{if } \bar{\pi}_1(i_1) = \min_{(i_1,j_1) in FS(i_1)} \{c_{i_1 j_1} + \bar{\pi}_1(j_1)\}, \text{ then } j_1 = \underset{(i_1,j_1) in FS(i_1)}{\operatorname{argmin}} \{c_{i_1 j_1} + \bar{\pi}_1(j_1)\},$$

and path P is extended through the arc (i_1, j_1) (see lines 18-19 in Figure 2.2). At the same time in R-Auction:

$$\text{if } \pi_1(i_1) = \min_{(i_1,j_1) in FS(i_1)} \{\bar{c}_{i_1 j_1} + \pi_1(j_1)\}, \text{ then } j_1 = \operatorname*{argmin}_{(i_1,j_1) in FS(i_1)} \{\bar{c}_{i_1 j_1} + \pi_1(j_1)\},$$

and $\overset{s}{T}_0$ is extended through the arc (i_1, j_1) (see line 19 in Figure 2.4). Thus, for each node $v \in \overset{r}{V}_0$, it results that:

$$\begin{aligned} &\bar{\pi}_1(v) = \bar{\pi}_0(v) && \text{in Auction;} \\ &\pi_1(v) = \pi_0(v) && \text{in R-Auction.} \end{aligned}$$

Given that $\bar{\pi}_0(v) = 0$ and $\pi_0(v) = 0$, then for step-1 condition 2.8 is satisfied. Concerning i_1, it holds that:

$$\begin{aligned} &\bar{\pi}_1(i_1) = c_{i_1 j_1} + \bar{\pi}_1(j_1) && \text{in Auction;} \\ &\pi_1(i_1) = \bar{c}_{i_1 j_1} + \pi_1(j_1) && \text{in R-Auction;} \\ &w_1(i_1) = w_1(i_1) + (\pi_1(i_1) - \pi_0(i_1)) && \text{in R-Auction.} \end{aligned}$$

Moreover, since $j_1 \in \overset{r}{V}_0$, it follows that $\bar{\pi}_1(j_1) = \pi_1(j_1) = 0$ and by substituting, we obtain that:

$$\begin{aligned} &\bar{\pi}_1(i_1) = c_{i_1 j_1} && \text{in Auction;} \\ &\pi_1(i_1) = \bar{c}_{i_1 j_1} && \text{in R-Auction;} \\ &w_1(i_1) = w_1(i_1) + (\pi_1(i_1) - \pi_0(i_1)) && \text{in R-Auction.} \end{aligned}$$

Since $\bar{c}_{i_1 j_1} = c_{ij} + [d_r(i_1) - d_r(j_1)]$ (lines 5-6 in Figure 2.4), it follows that:

$$\pi_1(i_1) = c_{i_1 j_1} + (d_r(i_1) - d_r(j_1)) \qquad \text{in R-Auction,}$$

then also condition 2.9 is verified:

$$\pi_1(i_1) = \bar{\pi}_1(i_1) + (d_r(i_1) - d_r(j_1)).$$

Concerning condition 2.10, for $\overset{s}{T}_0$,

$$\begin{aligned} w_1(i_1) &= w_1(i_1) + (\pi_1(i_1) - \pi_0(i_1)) \\ &= d_s(i_1) + c_{i_1 j_1} \qquad (2.11) \\ &= d_s(i_1) + \overline{\pi}_1(i_1), \end{aligned}$$

where equation(2.11) is true by the induction hypothesis (as stated and proved in [14, 15]).

After the extension, the algorithm `R-Auction` inserts in the partial solution the arc (i_1, j_1) and the sub-tree rooted in j_1, obtaining the partial solution $\overset{s}{T}_1 = (\overset{s}{V}_1, \overset{s}{A}_1)$. $\overset{s}{T}_1$ is an optimal solution to SPTP defined on the sub-graph G_1 of G induced by $\overset{s}{T}_1$.

At this point, $\overset{s}{T}_1$ is extracted from $\overset{r}{T}_0$, obtaining a new sub-tree $\overset{r}{T}_1 = (\overset{r}{V}_1, \overset{r}{A}_1)$, where:

- $\overset{r}{V}_1 = \overset{r}{V}_0 \setminus \overset{s}{V}_1$;
- $\overset{r}{A}_1 = \overset{r}{A}_0 \setminus \overset{s}{A}_1$.

At `step-k` (k^{th} extension), let $\overset{s}{T}_{k-1} = (\overset{s}{V}_{k-1}, \overset{s}{A}_{k-1})$ be the portion of the current optimal solution to the SPTP defined on the sub-graph G_{k-1} of G induced by $\overset{s}{T}_{k-1}$, while $\overset{r}{T}_{k-1} = (\overset{r}{V}_{k-1}, \overset{r}{A}_{k-1})$.

Thus, there is a sequence of iterations of the algorithm `Auction` which produces the same sub-solution starting from s, and, looking at the pseudocode in Figure 2.2, this solution is stored in the vector `pred`. Suppose that $P_{k-1} = (s, \dots, i_k)$ in `Auction`, then the algorithm will try to extend P from i_k. In particular:

$$\text{if } \overline{\pi}_k(i_k) = \min_{(i_k,j_k) in FS(i_k)} \{c_{i_k j_k} + \overline{\pi}_k(j_k)\}, \text{ then, } j_k = \operatorname{argmin}_{(i_k,j_k) in FS(i_k)} \{c_{i_k j_k} + \overline{\pi}_k(j_k)\},$$

then path P is extended through the arc (i_k, j_k), as described in lines `18-19` of Figure 2.2. At the same time, in `R-Auction`:

$$\text{if } \pi_k(i_k) = \min_{(i_k,j_k) in FS(i_k)} \{\overline{c}_{i_k j_k} + \pi_k(j_k)\}, \text{ then, } j_k = \operatorname{argmin}_{(i_k,j_k) in FS(i_k)} \{\overline{c}_{i_k j_k} + \pi_k(j_k)\},$$

and $\overset{s}{T}_{k-1}$ is extended through the arc (i_k, j_k), as described in line `19` of Figure 2.4. Thus, for each node $v \in \overset{r}{V}_{k-1}$:

$$\begin{aligned} \overline{\pi}_k(v) &= \overline{\pi}_{k-1}(v) && \text{in \texttt{Auction};} \\ \pi_k(v) &= \pi_{k-1}(v) && \text{in \texttt{R-Auction}.} \end{aligned}$$

Given that $\overline{\pi}_{k-1}(v) = 0$ and $\pi_{k-1}(v) = 0$, then for `step-1` condition 2.8 is true. Concerning i_k, it holds that:

$$\begin{aligned} \overline{\pi}_k(i_k) &= c_{i_k j_k} + \overline{\pi}_k(j_k) && \text{in \texttt{Auction};} \\ \pi_k(i_k) &= \overline{c}_{i_k j_k} + \pi_k(j_k) && \text{in \texttt{R-Auction};} \\ w_k(i_k) &= w_k(i_k) + (\pi_k(i_k) - \pi_{k-1}(i_k)) && \text{in \texttt{R-Auction}.} \end{aligned}$$

Moreover, since $j_k \in \overset{r}{V}_{k-1}$, it follows that $\overline{\pi}_k(j_k) = \pi_k(j_k) = 0$ and by substituting, we obtain that:

$$\begin{aligned} \overline{\pi}_k(i_k) &= c_{i_k j_k} && \text{in \texttt{Auction};} \\ \pi_k(i_k) &= \overline{c}_{i_k j_k} && \text{in \texttt{R-Auction};} \\ w_k(i_k) &= w_k(i_k) + (\pi_k(i_k) - \pi_{k-1}(i_k)) && \text{in \texttt{R-Auction}.} \end{aligned}$$

Given that $\overline{c}_{i_k j_k} = c_{i_k j_k} + [d_r(i_k) - d_r(j_k)]$, computed at lines `5-6` in Figure 2.4, it follows that:

$$\pi_k(i_k) = c_{i_k j_k} + (d_r(i_k) - d_r(j_k)) \qquad \text{in \texttt{R-Auction},}$$

then the condition 2.9 is true

$$\pi_k(i_k) = \overline{\pi}_k(i_k) + (d_r(i_k) - d_r(j_k)).$$

Concerning the condition 2.10, for $\overset{s}{T}_{k-1}$,

$$\begin{aligned} w_k(i_k) &= w_k(i_k) + (\pi_k(i_k) - \pi_{k-1}(i_k)) \\ &= d_s(i_k) + c_{i_k j_k} \qquad (2.12) \\ &= d_s(i_k) + \overline{\pi}_k(i_k), \end{aligned}$$

where equation(2.12) is true by the induction hypothesis (as stated and proved in [14, 15]).

After the extension, the algorithm `R-Auction` inserts in the solution under construction the arc(i_k, j_k) and the subtree rooted in j_k, obtaining the partial solution $\overset{s}{T}_k = (\overset{s}{V}_k, \overset{s}{A}_k)$. $\overset{s}{T}_k$ will be an optimal solution for the subgraph G_k of G induced by $\overset{s}{T}_k$. At this point, $\overset{s}{T}_k$ will be extracted from $\overset{r}{T}_{k-1}$, obtaining a new subtree for the nodes not yet in solution, $\overset{r}{T}_k = (\overset{r}{V}_k, \overset{r}{A}_k)$, where:

- $\overset{r}{V}_k = \overset{r}{V}_{k-1} \setminus \overset{s}{V}_k$;
- $\overset{r}{A}_k = \overset{r}{A}_{k-1} \setminus \overset{s}{A}_k$.

$\square$

2.4.2 Re-optimization Dijkstra algorithmic scheme: `R-Dijkstra`

Adopting the same principle used for `R-Auction` we designed a Dijkstra-like algorithm for addressing the R-SPTP, in the case of origin change, which reuses information related to the previous optimal solution. During its executions, the classical Dijkstra algorithm maintains for each node $v \in V$ a distance label $d(v)$, a predecessor $p(v)$, and a status $S(v) \in \{\mathtt{unreached}, \mathtt{labeled}, \mathtt{scanned}\}$. All nodes with a not `scanned` status are stored in non-decreasing order, according to the label $d()$, in a priority queue Q.

Let $\overset{r}{T} = (\overset{r}{V}, \overset{r}{A})$ be an optimal solution to the problem $P^{(k-1)}$. In order to solve the problem $P^{(k)}$, which requires finding a shortest path tree rooted in a new origin $s \neq r$, the `R-Dijkstra` algorithm extracts from $\overset{r}{T}$ the sub-tree rooted in s, $\overset{s}{T}$ (lines `4-6` in Figure 2.8). Then, it visits in post-order $\overset{r}{T}$ and for each visited node v, it assigns $p(v)$, $d(v)$, deletes $BS(v)$. Furthermore, if $FS(v) = \emptyset$, the algorithm sets $S(v) = \mathtt{scanned}$; otherwise, it sets $S(v) = \mathtt{labeled}$ and inserts v in Q, `lines 6-11`. Starting from these initial settings, the classical Dijkstra's algorithm is then applied, `line 12`.

The correctness of R-Dijkstra is evident and very simple to prove. $\overset{s}{T}$ is an optimal solution for the subgraph $\tilde{G}$ of G induced by $\overset{s}{T}$ and the rest of the solution is built according to the classical Dijkstra's paradigm.

```
R-Dijkstra( $G = (V, A)$, C, $d_r(\cdot)$, $p_r(\cdot)$, s, $T_r$ )
$d_s \leftarrow +\infty$ ;
$p_s \leftarrow$ NIL ;
$T_s \leftarrow$ extractSubtree( s, $T_r$, );
$T_r \leftarrow T_r \setminus T_s$ ;
foreach ( $i \in T_s$ ) do                                  /* in post-order visit */
    $d_s(i) \leftarrow d_r(i) - d_r(s)$;
    $p_s(i) \leftarrow p_r(i)$;
    remove-BS( $T_s$, i );
    enqueue ( Q, i );                                       /* if $FS(i) \neq \emptyset$ */
$p_s(s) \leftarrow 0$;
DIJKSTA( G, C, $d_s$, $p_s$, s, Q );
return $\{p_s, d_s\}$;
```

Fig. 2.8 The pseudocode of the re-optimization Dijkstra algorithmic scheme.

2.4.3 Computational complexity

We will analyze the theoretical computational complexity of the algorithms described above, R-Auction and R-Dijkstra.

Theorem 3. *The computational complexity of* R-Auction *is* $\mathcal{O}(n^2 \log n)$.

Proof. Since $|V| = n$ and $|E| = m$, for the R-Auction, we have that:

- the computation of the reduced costs matrix, $\overline{C}$ (lines 5-6 in Figure 2.4), requires $\mathcal{O}(m)$;
- tree and queuing operations, lines 10-12, and line 13, require in the worst case $\mathcal{O}(m)$. This case can occur if and only if s is the only child of r. In this case, only one extension is required to complete the solution.

Furthermore, we know from a theorem proved in [15] that the worst case complexity for the whole virtual source algorithm, whose essential policy is used in our R-Auction to extend the actual solution, is $\mathcal{O}(n \log n)$, i.e. $n - 1$ iterations in each of which one extension, with complexity $\mathcal{O}(\log n)$, is performed. Then our function find-extension is $\mathcal{O}(\log n)$. R-Auction performs at most $n - 1$ extensions, if and only if each sub-tree integrated in the solution consists of only one node. This is the worst case for `R-Auction`, whose complexity is given by:

$$\max\left\{\mathcal{O}(n\log n), \mathcal{O}(m), \mathcal{O}\Big(\sum_{i=1}^{n} i\log n\Big)\right\},$$

where $n\log n$ is the time required for $n-1$ applications of the procedure `find-extension`, $\mathcal{O}(m)$ is the time required for the deletion of all backward stars, and $\mathcal{O}\big(\sum_{i=1}^{n} i\log n\big)$ is the time required for the tree setting operations. Since we are considering the worst case, the under construction solution grows one node at each iteration and the updating of the node labels requires $\mathcal{O}(\log n)$. It is easy to conclude that the computational complexity is given by:

$$\sum_{i=1}^{n} i\log n = \frac{1}{2}n(n+1)\log n = \mathcal{O}(n^2\log n).$$

□

Theorem 4. `R-Dijkstra` *has the same computational complexity of the classical Dijkstra's algorithm,* $\mathcal{O}(m\log n)$

Proof. In the algorithm, described in Figure 2.8, the procedure `extractSubtree` works in $\mathcal{O}(n)$, subsequent phase, when the nodes of the extracted sub-tree are inserted in Q, requires $\mathcal{O}(\log n)$. While the last phase that recalls a procedure that applies the Dijkstra algorithm starting from the node in Q is $\mathcal{O}(m\log n)$. □

2.4.4 Computational Results

The aim of this paragraph is to assess the behavior of the proposed solution approaches (i.e., `R-Auction` and `R-Dijkstra` algorithms) in terms of computational cost and to compare them to a classic approach from scratch, based on the Dijkstra algorithm (referred in the sequel to as `R-Scratch`). All the algorithms have been coded in `C`, compiled with `gcc 5.4.0` and tested by using an Intel(R) core(TM) i7 CPU 4720HQ, 2.60 GHz, ram 8.00 GB, under a `ubuntu 16.04 LTS` operating system.

To make a fair comparison among all the algorithms used in the testing phase, we have adopted the same data structure to represent the priorities queue. This structure was inspired by the Multi-Level Bucket proposed in [18, 19]. It must be highlighted that this structure was designed ad hoc for the Dijkstra algorithm.

Test Problems

The instances used in the testing phase can be grouped into the following main four different categories:

1. `Grid`-instances: the test problems belonging to this set were built by using the network generator described in [46]. In particular, the following three different typologies of grid networks are considered.

   `PURE-GRID`: they are classical two-dimensional grids $G = (V, A)$, where the set $V = \{1, \dots, n^2\}$ is composed by n levels of nodes. All nodes $v \in V$ are divided into the following three main sets:

   corner-node: (CN), defined as $CN = \{1, n, n^2 - n + 1, n^2\}$;

   lateral-node: (LN), that is, $LN = \{v \mid 1 < v < n \text{ and } n^2 - n + 1 < v < n^2 \text{ and } \mod(v, n) = \{0, 1\}\} \setminus CN$;

   inner-node: (IN), the remaining nodes, $IN = V \setminus CN \setminus LN$;

   The aforementioned node sets are characterized by the following connections: $\forall\, v \in CN \Rightarrow |BS(v)| = |FS(v)| = 2$, $\forall\, v \in LN \Rightarrow |BS(v)| = |FS(v)| = 3$, $\forall\, v \in IN \Rightarrow |BS(v)| = |FS(v)| = 4$.

   `CYLINDER`: the structure is similar to the `PURE-GRID` one:

   $CN = \{\emptyset\}$;

   $LN = \{v \mid 1 < v < n \text{ and } n^2 - n + 1 < v < n^2\}$;

   and there are supplementary bilateral connections for each pair of nodes (u, v) such that $v = u + (n - 1)$, where $\mod(u, n) = 1$.

   `TORUS`: the structure is similar to the `CYLINDER` one:

   $LN = \{\emptyset\}$;

   and there are supplementary bilateral connections for each pair of nodes (u, v) such that $v = n^2 - n + u$, where $1 < u < n$.

   The computational experiments were carried out by considering instances of increasing dimensions. In particular, for each type of grid (i.e.,`PURE-GRID`, `CYLINDER`, `TORUS`) the number of nodes was chosen equal to: {`100`$\times$`100`, `200`$\times$`200`, `300`$\times$`300`, `400`$\times$`400`, `500`$\times$`500`}.

2. `GRIDGEN`-instances: obtained by using the network generator implemented by Bertsekas, [11], which constructs a two-dimensional grid with wraparound

structure. The skeleton structure is made up of a $n \times n$ grid, where each node is connected to four neighbors nodes. Since the generator was proposed for the minimum cost flow problem, each grid contains two special nodes, *source* and *sink*, which are connected to each other node in the grid. In this way, for each node v, $|FS(v)| = |BS(v)| = 6$. Furthermore, it is possible to add a given number of supplementary arcs between randomly selected nodes. The experiments were carried out on square grids, where the nodes were set equal to {`100×100, 150×150, 250×250, 500×500`}. For each size, we generated three different typologies of network: `n×n-6N` the standard skeleton network; `n×n-6N+25` standard network with 25% of additional randomly chosen arcs; `n×n-6N+50` standard network with 50% of additional randomly selected arcs.

3. `REAL`-instances: these test problems were chosen from the USA road networks available at the web page of `9th DIMACS Implementation Challenge - Shortest Paths`, [27]. Each graph comes in two versions: physical distance and transit time arc lengths. For our experiments, we considered three different road networks which describes the area of New York City, San Francisco Bay and Colorado.

4. `RANDGRAPH`-instances: these networks were generated with a command line tools generator, which builds different types of graphs and presented in [27, 103]. For our experiments we generated random graphs with a fixed maximum out degree according to the size of the graph. In particular, let n be the selected dimension (number of nodes) of the graph, we generate three different kinds of graph such that the maximum-out degree for each node δ^+ is given by: 0.1%, 0.25% and 0.5% of n. In the computational experiments, we considered networks for which the number of nodes was set equal to {`10000, 50000, 100000`}. For each size we generated three different typologies of network: `n-outx1`, `n-outx2` and `n-outx3`, where `n` establishes the size, `x1` $= 0.1 \cdot$ `n`, `x2` $= 0.25 \cdot$ `n` and `x3` $= 0.5 \cdot$ `n` defines the maximum random out degree for each node.

Experiments setup and results

The computational experiments were carried out on the networks described in the previous paragraphs. Since the R-SPTP in the case of origin changes is considered in this paper, given an instance $G = (V, A)$, two sets $R = \{r_1, r_2, \dots, r_p\}$ of random "old"-sources and $S = \{s1, s2, \dots, s_q\}$ of random "new"-sources have been selected, with $r_i, s_j \in V$ and $r_i \neq s_j$, $\forall\ 1 \leq i \leq p$ and $1 \leq j \leq q$. In particular, in our settings $p = 5$ and $q = 10$.

All the algorithms were executed for all the pairs in the Cartesian product $R \times S$. For each type of network, three different instances are generated, by randomly choosing the arc cost according to a uniform distribution, within the range $[0, 100000]$. In particular, the `rand()` function of the `stdlib` library with `srand(time(NULL))` initialization is used.

The computational results are reported in Tables 2.1 - 2.5, where the following information is given: the first two columns indicate the instances, with the corresponding name and dimension, the second, third and forth pair of columns summarize the results obtained by each algorithm.

In particular, the field "avg-time"indicates the average computational time (in seconds), while the field "%-win"indicates for each algorithm and for each instance the win rate, defined as the percentage of experiments (rounded to the nearest integer) in which a given algorithm "wins"the others, that is, it shows a lower computational time. To point out the best performances, for each instance, we highlight in bold the algorithm that is either the fastest or behaves the best in the majority of instances.

To better analyze the behavior of the considered approaches, it is useful to consider the results given in Figures 2.9 - 2.12, in which a graphical representation of the computational results achieved in the testing phase through the box-plot diagrams is given.

The computational times (in seconds) are given on the axis of the ordinates, while information related to the considered algorithms (R-Auction, R-Scratch and R-Dijkstra) are reported on the horizontal one.

The box-plots are generated on the basis of the algorithms' running times.

In addition, Figures 2.13-2.16 highlight the win rate in the form of a radar-chart.

Grid-instances results The computational results obtained on the Grid-instances are given in Table 2.1 and in the radar-chart of Figure 2.13. It is evident that for the first set of test problems, the win rate of R-Auction is always higher than the value obtained for the other algorithms. However, for the instances CYLINDER-100x100, TORUS-100x100 and PURE-GRID-200x200 the best running times, on average, are achieved by R-Dijkstra. Grouping the instances according to their typologies, on the PURE-GRID networks, the win rate is equal to 88.9%, 2.8% and 7.4% for R-Auction, R-Scratch and R-Dijkstra, respectively; on the CYLINDER networks the percentages are: 81.8% for R-Auction, 6% for R-Scratch and 12.4% for R-Dijkstra; finally on TORUS networks: 53.2% for R-Auction, 15% for R-Scratch and 34.6% for R-Dijkstra. The superiority of the proposed approach is more evident on the PURE-GRID and the CYLINDER networks, while on the TORUS instances, R-Dijkstra becomes competitive.

Table 2.1 Test results on Grid-instances.

instance		R-Auction		R-Scratch		R-Dijkstra	
type	**size**	t-avg	%-win	t-avg	%-win	t-avg	%-win
PURE-GRID	100x100	**0.009**	**91**	0.010	2	0.010	8
CYLINDER	100x100	0.011	**74**	0.011	8	**0.010**	18
TORUS	100x100	0.013	**71**	0.013	5	**0.012**	25
PURE-GRID	200x200	**0.052**	**78**	0.053	9	**0.052**	12
CYLINDER	200x200	**0.050**	**78**	0.053	11	0.053	11
TORUS	200x200	**0.071**	**65**	0.075	8	0.074	28
PURE-GRID	300x300	**0.130**	**88**	0.148	0	0.144	12
CYLINDER	300x300	**0.143**	**71**	0.153	6	0.149	23
TORUS	300x300	**0.217**	**42**	0.233	25	0.228	34
PURE-GRID	400x400	**0.272**	**92**	0.314	3	0.309	5
CYLINDER	400x400	**0.293**	**89**	0.328	3	0.323	8
TORUS	400x400	**0.494**	**45**	0.524	22	0.522	34
PURE-GRID	500x500	**0.497**	**100**	0.567	0	0.565	0
CYLINDER	500x500	**0.536**	**97**	0.606	2	0.599	2
TORUS	500x500	**0.955**	**43**	0.991	15	0.985	42

GRIDGEN-**instances results** The results obtained on the GRIDGEN-instances are given in Table 2.2 and Figure 2.14. From the radar-chart in Figure 2.14, it is quite clear that R-Auction outperforms both R-Scratch and R-Dijkstra. In fact, analyzing Table 2.2, for more details, one can establish that with the sole exception of the instances 250x250-6N+50 and 500x500-6N+50 where the win rate of R-Auction is equal to 77% and 70%, respectively, in all other cases it is around 90%. As far as the comparison between the two Dijkstra-like algorithms is concerned, R-Dijkstra outperforms R-Scratch. More specifically, in the majority of cases, the win rate of R-Dijkstra is close to 10% (with a peak of 23% for the instance 250x250-6N+50). On the other hand, in only one case R-Scratch reaches a win rate of 13% while in all others it is between 0% and 3% (see Table 2.2).

Table 2.2 Test results on instances generated with `GRIDGEN` generator, [11]. The name of the instances indicates the relative dimension, given by "$n \times m$", and the number of connections. The default number of arcs is given by the skeleton structure, $6 \cdot (n \times m)$, to which have been added 25% and 50% of new connections, respectively.

instance		`R-Auction`		`R-Scratch`		`R-Dijkstra`	
type	**size**	t-avg	%-win	t-avg	%-win	t-avg	%-win
`100x100-6N`	`100x100`	**0.112**	**87**	0.125	0	0.124	13
`100x100-6N+25`	`100x100`	**0.170**	**93**	0.213	3	0.212	4
`100x100-6N+50`	`100x100`	**0.180**	**90**	0.261	3	0.254	7
`150x150-6N`	`150x150`	**0.832**	**97**	0.986	0	0.956	3
`150x150-6N+25`	`150x150`	**0.995**	**90**	1.328	0	1.282	10
`150x150-6N+50`	`150x150`	**1.044**	**87**	1.530	0	1.471	13
`250x250-6N`	`250x250`	**8.117**	**90**	9.127	0	8.814	10
`250x250-6N+25`	`250x250`	**9.186**	**90**	11.528	0	11.100	10
`250x250-6N+50`	`250x250`	**11.394**	**77**	15.153	0	14.358	23
`500x500-6N`	`500x500`	**288.259**	**90**	419.797	3	399.360	7
`500x500-6N+25`	`500x500`	**319.562**	**90**	403.882	3	393.683	7
`500x500-6N+50`	`500x500`	**375.822**	**70**	530.773	13	522.269	17

`REAL`-instances results For this set of networks, two different settings for the experiments have been considered. In the first case, the same setup strategy used for the other instance classes as described in 2.4.4 is adopted. The related results, reported in Table 2.3, clearly underline that the Dijkstra-like strategies outperform `R-Auction`, both on the win rate and the average running times. In particular, the win rate of `R-Auction` is on average 14.5%, whereas it is 34% and 51.5%, for `R-Scratch` and `R-Dijkstra`, respectively. In addition, the comparison between `R-Scratch` and `R-Dijkstra`, underlines that the former behaves the best. Indeed, `R-Dijkstra` is about 4% faster than `R-Scratch`, according the running times, while the difference in the win rate is around 20% in favor of `R-Dijkstra`, with the sole exception of the instance `USA-road-SFbay` where the difference is 2%. The specific behavior of `R-Auction` on this set of test problems, that is not in line with the one observed on the `Grid` and `GRIDGEN` instances, can be justified by considering the specific data structure used to store the nodes with temporal label, i.e. the priority queue. We underline that since `R-Auction` is based on a dual approach (unlike a Dijkstra-like algorithm), it considers the arc costs as reduced costs. Therefore during the execution of a Dijkstra-like algorithm, for each node i, the label associated with the node specifies the cost of the best path found so far from the origin node to i. If i is a *scanned* node (see Paragraph 2.4.2 for more details) then its label indicates the cost of the shortest path from the origin to i (and the label will not change anymore), otherwise the label is an upper bound on the shortest path cost from the origin to i and it could be decreased afterwards. On the contrary, the label associated with a given node i during the execution of R-Auction represents the reduced cost of the best path found so far from the origin to i. By using the same terminology of the Dijkstra framework, if i is a *scanned* node (that is $FS(i) = \emptyset$, see Paragraph 2.4 for more details) then the label will not change anymore and a shortest path is found, otherwise the label could be increased afterwards. This difference between the two approaches justifies the deterioration of the performance and the efficiency of R-Auction, if a priority queue, well tailored to a Dijkstra-like approach is used, as in our case.

In order to investigate this issue better, another set of instances were generated, by defining opportunely the set S of the "new "-sources. Indeed, the new sources are selected on the basis of their distance from the old source, in decreasing order. In particular, the nodes belonging to the set of "old"-source R are selected randomly, as specified in Paragraph 2.4.4, while the nearest k nodes from the "old"-source are inserted in S. The results obtained on this new set of instances are reported in Table 2.4. The trend observed for the R-Auction is considerably different from that shown with the other REAL-instances. Indeed, R-Auction behaves the best, both in terms of win rate and average running times.

In particular, R-Auction is around 9% and 10% faster than R-Dijkstra and R-Scratch, respectively, considering the average of running times. On the other hand, as regarding the win rate, the value obtained with R-Auction is almost always greater than 90%, with the sole exception of the instances USA-road-COL-t, for which a win rate of the 89% is observed, and USA-road-SFbay, where it is 51%, while R-Scratch and R-Dijkstra show a win rate almost always lower than 10%, except for the instance USA-road-SFbay.

RANDGRAPH-**instances results** The computational results obtained on this set of test problems are given in Table 2.5 and in Figures 2.16 and 2.12. On this class of instances, the experimental results are comparable with those obtained on the GRIDGEN networks. In fact, the radar-chart in Figure 2.16 clearly underlines that R-Auction outperforms R-Scratch and R-Dijkstra, both in terms of win rate and computational overhead. Looking at the results of Table 2.5, it is evident that the win rate of R-Auction is ever greater than 90%. Furthermore, the good performances of R-Auction are also confirmed by the box-plot in Figure 2.12. In particular, for

Table 2.3 Test results on real instances, from 9th DIMACS Implementation Challenge - Shortest Paths, [27]. In instances with suffix "-t" the arc costs is given by the travel time, in the other by physical distance.

instance		R-Auction		R-Scratch		R-Dijkstra	
name	**size**	t-avg	%-win	t-avg	%-win	t-avg	%-win
USA-road-NY	264346	0.842	14	0.512	33	**0.509**	**52**
USA-road-SFbay	321270	0.651	32	0.463	33	**0.438**	**35**
USA-road-COL	435666	3.149	2	0.577	41	**0.569**	**57**
USA-road-NY-t	264346	1.262	2	0.863	38	**0.841**	**60**
USA-road-SFbay-t	321270	1.161	35	0.706	22	**0.683**	**43**
USA-road-COL-t	435666	4.300	2	0.787	37	**0.780**	**62**

Table 2.4 Test results on real instances, from 9th DIMACS Implementation Challenge - Shortest Paths, [27]. Where the new sources were chosen on the basis of their distance from the old source, in decreasing order. The set of "old"-source R is selected random as specified above in the setup section, while the set of "new"-source S is recomputed from scratch at each test and it is given by the list of the firsts q nodes further from the actual old-source.

instance		R-Auction		R-Scratch		R-Dijkstra	
name	**size**	t-avg	%-win	t-avg	%-win	t-avg	%-win
USA-road-NY	264346	**0.415**	**92**	0.426	2	0.425	6
USA-road-SFbay	321270	**0.430**	**51**	0.436	22	0.435	27
USA-road-COL	435666	**0.502**	**94**	0.550	2	0.546	5
USA-road-NY-t	264346	**0.576**	**100**	0.680	0	0.679	0
USA-road-SFbay-t	321270	**0.539**	**92**	0.675	3	0.673	5
USA-road-COL-t	435666	**0.720**	**89**	0.745	5	0.743	6

Table 2.5 Test results on random instances generated with RANDGRAPH generator, [103]. The names of the instances indicates the structure of the relative network. In particular, the first number indicates the dimension, in thousands, while the number after "out" specifies the maximum outgoing degree of each nodes.

instance		R-Auction		R-Scratch		R-Dijkstra	
type	**size**	t-avg	%-win	t-avg	%-win	t-avg	%-win
10k-out10	10000	**0.021**	**97**	0.040	0	0.039	3
10k-out25	10000	**0.047**	**90**	0.135	0	0.133	10
10k-out50	10000	**0.047**	**100**	0.329	0	0.322	0
50k-out50	50000	**6.025**	**100**	12.715	0	12.611	0
50k-out125	50000	**4.405**	**100**	20.448	0	20.376	0
50k-out250	50000	**5.317**	**100**	23.490	0	23.351	0
100k-out100	100000	**40.039**	**100**	161.805	0	153.912	0
100k-out250	100000	**46.957**	**100**	158.978	0	152.527	0
100k-out500	100000	**40.798**	**100**	250.255	0	248.564	0

the instance of dimension equal to 10000, R-Auction is around 78% and 77% faster than R-Scratch and R-Dijkstra, respectively; for the instance of dimension 50000, R-Auction is around 72% and 71% faster than R-Scratch and R-Dijkstra; finally, for the instance of dimension 100000, R-Auction is around 78% and 77% faster than R-Scratch and R-Dijkstra, respectively.

R-Auction **with an ad-hoc priority queue**

The computational results reported in the previous paragraphs show that, in some cases (i.e., especially for the Grid and only moderately for GRIDGEN instances) R-Auction exhibits significant variability in the execution time.

In order to overcome this drawback, we developed an ad-hoc structure to implement the priority queue for the R-Auction, and we report in this paragraph some computational results, for the sole purpose of showing how it is possible to make the proposed auction algorithm more stable from a computational point of view.

The related results are given in Table 2.6 and box-plots in Figures 2.17 2.18, 2.19 and 2.20, where the auction algorithm for the re-optimization equipped with the new priority queue is referred to as R-Auction-b.

In particular, Table 2.6 reports the averages of the Relative Standard Deviations (RSDs) for the two versions of the Auction algorithm. The RSDs were computed to measure the dispersion of the probability distribution of the running times collected in the testing phase. From the data reported in the aforementioned table, it is evident that the variability in the running times is drastically reduced when the new structure is adopted. In addition, an improvement in the efficiency is also observed.

In what follows, we analyze the results for each category of test problems.

Table 2.6 Average of RSD for R-Auction-b and R-Auction. REAL* indicates the special case, considered only for the REAL-instances, when the new sources are taken from the list of the further nodes from the old origin.

class-instance	R-Auction-b	R-Auction
Grid	10.442	16.897
GRIDGEN	17.568	22.726
REAL	12.993	25.454
REAL*	7.652	19.863
RANDGRAPH	17.119	44.800

Grid-instances (Figure 2.17): for this set of test problems, the use of the ad-hoc priority queue allows also a reduction in the execution time to be obtained, with the exception of the instances PURE-GRID-100x100 and CYLINDER-100x100. In particular, R-Auction-b is on average 53% times faster than R-Auction. While, the win rate is around 75%, 80% and 93% for R-Auction-b and around 23%, 16% and 6% for R-Auction, considering the average of the win rate for the classes PURE-GRID, CYLINDER and TORUS, respectively. In all the cases, especially for the instances of dimension greater than 300x300 an execution time variability reduction in comparison with R-Auction is observed.

GRIDGEN-instances (Figure 2.18): the R-Auction-b is more efficient than R-Auction on this set of instances. In particular, it is about three time faster than R-Auction. Except for the instance 150x150-6N, the execution time variability is appreciably less than the one observed for the R-Auction and the two Dijkstra-like techniques (R-Scratch and R-Dijkstra);

REAL-instances (Figure 2.19): the computational results collected on this set of instances underline two important issue. The first one is related to the efficiency of the proposed approach. Indeed, R-Auction-b is competitive with the Dijkstra-like approaches, also when the new sources belonging to the set S are chosen randomly. In particular, R-Auction-b is around 19% and 17% faster than R-Scratch and R-Dijkstra, respectively, when the new source is selected randomly. Meanwhile, `R-Auction-b` is around 25% and 24% faster than `R-Scratch` and `R-Dijkstra` when the new source is selected according to the second criterion specified in Paragraph 2.4.4. This behavior was not observed for `R-Auction` (see Table 2.3). Finally, looking at the box-plot, it is evident that the stability of the algorithm is increased with the adoption of the new ad-hoc structure.

`RANDGRAPH`-**instances** (Figure 2.20): the proposed `R-Auction` showed a stable trend when a general data structure was adopted (see Figure 2.12) in relation with the Dijkstra-like strategies. This trend is confirmed for `R-Auction-b` and a better behavior is observed when the new data structure is used (see Figure 2.20). For this class, `R-Auction-b` is about 56% faster than `R-Auction`, and the average win rate, considering all the instances, is around 77% for `R-Auction-b` and 13% for `R-Auction`, respectively.

Finally, it is important to point out that we have also implemented and tested the `R-Scratch` and `R-Dijkstra`, by using the ad-hoc structure, proposed to represent the priority queue for our auction approach. However, the collected computational results showed that the performance of the Dijkstra-like approaches deteriorates. Thus, we have decided to not include the related results.

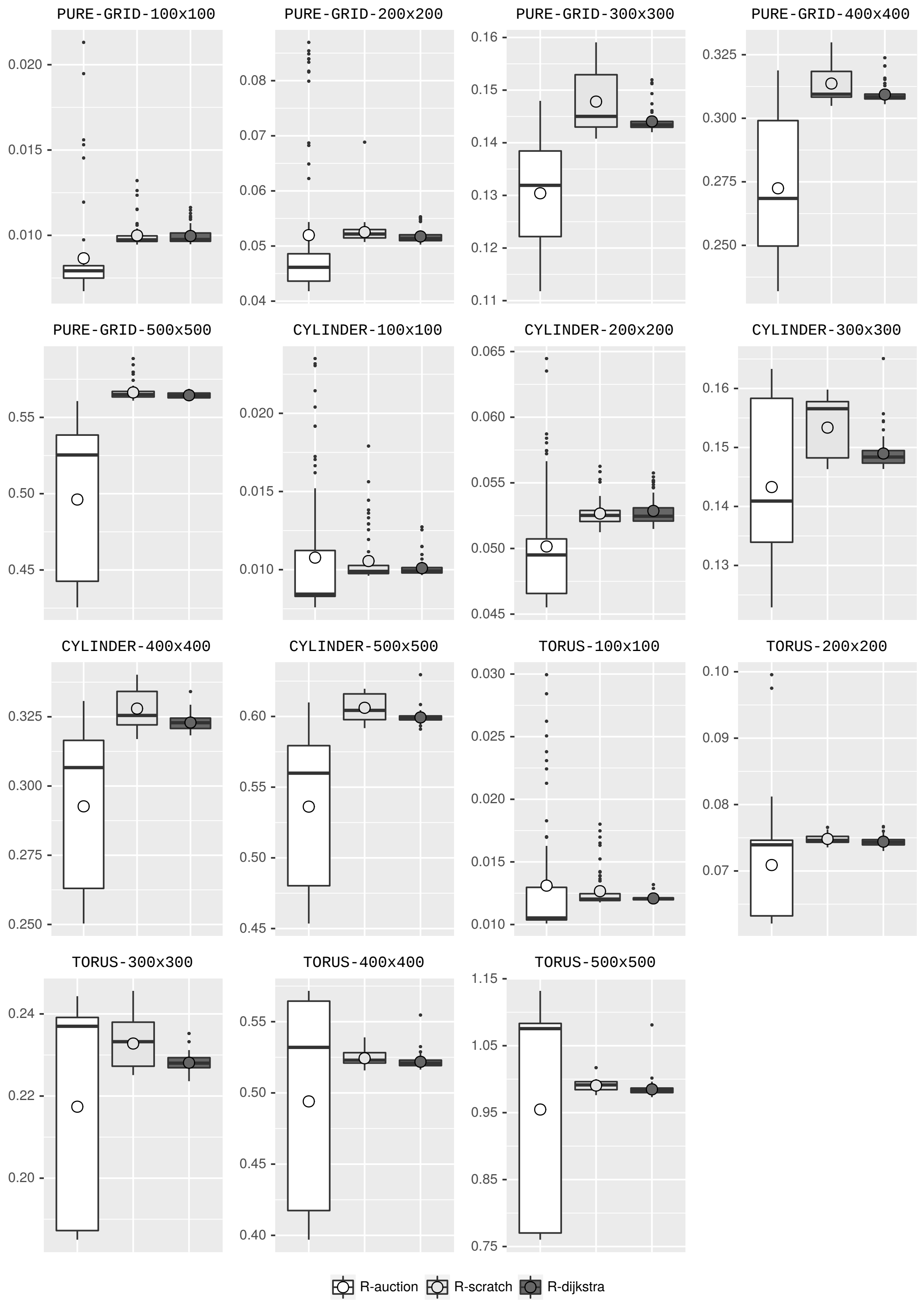


Fig. 2.9 Grid-instances box-plot.

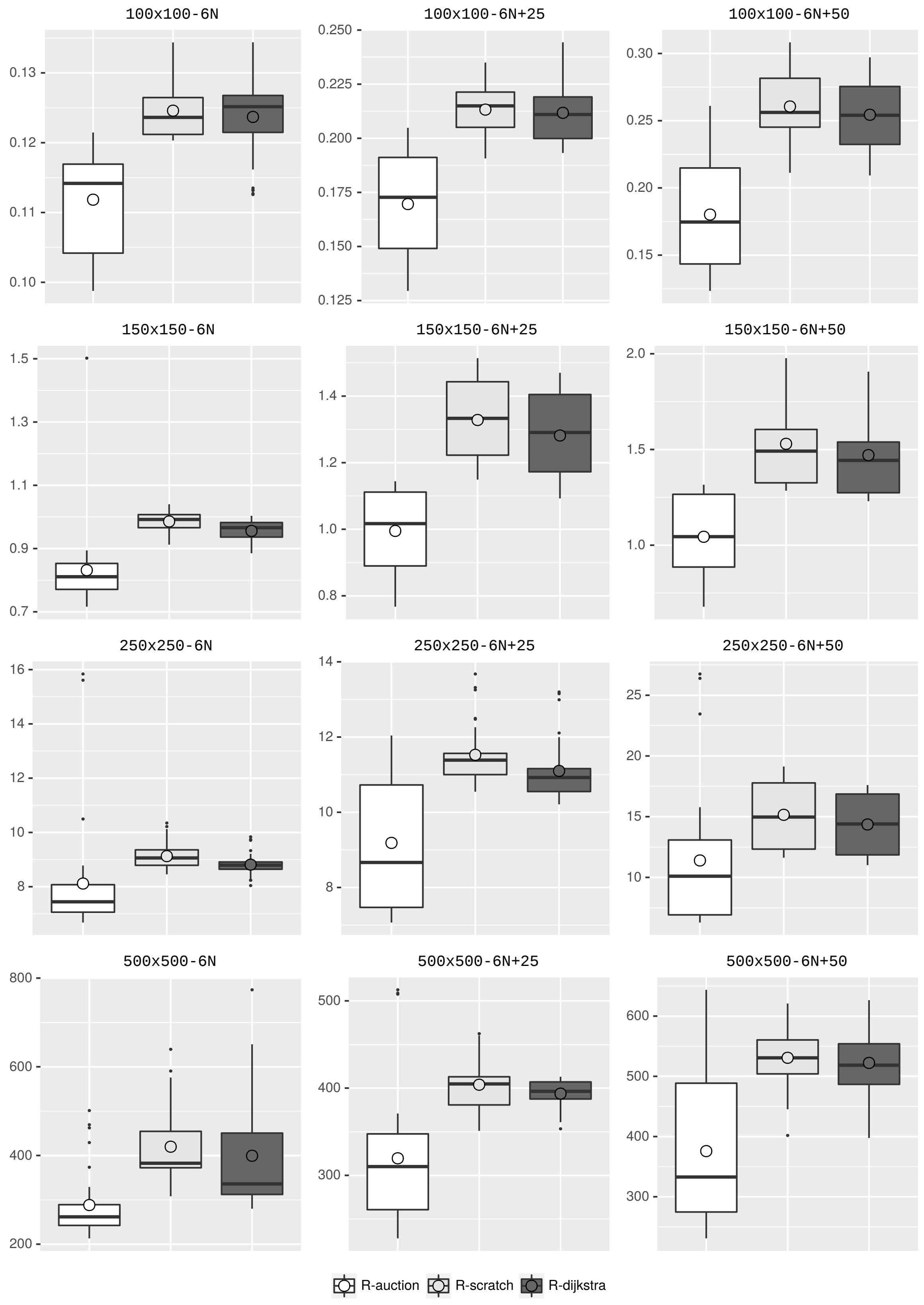


Fig. 2.10 GRIDGEN-instances box-plot.

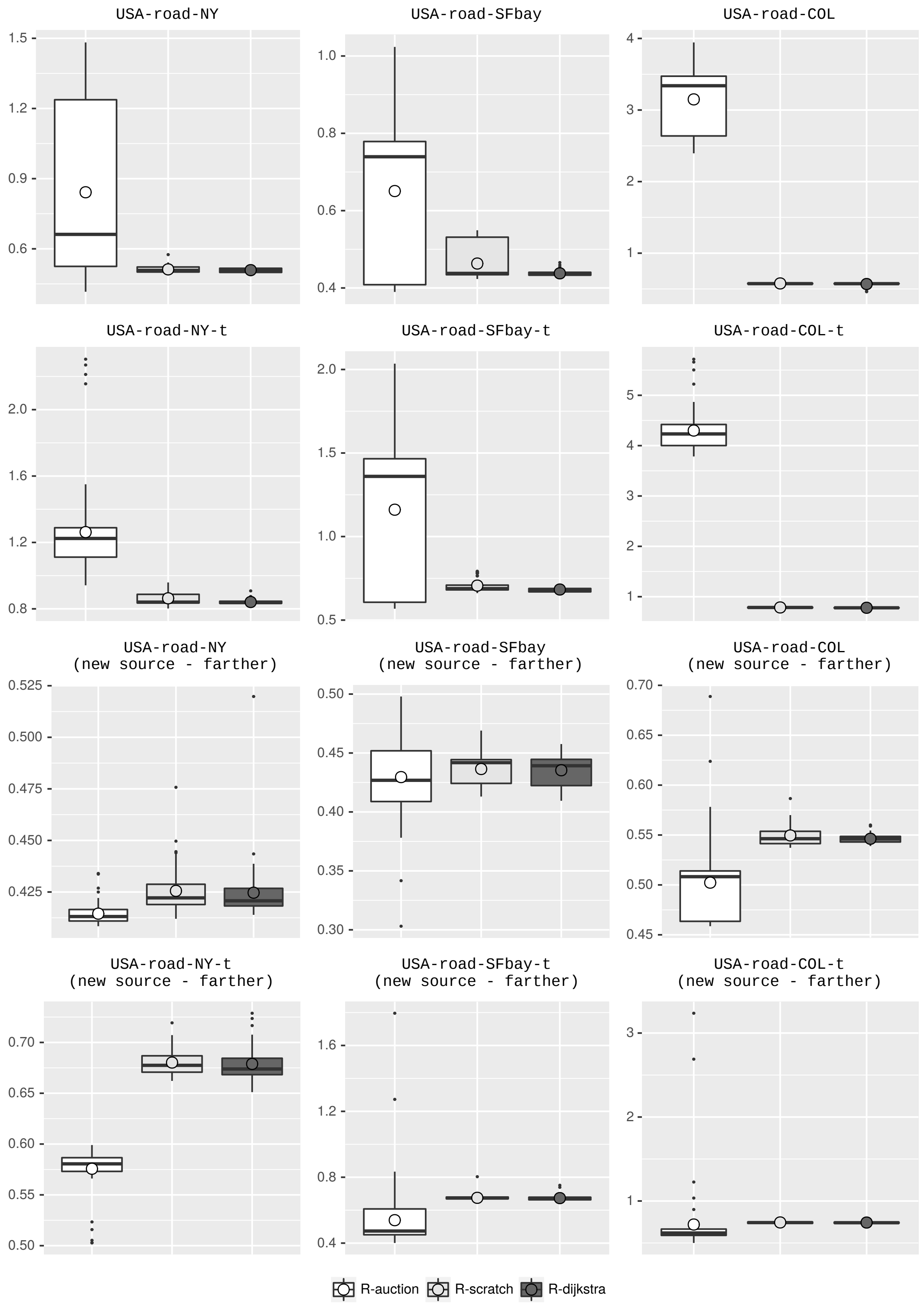


Fig. 2.11 REAL-instances box-plot.

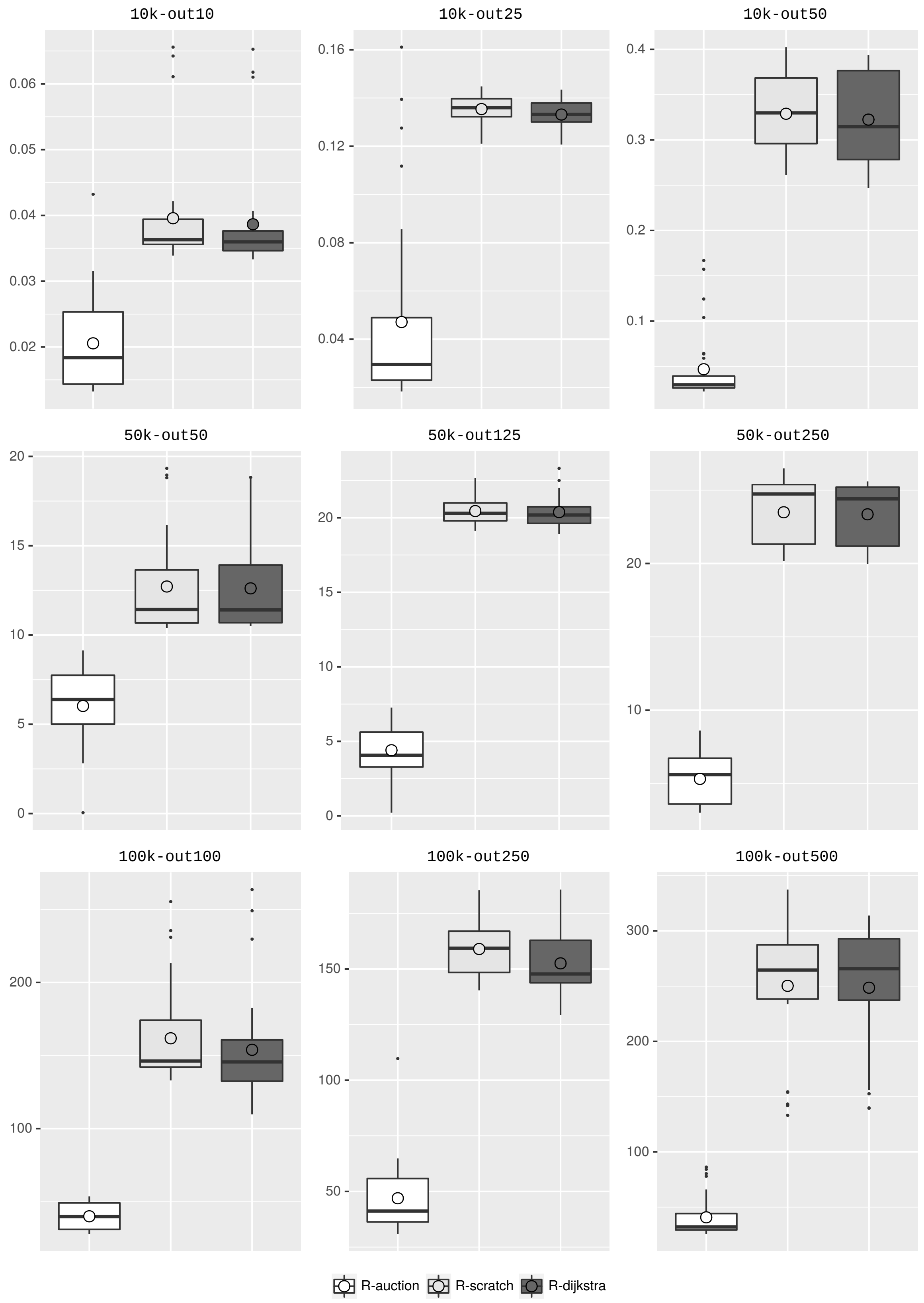


Fig. 2.12 RANDGRAPH-instances box-plot.

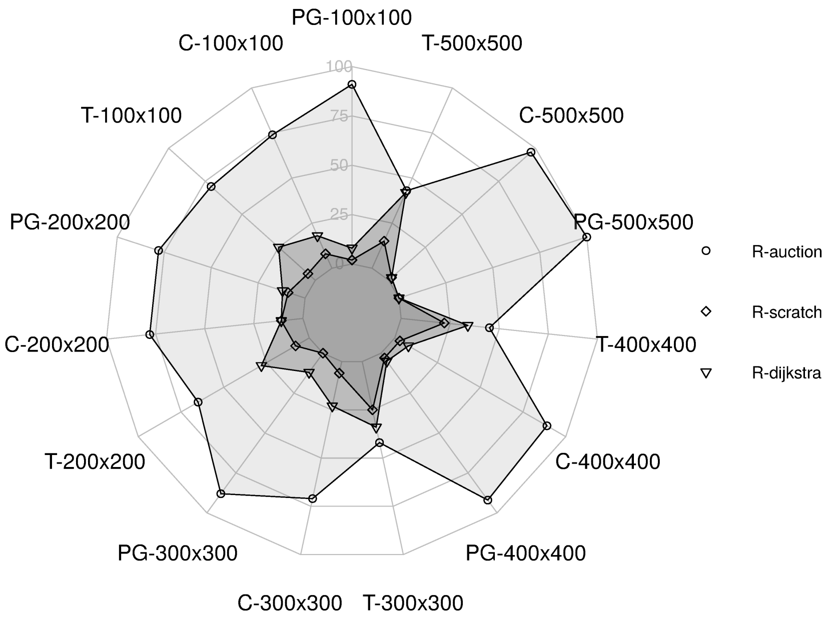


Fig. 2.13 Radar-chart of the win rate on Grid-instances.

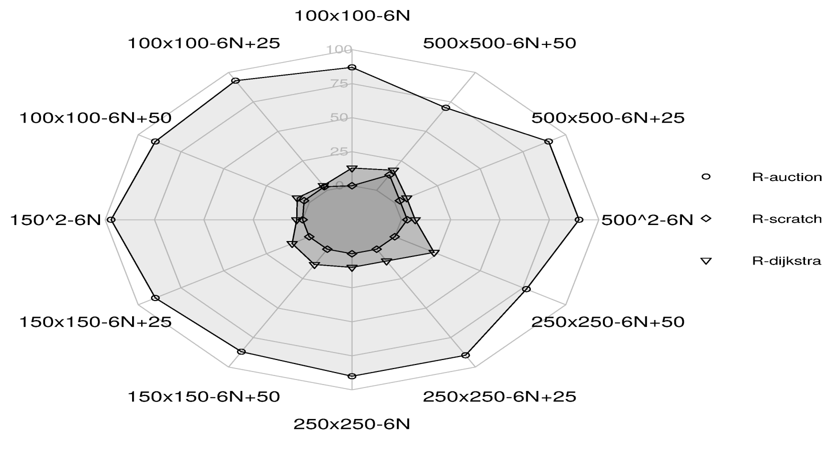


Fig. 2.14 Radar-chart of the win rate on GRIDGEN instances.

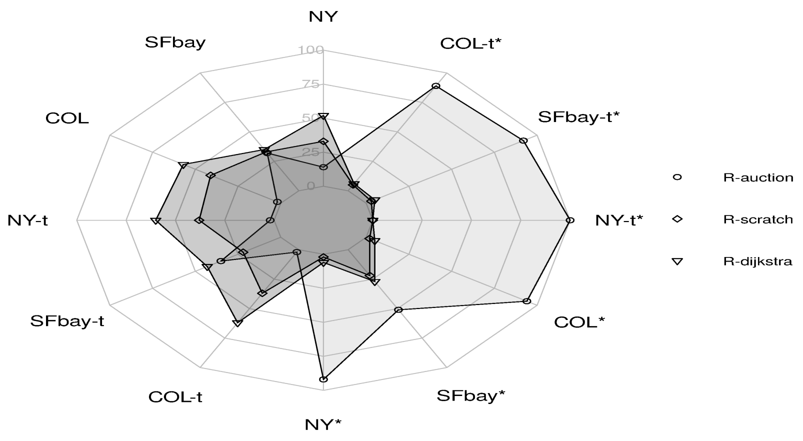


Fig. 2.15 Radar-chart of the win rate on `REAL` instances. The instances with "*" specifies the cases in which the new sources are taken from the list of the further nodes from the old origin.

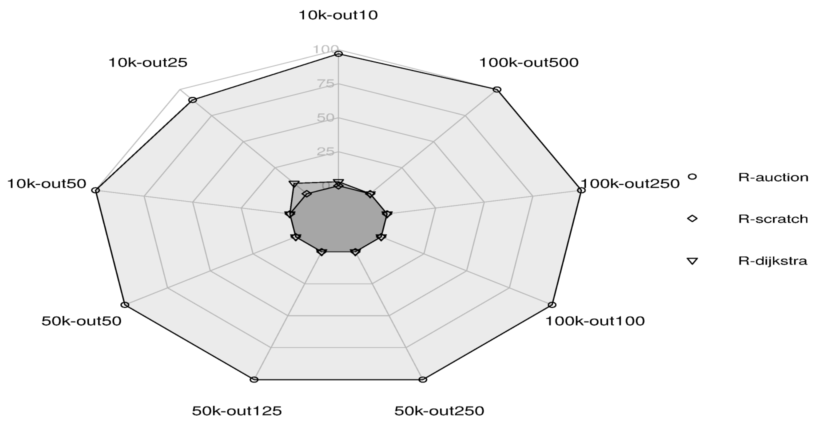


Fig. 2.16 Radar-chart of the win rate on `RANDGRAPH` instances.

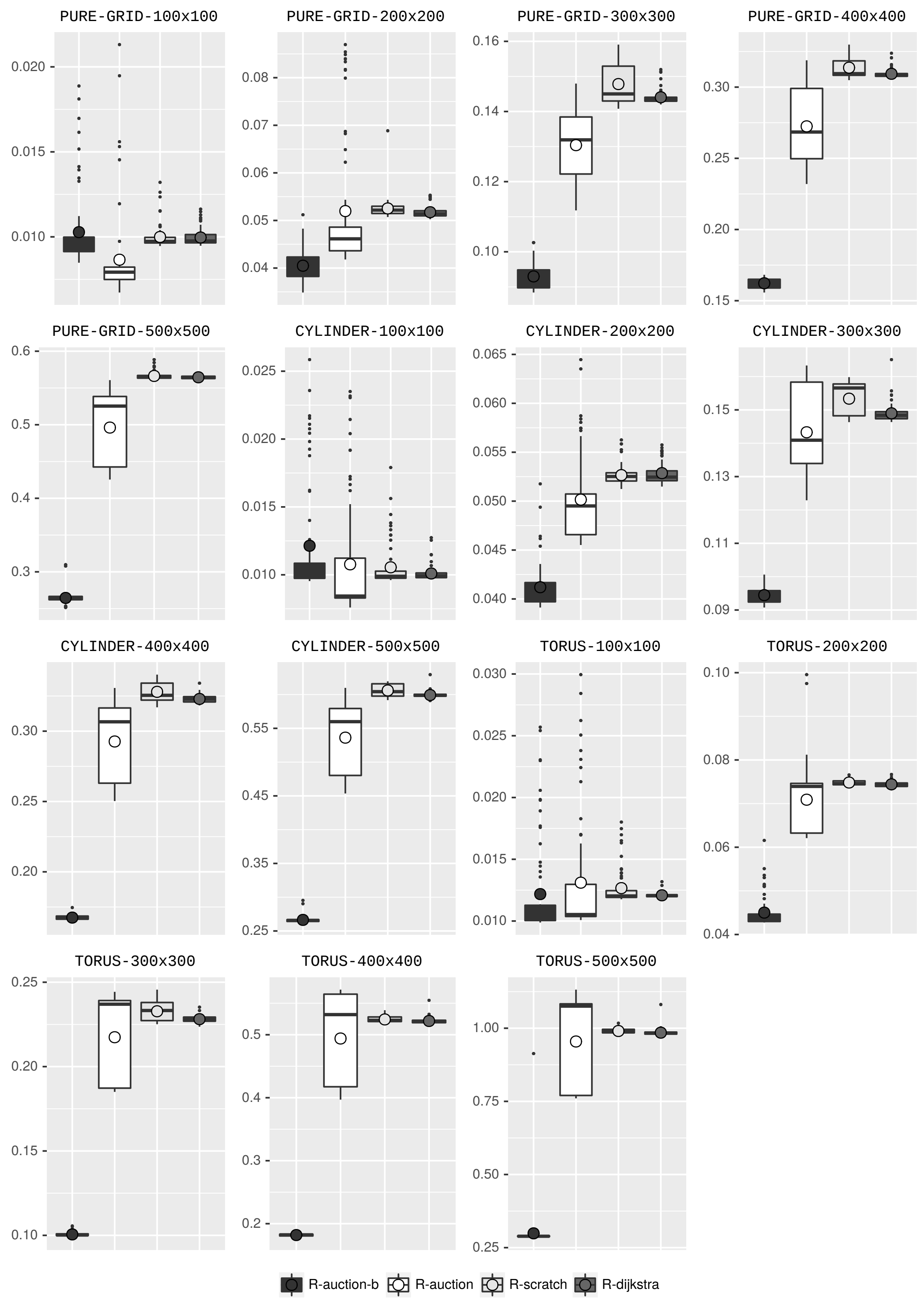


Fig. 2.17 `Grid`-instances box-plot considering an alternative ad hoc structure for the priority queue of `R-Auction`.

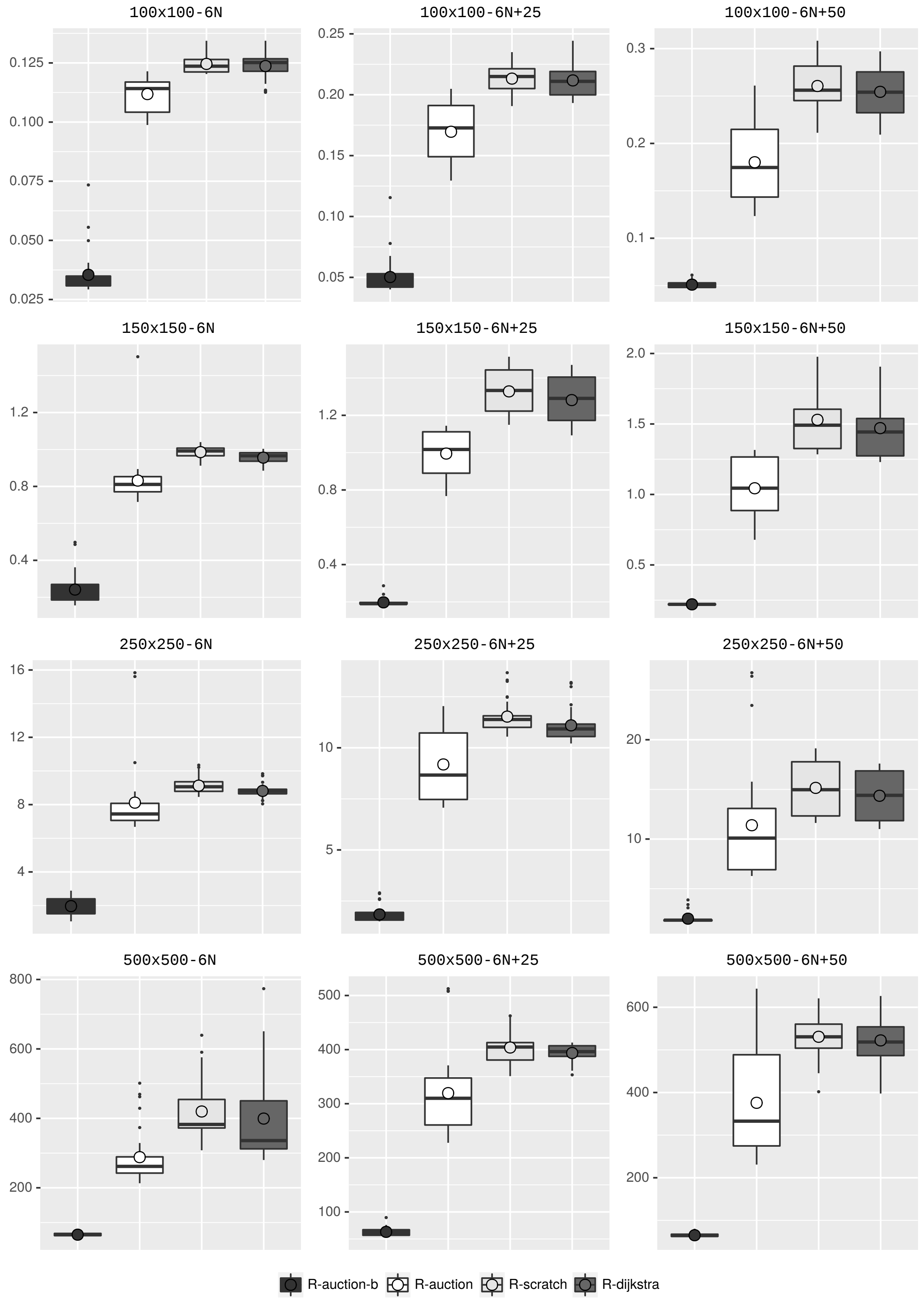


Fig. 2.18 GRIDGEN-instances box-plot considering an alternative ad hoc structure for the priority queue of R-Auction.

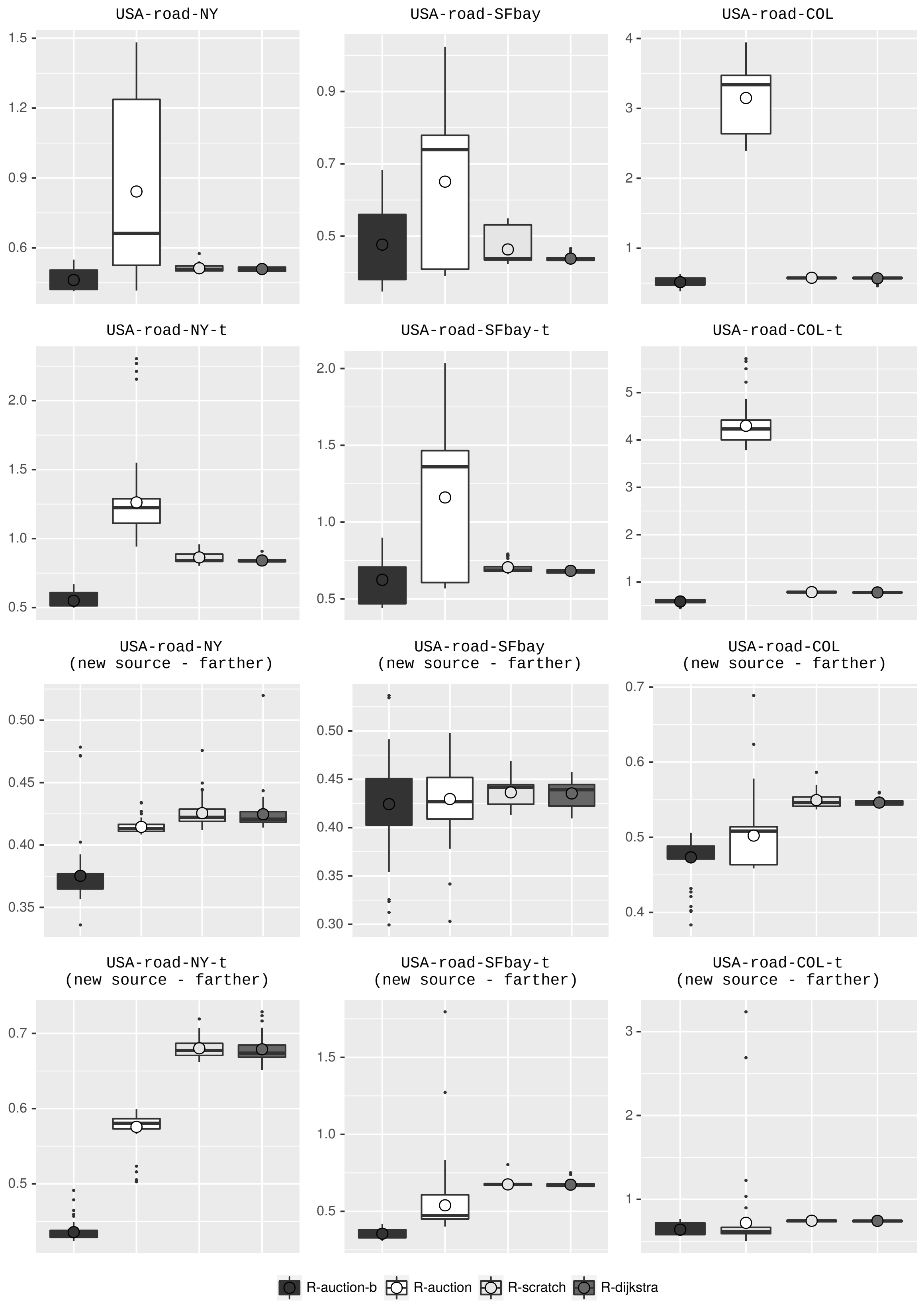


Fig. 2.19 REAL-instances box-plot considering an alternative ad hoc structure for the priority queue of R-Auction.

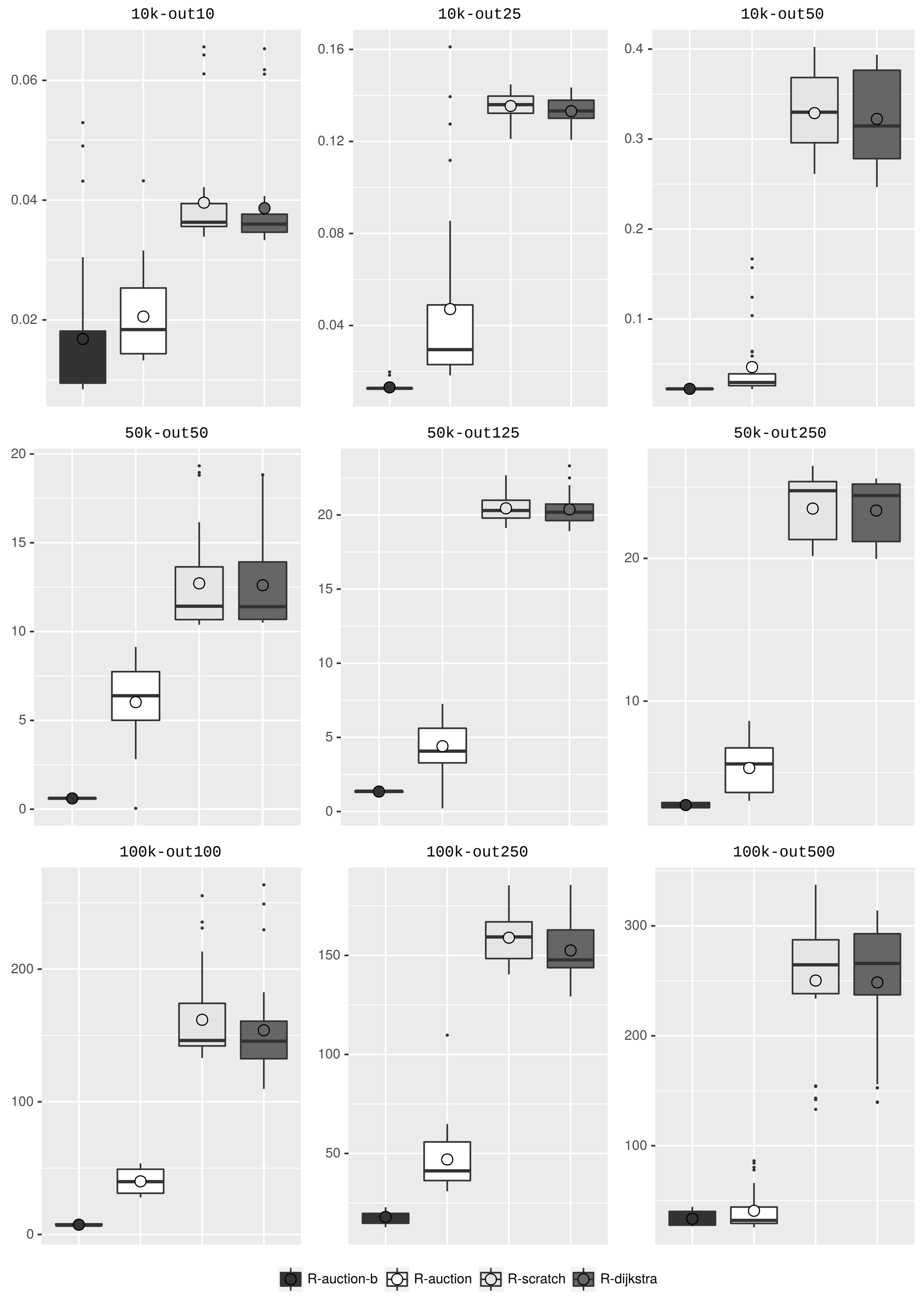


Fig. 2.20 `RANDGRAPH`-instances box-plot considering an alternative ad hoc structure for the priority queue of `R-Auction`.

Chapter 3

Graph Drawing Problem

In this chapter, we discuss the classical graph drawing problem in its two variants, *Two Layers Crossing Minimization Problem* and *Multi-Layer Crossing Minimization Problem*. Moreover, the dynamic version of the problem (Incremental Graph Drawing IGDP) and the novel problem that we proposed (Constrained IGDP, C-IGDP) will be addressed. For these problems we will present their mathematical models, and both recent and well known approaches designed for them. Finally, an exhaustive discussion will be devoted to our new problem, where we introduce the resolutive strategies designed for the C-IGDP , and an extensive numerical evaluation section where we will compare their performances.

3.1 Two Layer Crossing Minimization Problem (TLCMP)

Let $G = (V_1, V_2, A)$ be a bipartite graph with $|V_1| = n_1$ and $|V_2| = n_2$. The *Two-Layer Crossing Minimization Problem* (TLCMP) consists of determining the minimum number of crossing among the two layer of G such that nodes can be permuted in both layers and edges are drawn as straight lines. Any solution is uniquely determined by the permutations π_1 and π_2 of V_1 and V_2. The mathematical formulation for this problem is as follows:

$$\text{(TLCMP)} \qquad \min \sum_{(i,j),(k,l)\in A} c_{ijkl}$$

subject to:

$$-c_{ijkl} \leq y_{jl} - x_{ik} \leq c_{ijkl}, \qquad \forall\ (i,j),(k,l) \in A, j < l \tag{3.1}$$

$$1 - c_{ijkl} \leq y_{lj} + x_{ik} \leq 1 + c_{ijkl}, \qquad \forall\ (i,j),(k,l) \in A, j > l \tag{3.2}$$

$$0 \leq x_{ij} + x_{jk} - x_{ik} \leq 1, \qquad \forall\ 1 \leq i < j < k \leq n_1 \tag{3.3}$$

$$0 \leq y_{ij} + y_{jk} - y_{ik} \leq 1, \qquad \forall\ 1 \leq i < j < k \leq n_2 \tag{3.4}$$

$$x_{ij}, y_{ij}, c_{ijkl} \in \{0,1\}, \tag{3.5}$$

where, $x_{ij} = 1$ if if i precedes j, i.e. $\pi_1(i) < \pi_1(j)$, 0 otherwise. In the same way for the second level, $y_{ij} = 1$ if $\pi_2(i) < \pi_2(j)$, 0 otherwise. Furthermore, $c_{ijkl} = 1$ if and only if $x_{ik} \cdot y_{lj} = 1$ or $x_{ki} \cdot y_{jl} = 1$. The problem of minimizing the arc crossing between two layers is NP-complete [57].

3.2 Multi Layer Crossing Minimization Problem (MLCMP)

Let $G = (V, A)$ be a multi-layered graph, where V is partitioned into disjoint sets: $V_1 \cup V_2 \cup \cdots \cup V_p$ with $|V_i| = n_i$ such that for each edge $(u,v) \in A \Rightarrow u \in V_i$ and $v \in V_{i+1}, \forall i = 1,\ldots,p-1$. The set of edges, A, can be expressed as $A_1 \cup A_2 \cup \ldots A_{p-1}$, where A_i represents all connections between the layer i and $i+1$, for $i = 1,\ldots p-1$. The mathematical formulation for this problem is as follows:

$$\text{(MLCMP)} \qquad \min \sum_{t}^{p-1} \sum_{(i,j),(k,l)\in A_t} c^t_{ijkl}$$

subject to:

$$-c^t_{ijkl} \leq x^{t+1}_{jl} - x^t_{ik} \leq c^t_{ijkl}, \qquad \forall\ (i,j),(k,l) \in A_t, j < l \tag{3.6}$$

$$1 - c^t_{ijkl} \leq x^{t+1}_{lj} + x^t_{ik} \leq 1 + c^t_{ijkl}, \qquad \forall\ (i,j),(k,l) \in A_t, j > l \tag{3.7}$$

$$0 \leq x^t_{ij} + x^t_{jk} - x^t_{ik} \leq 1, \qquad \forall\ 1 \leq i < j < k \leq n_t \tag{3.8}$$

$$x^t_{ij}, c^t_{ijkl} \in \{0,1\}, \tag{3.9}$$

3.3 Incremental Graph Drawing Problem (IGDP)

Let $H = (V, A, k, L)$ be the corresponding hierarchical graph obtained from G through the procedure described in [57], where

- $L(v) : V \mapsto \{1, 2, \dots, k\}$, is a function which indicates the layer where the node v resides;
- $L_i = \{v \in V \mid L(v) = i\}$, for $i = 1, \dots, k$, indicates the set of nodes in the layer i;

Starting from H, an incremental graph $IH = (IV, IA, p, L)$ is defined as the graph resulting by adding to V a set of new nodes $\hat{V}$, with their corresponding arcs, such that $IV = V \cup \hat{V}$, $A \subseteq IA$ and $L(v) : IV \mapsto \{1, 2, \dots p\}$, with the value not changing for $v \in V$. A drawing for H is defined as $D = (H, \Phi)$, where $\Phi = \{\phi_1, \phi_2, \dots, \phi_k\}$, in particular ϕ_i establishes the ordering of the nodes in the layer i. With $\phi_i(j)$ we indicates the node in position j in the layer i. To find the position of a node v in the drawing D we can use $\pi(v)$ as in the case of the formulation for the TLCMP, in Paragraph 3.1, in this way if $\pi(v) = \phi_i(j)$ then $\pi(v) = j$.

Starting from D, the aim of the IGDP is to find a new drawing $ID = (IH, \Phi)$ for IH, in order to minimize the number of crossings while keeping the same relative ordering for the original node V. To better understand the goal of the problem: let u and v be two original nodes in V, such that $\pi(u) < \pi(v)$ in H, then in the new solution IH, the new positions for u and v, $\pi'(u)$ and $\pi'(v)$, must be such that $\pi'(u) < \pi'(v)$. To obtain a mathematical formulation for the IGDP, one can easily extend the model proposed for the MLCMP in Paragraph 3.2.

$$\text{(IGDP)} \qquad \min \sum_{t}^{p-1} \sum_{\substack{(i,j),(k,l)\in IA \\ \text{and} \\ L(i)=L(k)}} c^t_{ijkl}$$

s.t.:

$$
\begin{aligned}
& -c^t_{ijkl} \le x^{t+1}_{jl} - x^t_{ik} \le c^t_{ijkl}, && \forall\ (i,j),(k,l) \in IA, \\
& && \quad L(i) = L(k), j < l, \\
& 1 - c^t_{ijkl} \le x^{t+1}_{lj} + x^t_{ik} \le 1 + c^t_{ijkl}, && \forall\ (i,j),(k,l) \in IA, \\
& && \quad L(i) = L(k), j > l, \\
& 0 \le x^t_{ij} + x^t_{jk} - x^t_{ik} \le 1, && \forall\ i,j,k \in IV, i < j < k \\
& && \quad L(i) = L(j) = L(k), \\
& x^t_{ij} = 1, && \forall\ i,j \in V, L(i) = L(j), \\
& && \quad \pi(i) < \pi(j), \\
& x^t_{ij} = 0, && \forall\ i,j \in V, L(i) = L(j), \\
& && \quad \pi(i) > \pi(j), \\
& x^t_{ij}, c^t_{ijkl} \in \{0,1\},
\end{aligned}
$$

3.4 Constrained-Incremental Graph Drawing Problem (C-IGDP)

With a view to reach a widely acceptable compromise between the crossing minimization and preservation of the mental map, we proposed a constrained version of the IGDP. The C-IGDP requires an additional constraint that restricts the shift of the original nodes. Considering $n_1, n_2, \ldots, n_p$ be the numbers of original nodes in each layer, such that:

$$n_i = \sum_{\substack{v\in V \\ \text{and} \\ v\in L(v)}} 1, \ \forall\ i = 1, \ldots p. \tag{3.10}$$

Let $i \in V$ be an original node, and $\pi(i)$ its starting position, given a positive integer input value K such that $K \le n_i, \forall\ i = 1, \ldots, p$. Then, the new positions $\pi'(i)$

occupied by each original node in the solution must be such that:

$$\max\{1, \pi(i) - K\} \leq \pi'(i) \leq \min\{\pi(i) + K, n'_{L(i)}\}, \ \forall\, i \in V, \tag{3.11}$$

where $n'_{L(i)}$ represents the number of total nodes in the layer $L(i)$ considering also the incremental ones. Any original positions for the original nodes can be computed with:

$$\pi(i) = n_{L(i)} - \sum_{\substack{j \in V \\ \text{and} \\ j \in L(i)}} x_{ij} \tag{3.12}$$

Now, the mathematical formulation for the C-IGDP is obtained as follows:

$$\text{(C-IGDP)} \qquad \min \sum_{t}^{p-1} \sum_{\substack{(i,j),(k,l) \in IA \\ \text{and} \\ L(i)=L(k)}} c^t_{ijkl}$$

s.t.:

$$\begin{array}{ll}
-c^t_{ijkl} \leq x^{t+1}_{jl} - x^t_{ik} \leq c^t_{ijkl}, & \forall\ (i,j),(k,l) \in IA, \\
 & \quad L(i) = L(k), j < l, \\
1 - c^t_{ijkl} \leq x^{t+1}_{lj} + x^t_{ik} \leq 1 + c^t_{ijkl}, & \forall\ (i,j),(k,l) \in IA, \\
 & \quad L(i) = L(k), j > l, \\
0 \leq x^t_{ij} + x^t_{jk} - x^t_{ik} \leq 1, & \forall\ i,j,k \in IV, i < j < k \\
 & \quad L(i) = L(j) = L(k), \\
x^t_{ij} = 1, & \forall\ i,j \in V, L(i) = L(j), \\
 & \quad \pi(i) < \pi(j), \\
x^t_{ij} = 0, & \forall\ i,j \in V, L(i) = L(j), \\
 & \quad \pi(i) > \pi(j), \\
LB_i \leq \pi'(i) \leq UB_i, & \forall\ i \in V, \\
x^t_{ij}, c^t_{ijkl} \in \{0,1\}, \pi'(i) \in \mathbb{N}^+, &
\end{array}$$

where the penultimate constraint is called *position constraint*, and LB and UB represent a lower and an upper bound for the new position of each original node, in particular $LB_i = \max\{1, \pi(i) - K\}$ and $UB_i = \min\{\pi(i) + K, n'_{L(i)}\}$ for each $i \in V$.

3.5 Strategies for the C-IGDP

In this paragraph we will analyze four different strategies proposed for the C-IGDP and presented in [98]. The first three methods are based on the GRASP-framework, while the last is memory based and adopts a Tabu-like approach. Since all the mathematical formulations surveyed in the previous paragraphs have as objective the minimization of the number of crossing arcs, an explanation of what is mathematically considered as a crossing follows. Let (u_1, v_1) and (u_2, v_2) be two pairs of arcs. Naturally, both u_1 and u_2, and v_1, v_2, respectively reside on the same layer. Let us suppose that $u_1, u_2 \in L_i$ and $v_1, v_2 \in L_{i+1}$. Then, (u_1, v_1) and (u_2, v_2) are crossings if one of the following statements is true:

$$\pi(u_1) < \pi(u_2) \text{ and } \pi(v_1) > \pi(v_2),$$
$$\pi(u_1) > \pi(u_2) \text{ and } \pi(v_1) < \pi(v_2).$$

One of the most consolidated method in literature to minimize the number of crossings arcs in a HDAG is the *barycenter*. According to the barycenter method, the selected position for a node u in a level L_i, with $2 < i < k-1$, indicated with $bc(u, i)$, is given by:

$$bc(u, i) = \frac{\sum_{v \in BS(u)} \pi(v)}{2|BS(u)|} + \frac{\sum_{v \in FS(u)} \pi(v)}{2|FS(u)|},$$

where $BS(u) = \{v \in V \mid (v, u) \in A\}$ and $FS(u) = \{v \in V \mid (u, v) \in A\}$. Naturally, if u is a node in the first layer $BS(u) = \emptyset$ or if u is a node in the last layer $FS(u) = \emptyset$. All the strategies described below have been included in a multi-start framework, in which all the constructive phases are executed for a given number of iterations and for each iteration a new solution is built. Subsequently an improving method will be applied to each solution built after the constructive phase.

3.5.1 GRASP Proposals

GRASP is a well established iterative multi-start meta-heuristic method for difficult combinatorial optimization problems [39, 111]. The reader can refer to [47, 48] for a study of a generic GRASP meta-heuristic framework and its applications. Such method is characterized by the repeated execution of two main phases: a construction and a local search phase. The construction phase iteratively adds one component

```
GRASP($\alpha$)
$x^* \leftarrow$ NULL ;
$z(x^*) \leftarrow +\infty$ ;
while a stopping criterion is not satisfied do
    Build a greedy randomized solution $x$ ;
    $x \leftarrow$ LocalSearch($x$) ;
    if $z(x) < z(x^*)$ then
        $x^* \leftarrow x$ ;
        $z(x^*) \leftarrow z(x)$ ;
return $x^*$
```

Fig. 3.1 A generic GRASP for a minimization problem.

at a time to the current solution under construction. At each iteration, an element is randomly selected from a *restricted candidate list* (RCL), composed by the best candidates, according to some greedy function that measures the myopic benefit of selecting each element.

Once a complete solution is obtained, the local search procedure attempts to improve it by producing a locally optimal solution with respect to some suitably defined neighborhood structure. Construction and local search phases are repeatedly applied. The best locally optimal solution found is returned as final result. Figure 3.1 depicts the pseudo-code of a generic GRASP for a minimization problem.

GRASP construction version 1: C1

Given an instance $\mathcal{I}$ for the C-IGDP, the first version of the GRASP builds a new solution from scratch, considering all the nodes in $\mathcal{I}$ as nodes to insert in the solution. First of all, in its construction phase, the algorithm builds a list DEG^* containing all the original nodes with maximum degree:

$$DEG^* = \{v \in IV \mid deg(v) \geq deg(v'), \forall v' \in IV \text{ and } v' \neq v\},$$

where, $deg(v) = |BS(v)| + |FS(v)|$, is the sum of the in degree and the out degree of v. The algorithm selects randomly a node v from DEG^*, and put the node in its layer according the following rules:

- if v is an original node, then put it in a random position $\pi'(v)$ such that $LB_i \leq \pi'(v) \leq UB_i$ (according to the position constraint).
- otherwise, if v is a new node, then put it in a random position in the layer.

Once the first node has been inserted in the solution, the remaining nodes in IV are inserted in a Candidate List, CL. Staring from CL a Restricted Candidate List, RCL, is built as follows:

$$\mathsf{RCL} = \{v \mid v \in \mathsf{CL} \text{ and } \mathsf{deg}_{new}(v) \geq \alpha \mathsf{deg}_{\max}\},$$

where, α is an input parameter of the GRASP and it is a real value in the range $[0,1]$, $\mathsf{deg}_{new}(v)$ is given by the sum of the in and out degree of v according to the under-construction solution, while $\mathsf{deg}_{\max}$ is given by:

$$\mathsf{deg}_{\max} = \max_{v \in \mathsf{CL}} \mathsf{deg}_{new}(v).$$

Until the solution is not complete, a node v^* is randomly selected by the RCL and added to the solution in a position π' computed according to the barycenter method and taking into account if the v^* is an original or a new node. In fact, let $\mathsf{bc}(v^*, \mathsf{L}(v^*))$ be the barycenter computed for the node v^* according the under-construction solution, then two different approaches are allowed:

- if v^* is an original node, the nearest feasible position to $\mathsf{bc}(v^*, \mathsf{L}(v^*))$ is selected, tacking into account the position constraint and the remaining original nodes which have still to be inserted in the solution;
- otherwise, if v^* is a new node, the nearest feasible position to $\mathsf{bc}(v^*, \mathsf{L}(v^*))$ is selected, tacking into account only the remaining original nodes which have still to be inserted in the solution.

The pseudocode of the algorithm described above is reported in Figure 3.2

GRASP construction version 2: `C2`

The second constructive phase of the GRASP for the C-IGDP adopts the same methodology of the first one to find a position for the node in the new solution, using the barycenter method, but initially preserves the old positions for the original nodes. Specifically, let $\mathcal{I}$ an instance for the problem, the algorithm copies in the new solution all the original nodes in their old positions, and subsequently builds the CL containing only the new nodes. Then, starting from the candidate list, the algorithm builds the RCL according to the same rule adopted by the version 1. After that, iteratively, nodes are selected from RCL and inserted in the under-construction solution until a new solution is built. When a node v^* is randomly selected by

```
C1($\alpha$, $\hat{x}$)
Build the list of nodes with maximum degree, $DEG^*$ in $\hat{x}$;
$v \leftarrow$ select_node_randomly($DEG^*$);
$x^* \leftarrow$ add_node_to_solution($v$);
Build the candidate list, CL;
forall $v \in CL$ do
    if $deg_{new}(v) \leq \alpha deg_{max}$ then
        $RCL \leftarrow RCL \cup \{v\}$;
while $x^*$ is not complete do
    $v^* \leftarrow$select_node_randomly(RCL);
    $x^* \leftarrow$ add_node_to_solution($v^*$);
    rebuild RCL;
return $x^*$
```

Fig. 3.2 Constructive phase C1 for the C-IGDP.

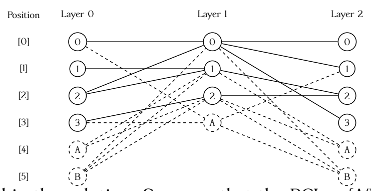
the RCL, the algorithm tries to put it in the new position $bc(v^*, L(v^*))$, then the following cases can occur:

- if the position $bc(v^*, L(v^*))$ is free, the node is added in that position,
- otherwise, the algorithm tries to shift up of one position all the nodes that occupies consecutive position starting from $bc(v^*, L(v^*))$.

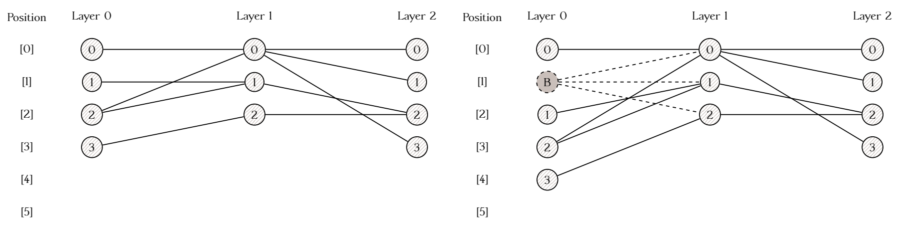
An example of construction phase C2 is depicted in Figure 3.3.

GRASP construction version 3: C3

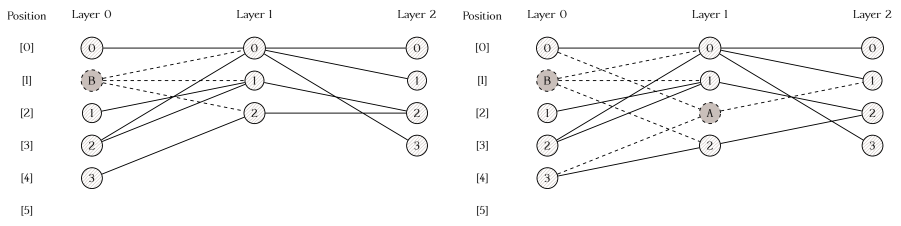
This last variant of the GRASP constructive phase adopts a greedy criterion for the eligibility of the nodes in the RCL which is significantly different from classic barycenter method, used in C1 and C2. The idea of proposing a new greedy criterion depends from the fact that although the barycenter is a consolidated method to minimize the number of crossings in a HDAG, it doesn't follows a real greedy criterion. In fact, during the construction of a solution for the C-IGDP , a purely greedy choice would insert a node in a position such that it involves a minimum increase of the number of crossings. Usually, the barycenter method does not guarantee this behavior, see the example in Figure 3.4. For this reason we introduce a new greedy criterion to build the RCL. As in the case of C2, also the C3 inserts all the original nodes in their initial positions and the candidate list will be composed by only new nodes. Let $\mathcal{I}$ an instance for the problem, then $CL = \{v \mid v \in IV \setminus V\}$. Let $C(v, p)$



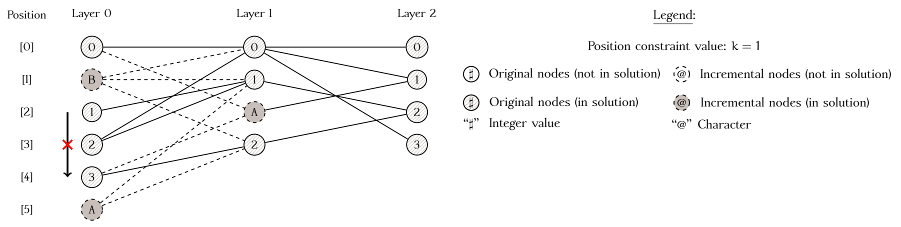
All the original nodes are copied in the solution. Suppose that the RCL = {A(L.1), B(L.0), B(L.2), A(L.0), A(L.2)} (where L indicates the layer). Randomly the node "B" in Layer 0 is selected to be inserted in solution, it's barycenter is given by $(\pi'(0)+\pi'(1)+\pi'(2))/3 = 1$. Then, the algorithm tries to insert B in that position. (See le legend bottom right, for complete information.)



Node "B" in Layer 0 is inserted in positon 1, all the nodes from the position 1 are shifted up. The insertion can be done, because the shifting of the original nodes do not violate the position constraint. After the insertion of B, the RCL = {A(L.1), B(L.2), A(L.0), A(L.2)}. Suppose that the node "A" in Layer 1 is selected to be inserted in solution.



The barycenter for A is given by $(\pi'(0)+\pi'(3))/2 = 2$ for the Layer 0, plus $(\pi'(1))/1 = 1$ for the Layer 2, all divided for 2, $(2+1)/2 = 1.5 \approx 2$. The node can be inserted in that position. The RCL is uptated, RCL = {B(L.2), A(L.0), A(L.2)}. Suppose that, the node "A" in Layer 0, is randomly selected to be inserted.


Position
Layer 0
Layer 1
Layer 2
[0]
[1]
[2]
[3]
[4]
[5]
Legend:
Position constraint value: k = 1
Original nodes (not in solution)
Incremental nodes (not in solution)
Original nodes (in solution)
Incremental nodes (in solution)
"♯" Integer value
"@" Character

The algorithm tries to inserts the node "A" in Layer 0, in position 2, the computed barycenter given by $(\pi'(1)+\pi'(2))/2 = 2$, but the original nodes cannot be shifted up otherwise they would violate the position constraint. Then, the node "A" is inserted in the first free position closest to its barycenter, in this case 5.

Fig. 3.3 Example of constructive phase C2.

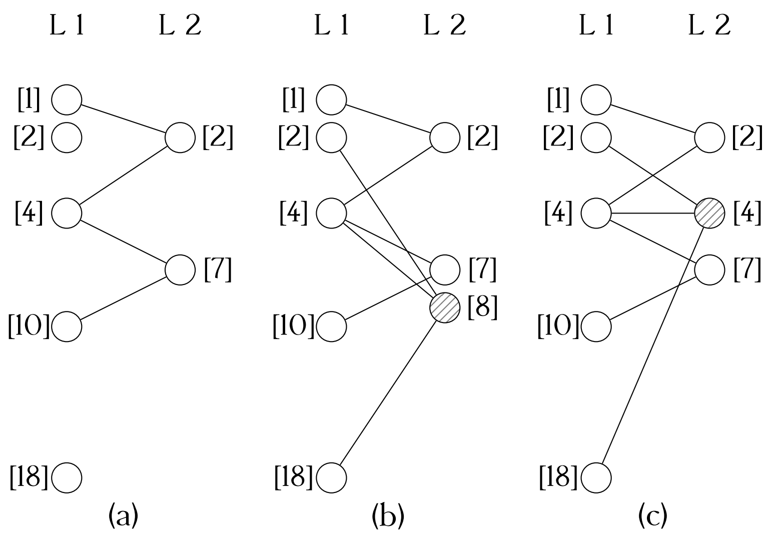


Fig. 3.4 Suppose that starting from Figure (a), a new node will be inserted in layer 2, which is connected with node in position [2], [4] and [18] in layer 1. According to the barycenter method the selected position for this will be the position [8] in layer 2. Analyzing the Figure (b) it can be noted that this insertion increase by 4 the number of crossings. While, an insertion of the new node in position [4], see Figure (c), increase by 3 the number of crossings.

be a function which assigns to each node v in CL a cost related to each possible position p in the layer where the node itself resides. Each of such costs is computed on the basis of the number of crossing generated in the under-construction solution by inserting the node v in position p. According to this function, if the cost of the under-construction solution is c and the algorithm inserts the node v in p the new cost of the partial solution will be $c + C(v, p)$. All the positions where a node in CL could be inserted will be examined and the best one will be selected, the elected position, then, will be p^* such that $C(v, p^*) = \min_p C(v, p)$. For all the nodes in CL the best position, and the RCL will be composed as follows:

$$\mathrm{RCL} = \{v \in \mathrm{CL} \mid C(v, p^*) \le \tau\},$$

where

$$\tau = \min_{v \in \mathrm{CL}} C(v, p^*) + \alpha\Big(\max_{v \in \mathrm{CL}} C(v, p^*) - \min_{v \in \mathrm{CL}} C(v, p^*)\Big).$$

Also in this case α is a real value in the range $[0, 1]$. In this way the, RCL is built according a greedy criterion which selects as candidate nodes those that achieve a

```
C3($\alpha$, $\hat{x}$)
$x^* \leftarrow$ copy_original_nodes($\hat{x}$);
$CL \leftarrow IV \setminus V$;
forall $v \in CL$ do
    compute $C(v, p^*)$ and $\tau$;
forall $v \in CL$ do
    if $C(v, p^*) \leq \tau$ then
        $RCL \leftarrow RCL \cup \{v\}$;
while $x^*$ is not complete do
    $v^* \leftarrow$select_node_randomly(RCL);
    $x^* \leftarrow$ add_node_to_solution($v^*$);
    recompute C and $\tau$;
    rebuild RCL;
return $x^*$
```

Fig. 3.5 Constructive phase C3 for the C-IGDP.

relative low increment to the cost of the solution. During the construction phase a node at time is randomly extracted for the RCL, added to the solution, and the process ends when the new solution is complete. When a node is selected to be added to the solution, then the selected position can be free or already occupied by another node. In the first case the node is added immediately to the solution, in the latter a shift up operation, as in the case of C2, is required. The pseudocode of the construction phase described above is reported in Figure 3.5.

3.5.2 Tabu approach

Tabu search, proposed by Glover [59], is one of the most well-known and most successful strategies proposed in recent decades and it achieved impressive practical successes in applications ranging from scheduling and computer channel balancing to cluster analysis and space planning and many others combinatorial optimization problems. It is a method for solving challenging problems where the goal is to identify the best ruling decision in order to maximize some measure of merit or to minimize some measure of demerit. The strength of this method lies in the ability to elude the local optima, in fact the Tabu approach enriches the behavior of a classical local search, whenever the search encounters a local optimum, by allowing non-improving moves. To do this, the method adopts a memory to prevent the repetition of actions already carried out in recent previous iterations.

For an exhaustive and detailed introduction and analysis of this algorithmic proposal refer to [60, 61, 63, 64].

Construction with memory: `C4`

Unlike the previous constructive phases this one adopts a mechanism based on memory to diversify the solutions in order to better investigate the feasible region. This phase is described as a Tabu-like approach because, during the construction of the solution, the actual choices of the algorithm are influenced by the previous ones. As in the case of `C2` and `C3` also this algorithm reports the original nodes in their initial position in the under-construction solution. Let $\mathcal{I}$ an instance for the problem, and CL the candidate list given by $\{v \mid v \in IV \setminus V\}$. The algorithm uses the same function $C(v, p)$ used in `C3` and an additional structure $F(v, p)$ which contains for each node $v \in CL$ and each possible positions p in the layer an integer value which specifies the number of times that the node v was inserted in the position p in the previous iterations. Now, a new function $\bar{C}(v, p) = C(v, p) + (\beta \cdot F(v, p))$ is defined, where β is a real number in the range $[0, 1]$. All the positions where a node in CL could be inserted will be examined and will be selected the best one p^* such that $\bar{C}(v, p^*) = \min_p \bar{C}(v, p)$. For all the nodes in CL will be computed the best position and the RCL is composed as follows:

$$\mathrm{RCL} = \{v \in CL \mid \bar{C}(v, p^*) \leq \tau\},$$

where τ is defined as in the case of `C3` but considering $\bar{C}$ instead of C. We notice that, unlike the GRASP approaches, in this case the parameters taken in input by the constructive phase are two instead of one, in fact in addition to α there is also β. All these parameters have been tuned during a training phase as will be reported in the paragraph where the computational results will be discussed.

3.5.3 Improvement phase for `C1`, `C2`, and `C3` : Local Search

The basic behavior of the local search used both in the GRASP and Tabu algorithm is the same. It is constituted by two sub-phases:

1. `Swapping` phase: where all the new nodes are swapped between other new nodes which reside in the same layer in order to try to improve the current solution,

2. `Sliding` phase: subsequents to the `Swapping` one, where the new nodes are slided up or down in the layer.

We underline that swapping and sliding are different operations in our algorithm. In fact, given two nodes i and j in two non-consecutive positions, $\pi'(i)$ and $\pi'(j)$, swapping i and j does not involve the nodes which occupy positions between $\pi'(i)$ and $\pi'(j)$. This operation is performed by an exchange of pointers and so it requires constant time, $\mathcal{O}(1)$. Differently, if a node i is slided from a position $\pi'(i)$ to a position $\pi'(i)+l$, then, the node i will be exchanges at first with the node in position $\pi'(i)+1$ then, with the node in position $\pi'(i)+2$, and so on, until the node i does not reach the position $\pi'(i)+l$. Anyway the two sub-phases introduced above are analyzed in details in the following paragraphs.

Swapping

Let ID be the drawing obtained after one of the constructive phase, and suppose that $\bar{V}_i = \{\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_t\}$ are the new nodes in a layer i of ID. During this phase of the local search, for each node $\bar{v}_j \in \bar{V}_i, 1 \leq j \leq t$, the algorithm tries to swap it with all the other new nodes $\bar{v}_l \in \bar{V}_i, 1 \leq l \leq t$ and $l \neq j$. At the end of the investigation of the neighborhood of the node $\bar{v}_j$, the best swap in terms of improvement in the objective function is performed, and the neighborhood of another new node is analyzed. This process is iterated for all the layers of the graphs and when the last layer is reached the algorithm restarts form the first one until the solution can not be further improved. The pseudocode of this phase is reported in Figure 3.6.

Sliding

For this second phase we designed two versions: best-improvement and first-improvement. Let ID be the graph obtained after the `Swapping` phase local search, and suppose that $\bar{V} = \{\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_t\}$ are the new nodes in a layer i of ID. During this phase of the local search, for each node $\bar{v}_j \in \bar{V}, 1 \leq j \leq t$, the algorithm tries to perform two actions:

- `Slide-up`: where the node $\bar{v}_j$ is slided up from its initial position in ID, $\pi'(\bar{v}_j)$, until the last position in the layer or until the furthest position allowed by the *position constraint*. During this process, the algorithm stores the position which leads to either the bests or the first improvement , π'^*. If a position π'^* is found, then the node $\bar{v}_j$ is slided according to π'^*, otherwise the algorithm applies `Slide-down` procedure,

```
Swapping($\hat{x}$)
best_cost $\leftarrow$ c($\hat{x}$);
$x^* \leftarrow x$;
improvement $\leftarrow$ true;
while improvement do
    improvement $\leftarrow$ false;
    forall layers: $i = 1 \dots k$ do
        forall nodes: $v \in L_i \mid v$ is a new node do
            best-swap $\leftarrow -1$;
            forall nodes: $\bar{v} \in L_i \mid \bar{v}$ is a new node do
                if $\bar{v} \neq v$ then
                    $x^* \leftarrow$ swap($v, \bar{v}$);
                    if c($x^*$) $<$ best_cost then
                        best_cost $\leftarrow$ c($x^*$);
                        best_swap $\leftarrow \bar{v}$;
                    $x^* \leftarrow$ swap($\bar{v}, v$);
            if best_swap $\neq -1$ then
                $x^* \leftarrow$ swap($v$,best_swap);
                improvement $\leftarrow$ true;
return $x^*$;
```

Fig. 3.6 Swapping phase local search for C-IGDP.

```
Slide-down-best-improvement($\hat{x}$, $\bar{v}_j$)
best_cost $\leftarrow c(\hat{x})$;
$x^* \leftarrow \hat{x}$;
best_position $\leftarrow \pi'(\bar{v}_j)$;
initial_position $\leftarrow \pi'(\bar{v}_j)$;
forall positions: p = initial_position−1 ... 0 do
    $v \leftarrow$ get_node_in (p);
    $x^* \leftarrow$ swap $(v, \bar{v}_j)$;
    if $x^*$ does not violate position constraint then
        if $c(x^*) <$ best_cost then
            best_cost $\leftarrow c(x^*)$;
            best_position $\leftarrow$ p;
    else
        $x^* \leftarrow$ swap $(\bar{v}_j, v)$;
        return best_position;
return best_position;
```

Fig. 3.7 Slide-down phase, best-improvement, local search for C-IGDP.

- Slide-down: the algorithm performs the same operations of Slide-up but in opposite way, i.e. the nodes are slided down in this case.

The pseudocodes of Slide-down in the best-improvement and first-improvement variants are reported in Figures 3.7 and 3.8. The operations Slide-up or Slide-down are performed for each new node in each layer and, as in the case of the Swapping-phase, when the last layer is reached the algorithm restarts from the first one until the solution can not be further improved.

A complete representation of the two phases of the local search is depicted in Figure 3.9.

3.5.4 Improvement phase for C4: Tabu Search

The strategy presented in paragraph 3.5.3 is the local search used as improvement phase after the construction phase for the GRASP approaches (C1, C1 and C3), while a variant with memory is considered for C4. In this variant the algorithm uses an additional structure $M(l, v)$ which stores for each node v the last iteration in which it changed position, and a constant real input user value, $tabu\_range$, used in the computation of a threshold, usually named - $tenure$ - in the tabù-search strategy, which specifies for how many iterations a node must be considered "tabù". In other words, if in the solution obtained at the iteration i the node v, in the level l, chose

```
Slide-down-first-improvement($\hat{x}$, $\bar{v}_j$)
cost ← $c(\hat{x})$;
$x^*$ ← $\hat{x}$;
initial_position ← $\pi'(\bar{v}_j)$;
forall positions: $p$ = initial_position−1 ... 0 do
    $v$ ← get_node_in ($p$);
    $x^*$ ← swap ($v, \bar{v}_j$);
    if $x^*$ does not violate position constraint then
        if $c(x^*) <$ cost then
            return $p$;
    else
        $x^*$ ← swap ($\bar{v}_j, v$);
        return initial_position;
return initial_position;
```

Fig. 3.8 `Slide-down` phase, first-improvement, local search for C-IGDP.

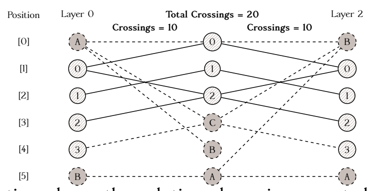
the last time to change position, then $M(l, v) = i$. Now, considering the iteration $i + j$, during the improvement phase the same node, v, can choose to move if and only if $((i + j) - M(l, v)) > tenure$. It is important to emphasize that in this phase a node which is not $tab\grave{u}$ must move also if the action does not improve the value of the current solution, in fact if an improvement action is not available, then the node will choose the movement which is the "least worst". As in the case of α and β, also $tabu\_range$ is a parameter that has been tuned during the training phase. Since the $tenure$ in our implementation was computed as follows:

$$tenure = \lfloor tabu\_range \cdot \lambda \rfloor$$

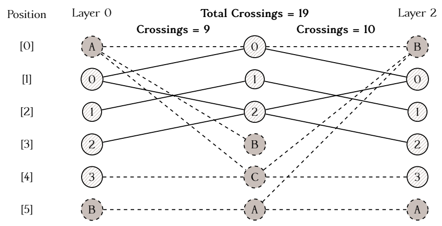
where, $\lambda = |\min_{j=1}^{|L|}\{v \in L_j \mid v \in \hat{V}\}|$, i.e. the minimum number of new nodes in a layer between all the layers in the instance, for $tabu\_range$ we considered the following range of possible choices to be appropriate: $\{0.25, 0.5, 0.75\}$.

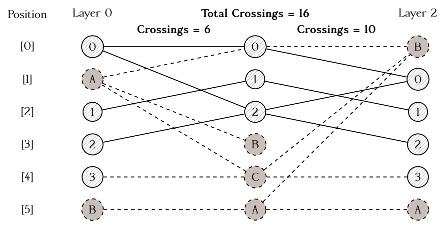
3.5.5 Post-optimization Phase: Path Relinking

In support of the construction and the improving phase we provided a post-optimization phase for our algorithms which is based on the Path Relinking. Path relinking (PR) is an approach suggested to integrate intensification and diversification [62]. During the execution this method explores trajectories which connect good-

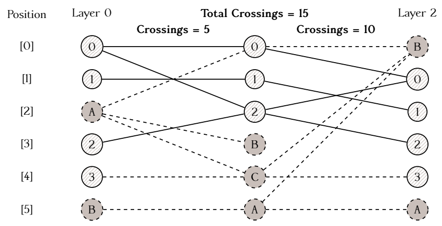


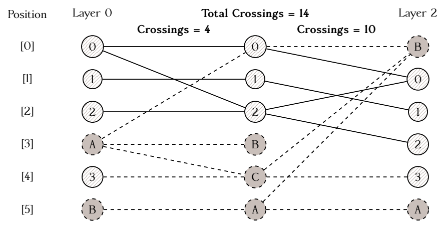
Suppose that after the construction phase, the solution above is computed. It's cost is 20. The phase 1 of the local-search plans to perform for each layers a sequence of swapping for the new nodes only in order to reduce the number of crossing edges.



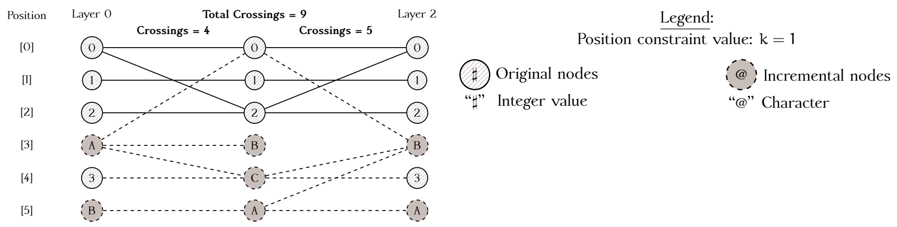


In the Layer 0, swapping the nodes "A" and "B" does not lead to an improvement. While in the Layer 1, the swap between the nodes "C" and "B" reduce the number of crossing from 20 to 19, others swaps in Layer 1 does not lead to an improvement, neither in Layer 2. The phase 1 ends when no others swaps reduce number of crossings edge. The phase 2, is formed by two sub-phase: slide-up or slide-down. Starting from Layer 0, the algorithm tries, at first, to slide the node "A" up. The first slide, with the node "0", reduce the numeber of crossings, form 19 to 16.





Continuing the slide-up of the node "A", the number of crossings is still improved, from 16 to 15. The shift-up continue untill the best slide is found for the node "A" (otherwise, in case of first-improvement local-search untill an improvement is found). The slide-down is performed only if the slide-up does not lead to improvement of the solution cost.



In Layer 1, there are not slides for the new nodes which improve the quality of the solution. While in the Layer 2 the slide-up of the node "B" leads a significant improvement.

Fig. 3.9 An application of the local search for the C-IGDP.

quality solutions, stored in a set called *Elite Set*. Starting from one of these solution, called *source solution*, a path to connect all the other solutions, called *target solutions*,is generated. In the exploration, the algorithm will find several *intermediate solutions* which reside in the neighboring space of the solutions in the *Elite Set*. Although several variants of PR exist, they differ from the way in which the path to connect solutions in the *Elite Set* is chosen. Given two solutions $x_1 = \{x_{11}, x_{12}, \dots, x_{1n}\}$ and $x_2 = \{x_{21}, x_{22}, \dots, x_{2n}\}$ if $c(x_1) < c(x_2)$, considering a minimization problem, then the solution x_1 is better than x_2, and the following path-finding strategies are possible:

- Forward PR: as source solution is considered x_2 while as target one x_1. A list of intermediate solutions $\hat{x}$ in the neighborhood of x_1 and x_2 are analyzed. At the beginning $\hat{x} = x_2$, and for each $j = 1 \dots n$, $\hat{x} = \hat{x} \setminus \{x_{2j}\} \cup \{x_{1j}\}$;
- Backward PR: in this case a path which starts form the worst solution to the best one is generated;
- Mixed PR: in this case two paths are generated, the first starting from x_1 to x_2 and the second one inthe opposite direction. At each iteration any solution makes a step in direction of the other one. When the two paths meet in an intermediate solution of the neighboring space of x_1 and x_2 the exploration ends.

Laguna and Marti [76] adapted the PR in a GRASP framework as an intensification method, anyway for an exhaustive list of examples of GRASP with PR refer to Resende and Ribeiro [110]. In our context we designed an algorithm for the PR which adopts the same strategy described in [116] and for our experiments we considered only the forward variant of PR. Given two solutions ID^s (source solution) and ID^t (target solution) for the C-IGDP, each single move generated by the PR consists in the replacement of an entire layer from a current solution with a layer from the target solution. The total numbers of solutions generated by the PR during the path from ID^s to ID^t is $\frac{k(k+1)}{2} - 1$. In Figure 3.10 an illustration of the algorithm. Starting for the source solution, ID^s, at the first step the algorithm generates three different solutions exchanging the layers 1, 2 and 3 of ID^s with the corresponding layers in the target solution ID^t. Then, the path continues from the intermediate solution ID^2 since it is the solution with the best cost and the algorithm generates other two solutions, ID^4 and ID^5. Finally, the path is completed from ID^4 to ID^t. During the execution the PR finds two improving solutions in the neighborhood of ID^s and ID^t, i.e. ID^2 and ID^4 both of which with cost 0. Also in the post-optimization phase are

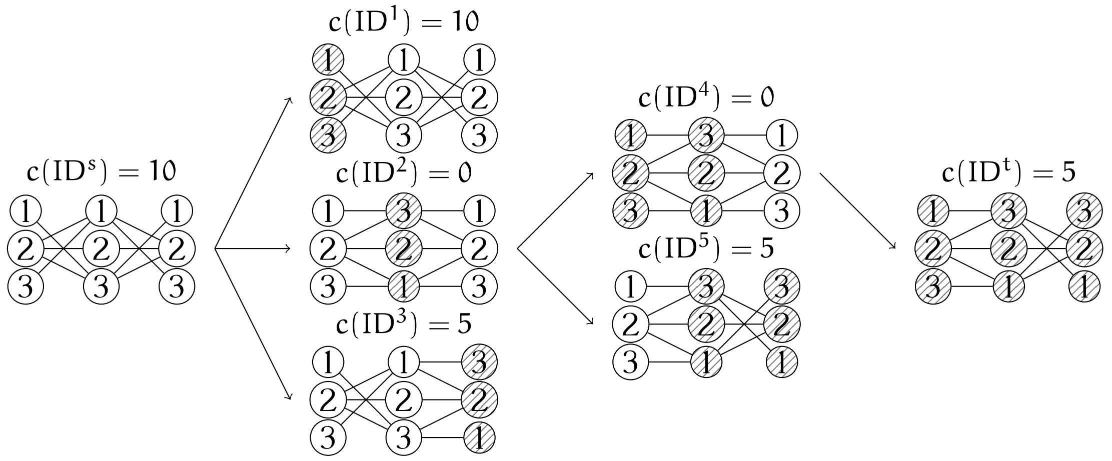


Fig. 3.10 A representation of the PR for the C-IGDP.

present some parameters which have been tuned in training phase. In particular, an integer value which specifies the dimension of the *Elite Set*, and a real value γ in the range $[0, 1]$ which denotes the degree of difference between two solutions. Specifically, let x and $\hat{x}$ be two solutions and π' and $\hat{\pi}'$ the corresponding positions of the nodes in x and $\hat{x}$, then, according to our criterion of diversification, their difference is given by summing up, for each layer $l \in L$, the number of nodes $v \in L_l$ such that $\pi'(v) \neq \hat{\pi}'(v)$ and dividing this sum for $|IV|$. After each improving phase, let $ES = \{x_1, \dots, x_t\}$ be the *Elite Set*, a solution $\hat{x}$ can be inserted in ES according to the following requirements:

- if $c(\hat{x}) \leq c(x_j)$ for each x_j in ES, in this case if the ES is full the solution with the worst value is replaced;
- else if ES is full, then
  - if $\exists x_j \in ES \mid c(x_j) > c(\hat{x})$ and $\forall x_j \in ES$ the *diversification degree* between x_j and $\hat{x}$ is greater than γ, then $\hat{x}$ can be inserted in ES in place of the worst solution;
  - else $\hat{x}$ can not be inserted in ES;
- else if $\forall x_j \in ES$ the *diversification degree* between $\hat{x}$ and x_j is greater than γ, then $\hat{x}$ can be inserted in ES;
- else $\hat{x}$ can not be inserted in ES.

3.5.6 Experimental Results

All the algorithms proposed were implemented in `C++`, compiled with `gcc 5.4.0`, and the experiments were conducted on a `Intel Corei7-4020HQ CPU @2.60Ghz x 8`. In the testing phase we compared the algorithms GRASP and `Tabu` among them and with an implementation in `CPLEX` of the model described in Paragraph 3.4 on a set of 250 instances. The set of instances used are available at http://www.optsicom.es/igdp, and as described in [116] the hierarchical graphs were generated following the guidelines in the literature Laguna et al. [77]. In each instance all the nodes for each layer have either an in-degree and out-degree of at least one. The nodes in the first layer have in-degree zero while the nodes in the last layer have out-degree zero. All the instances can be grouped in five categories which differ by number of layers: $2, 6, 13, 20, 50$. In particular, we considered 60 instances with 6, 13, and 20 layers and 10 instances with 50 layers. The total number of original nodes for each layers is between 5 and 30, while the total number of new nodes, for each layer $1 \leq i \leq k$, is given by $(1 + \delta)n_i$, where n_i is the number of original nodes in the layer and δ is a parameter chosen in the set $\{0.2, 0.6\}$. A summary about the characteristics of the instances is reported in Table 3.1, where the first column represents the Layers, the second the average of total nodes, the third one the average of new nodes and the last one the average of arcs for each instances grouped according the layer number.

Table 3.1 Summary of the instances properties.

L/s	n	new	m
2	26.60	7.90	88.85
6	126.60	37.60	421.17
13	312.30	92.90	997.20
20	490.45	146.05	1618.72
50	1291.42	325.31	3991.76

A key parameter in the experimentation phase is the separation value K, which establishes the distance between the initial position of the original nodes and the final one. Since the aim of our new formulation is to keep the original nodes close to their original position in order to preserve the user's mental map, we expected low K values, specifically $K = 1, 2, 3$. Therefore for each graph we generated three different instances, one for each K value. Note that, in few instances, the number of new nodes in a given layer is less than 2 or 3, then some value of K cannot be considered. In this way we obtained a *testing set* consisting of a total of 639 instances.

Preliminary Experiments

In a preliminary stage we carried out a training phase in order to find the best setting for the parameters of the algorithms. In this phase we have taken into account a sub-set with a 10% of the total instances in the *testing set*, precisely 62 instances, considering only the graph with 3, 6, 13, and 20 layers, we name it *training set*. The parameter tuned were:

- α for C1, C2, C3, and C4,
- β for C4,
- $tabu\_range$ for the tabu search,
- $|ES|$ and γ, for the post-optimization phase.

A summary with the data collected during the tuning of the parameter α for construction phase of the first GRASP algorithm is reported in Table 3.2. We considered four different values for α: 0.25, 0.5, 0.75 and $[0, 1]$, the last one corresponding to a totally random setup, for a total of 1000 iterations each. The column $\overline{C}$ indicates the average crossings between all the best solutions found by the algorithm for the instances considered in the training set, Dev the the standard deviation, $Best$ measures the number of times when an algorithm obtains better solution than the others, $Score$ measures the number of times when an algorithm does not obtain worse solutions than the others, while $t(s)$ expresses the average of the CPU running times in seconds. As can be observed in Table 3.2 the best setting for α is totally random.

Table 3.2 Tuning of the parameter alpha for C1.

α	$\overline{C}$	Dev	$Best$	$Rate$	$t(s)$
0.25	17711.800	0.0076	13	0.462	1.795
0.50	17747.100	0.0082	10	0.430	1.891
0.75	17780.100	0.0086	10	0.344	1.884
$[0, 1]$	17645.500	0.0002	55	0.957	1.837

In Table 3.3 the tuning of the parameter β used in C4 is reported. This construction phase takes as input two parameters, α and β. The best settings for this construction phase is given by $\alpha = [0, 1]$ and $\beta = [0, 1]$, even if there is a substantial balance between the different settings.

Table 3.3 Tuning of the parameter beta for C4.

α, β	$\overline{C}$	Dev	Best	Rate	t(s)
$[0,1], 0.25$	17420.100	0.0094	18	0.500	2.308
$[0,1], 0.50$	17399.900	0.0080	12	0.500	2.121
$[0,1], 0.75$	17361.400	0.0085	17	0.538	2.091
$[0,1], [0,1]$	17387.600	0.0084	20	0.532	2.185

A complete overview of the best setting for all the parameters used in our heuristics can be found in Table 3.4, and in Table 3.5 the different constructive phases with their best settings are compared.

Table 3.4 Best settings for all the parameters used by the heuristics for the C-IGDP.

constructive					Tabu Search	PR	
C1	C2	C3	C4				
α	α	α	α	β	tabu_range	$\|ES\|$	γ
-1.0	-1.0	-1.0	-1.0	-1.0	0.75	20	0.2

Table 3.5 Comparison constructive phases with their best settings.

methods	$\overline{C}$	Dev	Best	Rate	t(s)
C1	17645.500	0.0379	0	0.204	1.837
C2	17681.900	0.0377	0	0.156	1.973
C3	17395.300	0.0101	29	0.796	2.076
C4	17387.600	0.0027	36	0.860	2.185

Using these best settings, in a subsequent phase we carried out a testing phase in order to evaluate the performances of our proposals and establish a dominant strategy whenever possible. From this preliminary stage it emerges that, although they are more expensive according the computational point of view, both C3 and C4 return qualitatively better solutions.

Final Experiments

In the testing phase, for all the algorithms we considered 100 different constructions for all the instances taken into account, and a time-limit of 30 minutes. Furthermore, for each instance we considered three different choices for the separation value, $K = \{1, 2, 3\}$. In the experiments, we used the following denominations:

- GRASP1 is the GRASP with construction C1 and local search,
- GRASP2 is the GRASP with construction C2 and local search,
- GRASP3 is the GRASP with construction C3 and local search,
- Tabu is the algorithm with construction C4 and tabu search,
- GRASP3+PR, is the algorithm GRASP3 with Path Relinking.

In Table 3.6 only the instances where CPLEX found an optimal solution in the time-limit are summarized. In these tests the heuristics adopts only the improvement phase. In such table the first column express the average of the crossings, the second one the average of the gap percentage, i.e. the distance between the optimal solution and the solution obtained by the heuristics, the third column reports the number of times when an algorithm finds the best solution, the forth one the number of optimal solution found, and in the last one the average of the execution times in seconds is reported.

In the Tables 3.7 and 3.8, are presented the results, instance by instance, when CPLEX is not able to find an optimal solution within the time-limit. Comparing, respectively, CPLEX with the heuristics with only the improvement phase and the combination of improvement phase and post-optimization.The best values found are reported in red.

In Table 3.9, GRASP3, that we consider as our best algorithm taking into account the quality of the solutions and the running times, is compared with CPLEX and LocalSolver. LocalSolver (http://www.localsolver.com/home.html) is a new-generation meta-heuristic solver with hybrid mathematical programming solver, it combines the best of all optimization technologies to solve complex combinatorial problems. In this experimentsall the instances are included in the analysis , also when CPLEX did not obtain the optimal value.

Finally, in Table 3.10, we compare GRASP3+PR with LocalSolver on the largest instances, based on graphs with 50 layers, and report instance by instance the solution value and the computational time.

What clearly emerges from the computational testing is that, in general, all the algorithms proposed obtain solutions of extremely high quality. Considering only the constructive and the subsequent improvement phases, as shown in Table 3.6, the solutions obtain in average gaps much smaller than 1%. In fact in the worst case the higher gap is 0.24%, obtained by GRASP1 on instances with 20 layers but in correspondence of an average running time of 4.7 seconds against 107.44 exhibited

Table 3.6 CPLEX vs Heuristics, only optimal solutions.

methods	$\overline{C}$	% gap	Best	Opt	t(s)
	2 Layers instances				
CPLEX	2408.50	-	171	171	0.77
GRASP1	2408.53	0.00	167	167	0.09
GRASP2	2409.46	0.04	159	159	0.13
GRASP3	2409.19	0.03	161	161	0.16
Tabu	2408.70	0.01	150	150	0.55
	6 Layers instances				
CPLEX	9362.31	-	157	157	33.70
GRASP1	9369.82	0.08	73	73	1.62
GRASP2	9373.45	0.12	73	73	1.52
GRASP3	9367.46	0.05	81	81	2.35
Tabu	9365.84	0.04	55	55	5.76
	13 Layers instances				
CPLEX	17610.20	-	128	128	115.48
GRASP1	17643.90	0.19	19	19	3.03
GRASP2	17633.00	0.13	31	31	4.22
GRASP3	17637.80	0.16	27	27	4.13
Tabu	17631.00	0.12	11	11	18.49
	20 Layers instances				
CPLEX	27451.70	-	116	116	107.44
GRASP1	27517.20	0.24	10	10	4.70
GRASP2	27498.60	0.17	20	20	5.94
GRASP3	27507.40	0.20	20	20	7.82
Tabu	27495.90	0.16	2	2	30.68

by CPLEX. The goodness of our proposals is more evident when CPLEX is not able to find an optimal solution, as highlighted in Table 3.7 and 3.8. In fact in the first Table we have that all the algorithms exhibit a negative average gap which spans in percentage from -2.08 for GRASP1 to -2.14 for Tabu, and this gap increases further when we consider also the post-optimization phase, in the second Table. This certifies that the post-optimization phase significantly refines the solutions of the algorithms obtained after the improving phase.

In Table 3.9 we compare our best method, GRASP3, also with LocalSolver and it is clear how the difference between the solutions obtained, and the computational times, supports the goodness of our approach. Finally, in Table 3.10 the superiority of our proposals is confirmed also when we consider the largest instance, taking

Table 3.7 `CPLEX` vs Heuristics, no optimal solutions.

instance			`CPLEX`		`GRASP1`			`GRASP2`			`GRASP3`			`Tabu`		
L/s	dens	K	$\bar{C}$	t(s)	$\bar{C}$	% gap	t(s)	$\bar{C}$	% gap	t(s)	$\bar{C}$	% gap	t(s)	$\bar{C}$	% gap	t(s)
6	0.30	2	59687	1826.4	59595	-0.15	18.9	59524	-0.27	6.3	59535	-0.26	16.8	59590	-0.16	1.4
6	0.30	3	59748	1824.8	59381	-0.62	19.3	59363	-0.65	13.1	59366	-0.64	57.0	59368	-0.64	19.7
13	0.30	2	69119	1831.4	69008	-0.16	3.6	68996	-0.18	5.4	69082	-0.05	5.6	69012	-0.16	41.7
13	0.30	3	68881	1834.9	68593	-0.42	15.8	68411	-0.69	41.5	68565	-0.46	11.0	68511	-0.54	64.3
13	0.30	2	86298	1800.0	82095	-5.12	43.0	82092	-5.12	25.3	82097	-5.12	28.3	82094	-5.12	136.9
13	0.30	3	86298	1800.0	81803	-5.49	23.2	81801	-5.50	71.6	81816	-5.48	6.2	81802	-5.50	188.2
13	0.30	1	121127	1800.0	116500	-3.97	2.4	116532	-3.94	36.2	116531	-3.94	12.5	116524	-3.95	132.6
13	0.30	2	115360	1880.2	115375	0.01	33.9	115493	0.12	64.4	115451	0.08	17.3	115472	0.10	165.7
13	0.30	3	120180	1880.0	114716	-4.76	29.9	114706	-4.77	109.3	114496	-4.96	60.1	114460	-5.00	111.2
13	0.30	2	60284	1828.9	60277	-0.01	0.6	60280	-0.01	19.5	60302	0.03	18.9	60282	-0.00	57.0
13	0.30	3	59915	1831.2	59876	-0.07	10.9	59955	0.07	22.8	59963	0.08	40.2	59889	-0.04	137.6
13	0.30	2	63066	1827.8	62118	-1.53	2.5	62115	-1.53	4.8	62155	-1.47	27.0	62135	-1.50	133.4
13	0.30	3	62615	1827.8	61840	-1.25	33.0	61738	-1.42	47.3	61646	-1.57	30.7	61793	-1.33	7.4
13	0.30	2	70589	1838.0	69402	-1.71	18.2	69343	-1.80	14.3	69349	-1.79	34.6	69422	-1.68	97.5
13	0.30	3	71303	1839.9	69040	-3.28	23.9	68758	-3.70	7.4	68942	-3.42	75.8	68837	-3.58	27.7
20	0.17	3	55940	1848.7	56153	0.38	16.9	55994	0.10	77.0	56058	0.21	16.5	56098	0.28	167.6
20	0.17	3	31770	1826.9	32071	0.94	13.3	32037	0.83	13.6	31919	0.47	2.3	31929	0.50	79.9
20	0.30	2	78214	1849.8	77344	-1.12	5.2	77323	-1.15	40.5	77342	-1.13	60.3	77319	-1.16	77.6
20	0.30	3	77754	1849.5	76866	-1.16	46.6	76896	-1.12	1.4	76811	-1.23	19.0	76855	-1.17	4.8
20	0.30	2	188868	1800.0	177698	-6.29	53.0	177460	-6.43	54.6	177577	-6.36	129.9	177482	-6.42	139.3
20	0.30	3	188868	1800.0	176712	-6.88	83.8	176387	-7.08	96.3	176753	-6.85	183.6	176444	-7.04	44.4
20	0.30	1	137460	1800.0	132409	-3.81	11.8	132409	-3.81	57.3	132411	-3.81	8.7	132415	-3.81	232.1
20	0.30	2	137460	1800.0	131565	-4.48	77.8	131524	-4.51	81.8	131558	-4.49	52.2	131553	-4.49	290.3
20	0.30	3	137460	1800.0	130933	-4.98	27.7	130967	-4.96	88.0	130852	-5.05	97.8	130885	-5.02	269.9
20	0.30	2	77628	1853.6	73963	-4.96	20.0	73933	-5.00	20.1	74000	-4.90	29.8	73975	-4.94	3.7
20	0.30	3	73874	1852.5	73666	-0.28	21.7	73476	-0.54	16.4	73552	-0.44	28.3	73561	-0.43	127.6
20	0.30	2	103062	1877.9	98829	-4.28	18.9	98805	-4.31	59.8	98786	-4.33	5.3	98864	-4.25	105.0
20	0.30	3	99590	1882.5	98125	-1.49	53.0	98075	-1.54	65.0	98043	-1.58	7.7	98074	-1.55	86.2
20	0.30	2	50524	1825.5	50526	0.00	13.3	50523	-0.00	3.4	50524	0.00	21.0	50532	0.02	38.6
20	0.30	3	50201	1825.2	50148	-0.11	29.1	50156	-0.09	34.2	50124	-0.15	31.4	50129	-0.14	91.6
20	0.30	2	87356	1864.4	87086	-0.31	33.8	87043	-0.36	42.2	87048	-0.35	23.4	87037	-0.37	54.8
20	0.30	3	90878	1862.2	86715	-4.80	38.2	86719	-4.80	64.4	86718	-4.80	19.3	86690	-4.83	105.0
20	0.30	2	94124	1875.3	93972	-0.16	39.9	93973	-0.16	50.0	94009	-0.12	64.0	93979	-0.15	107.7
20	0.30	3	96522	1875.3	93295	-3.46	57.9	93284	-3.47	99.5	93283	-3.47	102.6	93314	-3.44	13.9
20	0.30	3	75612	1847.5	75288	-0.43	15.1	75202	-0.55	30.7	75204	-0.54	1.7	75084	-0.70	2.1
20	0.30	2	57637	1836.5	57415	-0.39	31.1	57389	-0.43	25.8	57402	-0.41	22.7	57395	-0.42	22.8
20	0.30	3	57511	1838.5	57220	-0.51	11.9	57213	-0.52	39.2	57258	-0.44	41.4	57217	-0.51	49.7
					avg	-2.08		avg	-2.14		avg	-2.13		avg	-2.14	

into account in this case only the comparison with `LocalSover`. In Figure 3.11 is depicted a test case where the performances of the GRASP `C3` with local search and a combination of local search and post-optimization are compared.

Table 3.8 CPLEX vs Heuristics+PR, no optimal solutions.

instance			CPLEX		GRASP1+PR			GRASP2+PR			GRASP3+PR			Tabu+PR		
L/s	dens	K	$\overline{C}$	t(s)	$\overline{C}$	% gap	t(s)	$\overline{C}$	% gap	t(s)	$\overline{C}$	% gap	t(s)	$\overline{C}$	% gap	t(s)
6	0.30	2	59687	1826.4	59590	-0.16	33.8	59524	-0.27	6.3	59521	-0.28	46.4	59527	-0.27	114.9
6	0.30	3	59748	1824.8	59364	-0.65	43.0	59362	-0.65	62.5	59361	-0.65	68.2	59362	-0.65	145.2
13	0.30	2	69119	1831.4	68999	-0.17	38.4	68996	-0.18	21.7	68998	-0.18	51.5	69012	-0.16	41.7
13	0.30	3	68881	1834.9	68478	-0.59	38.0	68398	-0.71	63.1	68525	-0.52	74.0	68408	-0.69	167.2
13	0.30	2	86298	1800.0	82095	-5.12	43.0	82090	-5.13	55.0	82086	-5.13	61.9	82085	-5.13	156.6
13	0.30	3	86298	1800.0	81798	-5.50	59.0	81752	-5.56	78.7	81795	-5.51	39.5	81791	-5.51	205.3
13	0.30	1	121127	1800.0	116500	-3.97	2.4	116516	-3.96	18.3	116520	-3.95	51.8	116496	-3.98	139.3
13	0.30	2	115360	1880.2	115375	0.01	33.9	115493	0.12	64.4	115385	0.02	88.8	115395	0.03	187.7
13	0.30	3	120180	1880.0	114452	-5.00	87.6	114636	-4.84	111.5	114400	-5.05	124.0	114460	-5.00	111.2
13	0.30	2	60284	1828.9	60274	-0.02	35.1	60275	-0.01	40.7	60291	0.01	44.7	60264	-0.03	139.0
13	0.30	3	59915	1831.2	59876	-0.07	10.9	59888	-0.05	59.0	59932	0.03	63.1	59845	-0.12	173.1
13	0.30	2	63066	1827.8	62118	-1.53	2.5	62115	-1.53	4.8	62070	-1.60	52.2	62080	-1.59	155.2
13	0.30	3	62615	1827.8	61740	-1.42	41.0	61738	-1.42	47.3	61646	-1.57	30.7	61724	-1.44	189.8
13	0.30	2	70589	1838.0	69393	-1.72	40.5	69331	-1.81	52.7	69349	-1.79	34.6	69343	-1.80	153.7
13	0.30	3	71303	1839.9	68807	-3.63	46.0	68758	-3.70	7.4	68701	-3.79	83.7	68836	-3.58	199.8
20	0.17	3	55940	1848.7	56052	0.20	59.9	55979	0.07	91.6	56015	0.13	101.1	56011	0.13	186.2
20	0.17	3	31770	1826.9	31866	0.30	30.4	31964	0.61	54.9	31919	0.47	50.1	31850	0.25	128.5
20	0.30	2	78214	1849.8	77323	-1.15	47.7	77320	-1.16	52.5	77327	-1.15	63.3	77319	-1.16	77.6
20	0.30	3	77754	1849.5	76713	-1.36	58.4	76726	-1.34	78.5	76775	-1.28	84.7	76711	-1.36	249.4
20	0.30	2	188868	1800.0	177463	-6.43	134.1	177451	-6.43	142.5	177477	-6.42	176.7	177459	-6.43	370.9
20	0.30	3	188868	1800.0	176434	-7.05	158.7	176387	-7.08	96.3	176424	-7.05	222.6	176327	-7.11	495.1
20	0.30	1	137460	1800.0	132409	-3.81	11.8	132409	-3.81	50.9	132411	-3.81	71.5	132415	-3.81	249.8
20	0.30	2	137460	1800.0	131488	-4.54	95.5	131514	-4.52	105.5	131501	-4.53	124.0	131492	-4.54	326.2
20	0.30	3	137460	1800.0	130791	-5.10	112.7	130793	-5.10	146.1	130848	-5.05	163.0	130842	-5.06	391.3
20	0.30	2	77628	1853.6	73923	-5.01	47.9	73916	-5.02	61.8	73914	-5.02	63.9	73940	-4.99	193.4
20	0.30	3	73874	1852.5	73536	-0.46	55.7	73451	-0.58	83.3	73515	-0.49	88.3	73474	-0.54	239.2
20	0.30	2	103062	1877.9	98829	-4.28	18.9	98798	-4.32	5.9	98746	-4.37	84.4	98772	-4.34	223.4
20	0.30	3	99590	1882.5	98085	-1.53	74.0	98075	-1.54	65.0	98034	-1.59	114.8	98017	-1.60	275.6
20	0.30	2	50524	1825.5	50526	0.00	30.3	50523	-0.00	3.4	50521	-0.01	38.8	50515	-0.02	124.4
20	0.30	3	50201	1825.2	50148	-0.11	29.1	50114	-0.17	48.8	50110	-0.18	53.2	50073	-0.26	155.7
20	0.30	2	87356	1864.4	87027	-0.38	58.1	87015	-0.39	64.9	87035	-0.37	66.7	87027	-0.38	219.3
20	0.30	3	90878	1862.2	86678	-4.85	70.2	86690	-4.83	90.2	86701	-4.82	91.4	86689	-4.83	290.0
20	0.30	2	94124	1875.3	93935	-0.20	69.1	93956	-0.18	83.2	93921	-0.22	89.6	93979	-0.15	107.7
20	0.30	3	96522	1875.3	93281	-3.47	79.1	93277	-3.48	115.9	93271	-3.49	116.3	93294	-3.46	296.6
20	0.30	3	75612	1847.5	75200	-0.55	54.0	75194	-0.56	78.2	75074	-0.72	86.0	75084	-0.70	2.1
20	0.30	2	57637	1836.5	57391	-0.43	42.1	57387	-0.44	49.6	57388	-0.43	50.0	57386	-0.44	182.0
20	0.30	3	57511	1838.5	57215	-0.52	43.7	57209	-0.53	66.7	57211	-0.52	70.2	57213	-0.52	224.8
					avg	-2.17		avg	-2.18		avg	-2.19		avg	-2.20	

Table 3.9 CPLEX and LocalSolver versus GRASP3

methods	$\overline{C}$	% gap	Dev	Best	Opt	t(s)
			2 Layers instances			
CPLEX	2408.50	-	0.000	171	171	0.77
GRASP3	2409.19	0.03	0.001	161	161	0.32
GRASP3+PR	2408.92	0.02	0.001	164	164	0.41
LocalSolver	2785.47	13.53	0.170	12	12	20.11
			6 Layers instances			
CPLEX	9995.70	-	0.000	157	157	56.24
GRASP3	9997.43	0.02	0.008	81	81	5.31
GRASP3+PR	9994.32	-0.01	0.000	100	98	5.43
LocalSolver	11024.90	9.34	0.166	0	0	62.43
			13 Layers instances			
CPLEX	23469.00	-	0.002	131	128	273.77
GRASP3	23319.40	-0.64	0.003	29	27	15.22
GRASP3+PR	23305.20	-0.70	0.002	54	44	15.32
LocalSolver	25530.20	8.07	0.162	0	0	162.06
			20 Layers instances			
CPLEX	37918.20	-	0.003	118	116	383.73
GRASP3	37522.40	-1.05	0.003	21	20	25.77
GRASP3+PR	37495.40	-1.13	0.002	49	29	28.32
LocalSolver	40954.10	7.41	0.147	0	0	328.77

Table 3.10 LocalSolver versus GRASP3+PR on largest instances.

K = 1				K = 2				K = 3			
GRASP3+PR		LocalSolver		GRASP3+PR		LocalSolver		GRASP3+PR		LocalSolver	
$\overline{C}$	t(s)	$\overline{C}$	t(s)	$\overline{C}$	t(s)	$\overline{C}$	t(s)	$\overline{C}$	t(s)	$\overline{C}$	t(s)
53048	49.92	59220	1019.69	52010	77.02	62547	1020.65	51413	99.90	63430	1019.81
47051	42.85	52037	1003.95	46152	66.63	54934	1004.17	45361	83.13	57458	1004.67
72018	64.73	78800	1058.08	70711	99.65	83344	1058.18	69940	132.82	84770	1058.01
101428	93.84	110649	1127.93	99767	142.35	108847	1128.66	98704	187.58	120328	1128.18
55112	51.73	61279	1016.93	54020	79.18	63984	1016.26	53298	106.53	65782	1016.16
89599	76.68	97372	1093.13	88247	115.74	100745	1093.07	87324	155.52	103617	1093.30
58829	54.32	65527	1023.40	57639	84.00	67388	1023.72	56783	111.33	70991	1024.39
79236	75.52	87359	1069.29	77716	111.58	91685	1070.21	76828	152.64	94363	1072.14
58147	55.45	65320	1028.31	57160	83.56	68692	1028.18	56527	106.78	70587	1027.96
84496	79.64	92960	1112.17	82910	123.70	96252	1113.10	81838	160.89	91291	1112.31

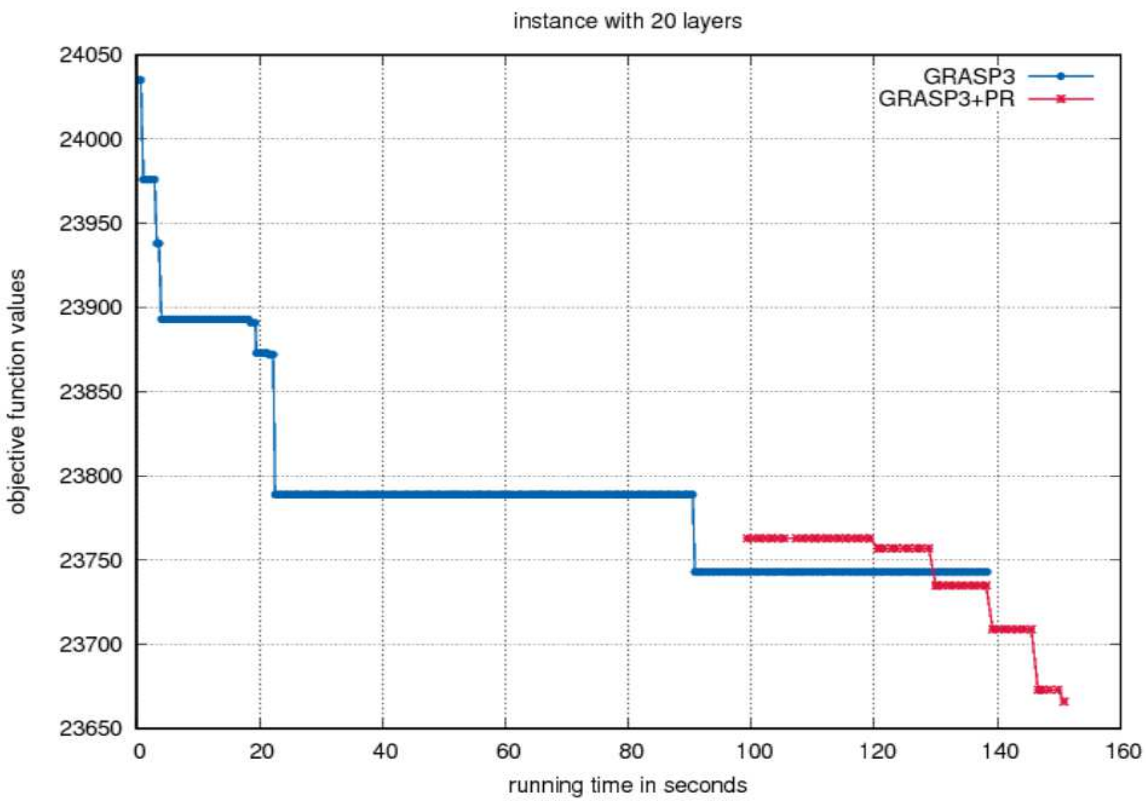


Fig. 3.11 GRASP C3, Local Search vs Path Relinking

Chapter 4

GRASP Approaches for Other Combinatorial Problems

In this chapter several GRASP approaches will be introduced in order to obtain efficient solutions to three combinatorial optimization problems: Maximum Cut-Clique (MCCP), p-Center (p-CP) and Minimum Cost Satisfiability Problem (MinCost-SAT). All the strategies proposed are based on the classical GRASP framework, with a constructive and an improvement phase. Furthermore, for the MinCost-SAT in [38] a new and original stopping rule has been proposed, which proved to be extremely effective in deciding when the search space has been explored extensively enough. All the designed algorithms were compared for each specific problem with state-of-the-art approaches.

4.1 Maximum Cut-Clique Problem (MCCP)

One of the classical problem in combinatorial optimization is the Max Clique Problem, which finds different applications in a very heterogeneous context. This problem, belonging to Karp's 21 NP-complete problems Karp [75], has many practical applications, ranging from pattern recognition in communication networks to software validation and verification, from computer vision to computational biology. Below are the notations used during the illustration of the problem. Let $G = (V, E)$ be a undirected graph, where:

- V: is the set of nodes,
- $E = \{(i, j) \mid i, j \in V\}$: is the set of edges,

- $d(v), \forall v \in V$: the degree of the node v,
- $\forall S \in V, G(S) = (S, E \cap S \times S)$: the sub-graph induced by S,
- $\overline{E} = \{(i,j) \in V \times V \mid (i,j) \in E \text{ and } i \neq j\}$: the complementary set of E,
- $\overline{G} = (V, \overline{E})$: the complementary graph of G,
- $N(i) = \{j \in V \mid (i,j) \in E\}$: the set of the nodes which are adjacent to i in G,
- $\overline{N}(i) = \{j \in V \mid (i,j) \in \overline{E}\}$: the set of the nodes which are adjacent to i in $\overline{G}$.

4.1.1 Maximum Clique Problem (MCP)

Given a undirected graph G, a *clique* C is a sub-set of V such that $G(C) = (C, E \cap C \times C)$ is complete, i.e. for each $i, j \in C, i \neq j$ then $(i,j) \in E \cap C \times C$. $\Gamma(G)$ indicates the set of all the cliques in G. The Maximum Clique Problem (MCP) consist in determining a clique of G with maximum cardinality. It is interesting to note that the MCP is closely correlated to other two well known combinatorial problems: Maximum Independent Set (MISP) and Minimum Vertex Cover (mVCP). An *independent set* is a sub-set of V such that for each $v \in V$ only one of the following is true:

1. $v \in S$,
2. $N(v) \cap S = \emptyset$.

The aim of the MISP is to find a maximum independent set of V. A *vertex cover* is a sub-set V' of V such that for each $(i,j) \in E$ only one of the following is true:

1. $u \in V'$,
2. $v \in V'$.

The aim of the mVCP is to find a vertex cover with minimum cardinality. It easy to check that these tree problems are equivalent, in fact C is a clique of G if and only if $\overline{C}$ is an independent set in $barG$ and if and only if $V \setminus V'$ is a vertex cover in $\overline{G}$. All three problems were proved to be NP-hard, Karp [75]. A simple formulation for the MCP was presented by Bomze et al. [12]:

$$(\text{MCP}) \qquad \min \sum_{i \in V} x_i$$

subject to:

$$x_i + x_j \leq 1, \qquad \forall\ (i,j) \in \overline{E}$$
$$x_i \in \{0,1\}, \qquad \forall\ i \in V$$

where $|V|$ binary decision variables are introduced such that:

$$x_i = \begin{cases} 1, & \text{if } i \text{ belongs to the clique,} \\ 0, & \text{otherwise.} \end{cases}$$

The second constraint expresses that for each edge in $\overline{G}$ at most one node belongs to the clique. An alternative formulation was proposed in Della Croce and Tadei [26]:

$$(\text{MCP}) \qquad \min \sum_{i \in V} x_i$$

subject to:

$$\sum_{j \in \overline{N}(i)} x_j \leq |\overline{N}(i)|(1 - x_i), \qquad \forall\ i \in V$$
$$x_i \in \{0,1\}, \qquad \forall\ i \in V.$$

A more recent formulation was proposed by Martins, where the following decision variables were introduced:

$$\forall q \in Q, w^q = \begin{cases} 1, & \text{if the clique has cardinality } q, \\ 0, & \text{otherwise,} \end{cases}$$

$$\forall i \in V \text{ and } \forall q \in Q, v_i^q = \begin{cases} 1, & \text{if the clique has cardinality } q \text{ and } i \text{ belongs to it,} \\ 0, & \text{otherwise} \end{cases}$$

where, $Q = \{q_{\min}, \dots, q_{\max}\}$ is a set of values of consecutive sizes. If the set of cliques with max cardinality in G is indicated with $\omega(G)$, then $q_{\min} \leq \omega(G) \leq q_{\max}$. The mathematical formulation is obtained as follows:

$$\text{(MCP)} \qquad \min \sum_{i \in V} x_i$$

subject to:

$$\begin{aligned}
&\sum_{j \in \overline{N}(i)} v_j^q \geq (q-1)v_i^q, && \forall\ i \in V \text{ and } q \in Q\\
&\sum_{i \in V} v_i^q = q w^q, && \forall q \in Q\\
&\sum_{q \in Q} w^q = 1, &&\\
&v_i^q \in \{0,1\}, && \forall\ i \in V \text{ and } q \in Q\\
&w^q \in \{0,1\}, && \forall\ q \in Q.
\end{aligned}$$

According to Martins [85] the three models presented above can be used according to the feature of the input graph. In fact, the first model is recommended for dense graph, the second one for sparse graph while the third one for intermediate cases.

4.1.2 MCCP: Mathematical Formulation

Let G be an undirected graph and C a clique of G. A cut-clique of C is indicated by:

$$E'(C) = \{(i,j) \in E \mid i \in C \text{ and } j \in V \setminus C\},$$

i.e. it is the set of all edges in the cut $(C, V \setminus C)$. In addition, if C is a singleton, namely $C = \{i\}$, $E'(C)$ is denoted by $E'(i)$. Since by definition every clique induces a complete sub-graph, denoting by $\Gamma(G)$ the set of all cliques of G, the MCCP can be stated as follows:

$$\text{(MCCP)} \qquad \max_{C \in \Gamma(G)} \left\{ \Big(\sum_{i \in C} |E'(i)| \Big) - |C|(|C|-1) \right\}$$

Although this problem received little attention in the scientific literature it is well known to be NP-hard [75, 85].

4.1.3 State of the Art Approaches for the MCCP

Although the MCCP is a problem very similar to a classic and well known problem as the MCP, the MCCP is a very little studied problem, in fact the only work presented in literature was [86]. In this work were proposed several adaptations of the *Iterated Local Search* (ILS) Framework. All the designed algorithms are based on the two main operations:

- *add*: is used to build a feasible solution in incremental way. If this process leads to an infeasible solution, then the second operation is used,
- *swap*: in the event that the first operation builds an infeasible solution, this latter tries to transform the current solution in a feasible one.

Two main strategies are developed depending by the way in which the nodes are selected in the *add* and *swap* operations, these strategies are *Random*-ILS (D-ILS) and *Degree*-ILS (R-ILS). In the matter the nodes are selected in a random way while in the latter strategy the nodes are selected according to their degree.

4.1.4 A GRASP for the MCCP

In order to tackle the MCCP we devised in [42] a meta-heuristic algorithm which, within a GRASP framework, hybridized a greedy randomized adaptive constructive phase with an improvement phase based on the Phased Local Search (PLS).

Construction Phase

The GRASP construction phase developed for the MCCP puts together the clique C one node at time, such nodes are selected from the RCL. The RCL in our case is made up of the nodes with highest degree among those neighboring the elements of C selected up to the previous step. The construction phase stops whenever there is no node $v \in V \setminus C$ such that $v \in N(u), \forall u \in C$. The pseudocode of the construction phase is shown in Figure 4.1.

Improvement Phase based on the Phased Local Search

Given the analogies with MCP, we decided to use as local search in our GRASP a suitably adapted version of the PLS proposed in [106], and depicted in Figure 4.2. Let $K_0(C)$ be the set of nodes of V connected to all the elements of the clique C and $K_1(C)$ be the *missing one* set, made up of nodes v connected to all but one elements of C, i.e,

$$K_0(C) = \{v \in V \mid C \cup \{v\} \text{ is a clique}\}; \qquad K_1(C) = \{v \in V \mid \exists!\, u \in C : u \notin \mathcal{N}(v)\}.$$

```
construction-GRASP-MCCP (G, $\alpha$)
$C \leftarrow \emptyset$;
$Q \leftarrow V$;
while $|Q| > 0$ do
    $d_{\min} \leftarrow \min\{d_{G(Q)}(u) \mid u \in Q\}$;
    $d_{\max} \leftarrow \max\{d_{G(Q)}(u) \mid u \in Q\}$;
    $RCL \leftarrow \{u \in Q \mid d_{G(Q)}(u) \geq d_{\min} + \alpha(d_{\max} - d_{\min})\}$;
    $u \leftarrow$select_node_randomly(RCL);
    $C \leftarrow C \cup \{u\}$;
    $Q \leftarrow Q \setminus \{u\} \setminus \{v \mid v \notin N(u)\}$;
```

Fig. 4.1 GRASP construction for MCCP.

The main feature of the PLS consists in the *phase* sub-routine which adopts three different selection criteria to move from the current clique C to an adjacent one. A clique C' is adjacent to C if it is obtained either by adding to C a node of $K_0(C)$ or swapping a node $v \in K_1(C)$ with the only $u \in C \setminus \mathcal{N}(v)$. Starting from the clique C built in the aforementioned construction phase, and up to an user-defined maximum selection number, the PLS repeatedly applies the *phase* sub-routine in the three following setups: 1) *Random select*: in which the nodes to be added, or respectively swapped, are randomly selected from $K_0(C)$ and $K_1(C)$; 2) *Degree select*: in which the nodes are selected from $K_0(C)$ and $K_1(C)$ according to their degree; 3) *Penalty select*: in which the nodes are chosen from $K_0(C)$ and $K_1(C)$ on the basis of a penalty function, which penalizes frequently selected nodes. As described in the paper of Martins et al. [86], we implemented two different versions of our algorithm: GRASP-PLS and GRASP-PLS$_p$. The latter implementation exploits the following proposition proposed in [86]:

Proposition 4. *Given a clique* C *of* G *and a node* $i \in V \setminus C$. *If* $|E'(i)| < 2|C|$ *then the node* i *will not belong to* $K_0(C)$, *and then it does not belong to any one optimal solution with a clique of cardinality* $|C| + 1$ *or higher. Furthermore, if* $|E'(i)| = 2|C|$ *if the node* i *is inserted in* C *then the cardinality of the cut given by* C *will not increase.*

The results obtained by the Proposition 4 allows to recognize nodes whose addition (or swap, in case of nodes of $K_1(C)$) generates an improvement in the value of the current cut-clique.

```
PLS-for-MCCP (C, max_sel)
C* ← C ;
for i = 1 to max_sel do
    C ← Random_selection(n_iter_1) ;
    C ← Degree_selection(n_iter_2) ;
    C ← Penalty_selection(n_iter_3) ;
update_solution(C, C*) ;
return C*
```

Fig. 4.2 The phased local search procedure used in our GRASP.

4.1.5 Experimental Results

All the algorithms considered in the computational experiments were implemented in C++ and compiled with g++ 5.4.0 with the flag -std=c++11. All tests were run on an Intel Core i7-4720HQ CPU @2.60GHz $\times$ 8. The instances tackled in our experiments are taken from the second DIMACS implementation challenge, and are the same used in [86]. For the sake of scientific fairness, for our testing we implemented in C++ the algorithms of [86] (originally implemented in FORTRAN). Each algorithm was executed for 100 runs. The results obtained, reported in Table 4.1, show how the proposed algorithm outperforms the previous methods on the vast majority of the benchmark set. The occurrences in which one algorithm yields either the absolute best or tied-best objective function mean value and the best computation time are reported in bold.

4.2 p-Center Problem (p-CP)

The p-Center Problem (p-CP) is one of the best-known location problems first introduced in 1964 by Hakimi [68]. It consists of locating p facilities and assigning clients/users to them in order to minimize the maximum distance between a client and the facility to which the client is assigned. The p-CP arises in many different real-world contexts, whenever one designs a system for public facilities, such as schools or emergency services. Formally, we are given a complete undirected edge-weighted bipartite graph $G = (V \cup U, E, c)$, where

- $V = \{1, 2, \dots, n\}$ is a set of n potential locations for facilities;
- $U = \{1, 2, \dots, m\}$ is a set of m clients or demand points;
- $E = \{(i, j) | \ i \in V, \ j \in U\}$ is a set of $n \times m$ edges;

Table 4.1 Average of the solution values (avg-z) and execution time (avg-t) of the algorithms. The p subscript specifies when the implementation uses Proposition 4.

	D-ILS		D-ILS$_p$		R-ILS		R-ILS$_p$		GRASP-PLS		GRASP-PLS$_p$	
instance	avg-z	avg-t	avg-z	avg-t	avg-z	avg-t	avg-z	avg-t	avg-z	avg-t	avg-z	avg-t
C125.9	2766	0.176	2766	0.176	2766	0.189	2766	0.186	**2766**	**0.152**	2766	0.161
C250.9	8123	0.277	8123	0.279	8119.45	0.284	8119.68	0.285	**8123**	**0.25**	8123	0.264
C500.9	22367.3	0.476	22353.5	0.478	22325.2	0.487	22334.5	0.487	**22616.6**	**0.43**	22578.8	0.453
C1000.9	55371.7	0.826	55263.2	0.826	54675.6	0.854	54725.7	0.849	**56500.5**	**0.731**	56499	0.794
C2000.5	16036.9	2.86	16000.1	2.86	15890.4	3.12	15889.6	3.1	15973.6	2.25	15939.2	2.36
C2000.9	127363	1.56	127231	1.55	125979	1.6	125998	1.61	**130832**	**1.34**	130730	1.48
C4000.5	34268.4	5.63	34263.2	5.68	34011.9	6.08	34059.2	6.05	34194.5	4.46	34182	4.74
keller4	1140	0.169	1140	0.17	1140	0.199	1140	0.199	**1140**	**0.16**	1140	0.168
keller5	15030.5	0.48	15020.7	0.481	15156.8	0.54	15147.8	0.542	**15184**	**0.445**	15184	0.487
keller6	141174	1.61	140868	1.61	139669	1.79	142883	1.78	**147728**	**1.42**	147452	1.64
MANN-a9	**412**	**0.658**	412	0.664	412	0.73	412	0.732	412	0.679	412	0.688
MANN-a27	31080.5	0.28	31077	0.289	31136.4	0.268	31137.5	0.26	**31254.4**	**0.234**	31249.8	0.247
MANN-a45	232838	0.787	232900	0.801	232813	0.732	232786	0.704	**234382**	**0.601**	234382	0.643
MANN-a81	2418810	2.66	2418990	2.7	2417850	2.9	2417860	2.76	2418930	3.01	2418930	2.86
p-hat300-1	789	0.449	789	0.449	789	0.51	789	0.511	**789**	**0.34**	789	0.354
p-hat300-2	4637	0.379	4637	0.381	4637	0.417	4637	0.417	**4637**	**0.327**	4637	0.342
p-hat300-3	7740	0.333	7740	0.333	7740	0.355	7740	0.354	**7740**	**0.296**	7740	0.314
p-hat500-1	1621	0.682	1621	0.684	1621	0.776	1621	0.78	**1621**	**0.537**	1621	0.563
p-hat500-2	11539	0.541	11539	0.543	11539	0.587	11539	0.586	**11539**	**0.493**	11539	0.52
p-hat500-3	18859	0.456	18859	0.457	18858.8	0.49	18858.8	0.49	**18859**	**0.422**	18859	0.453
p-hat700-1	2606	0.92	2606	0.921	2606	1.05	2606	1.05	**2606**	**0.729**	2606	0.766
p-hat700-2	20425	0.701	20425	0.703	20425	0.763	20425	0.763	**20425**	**0.644**	20425	0.684
p-hat700-3	33480	0.587	33480	0.589	33479.4	0.629	33479.4	0.626	**33480**	**0.548**	33480	0.59
p-hat1000-1	3556	1.27	3556	1.27	3556	1.45	3556	1.46	**3556**	**1.01**	3556	1.06
p-hat1000-2	31174	0.973	31174	0.977	31172.9	1.05	31173.2	1.06	**31174**	**0.895**	31174	0.952
p-hat1000-3	53259	0.796	53259	0.797	53259	0.854	53259	0.853	**53259**	**0.744**	53259	0.811
p-hat1500-1	6018	1.87	6018	1.87	6018	2.14	6018	2.15	**6018**	**1.5**	6018	1.58
p-hat1500-2	67486	1.32	67486	1.31	67480.5	1.42	67478.6	1.43	**67486**	**1.24**	67486	1.34
p-hat1500-3	112867	1.03	112864	1.03	112842	1.12	112841	1.12	**112873**	**1.01**	112873	1.12

- $c : E \mapsto \mathbb{R}^+ \cup \{0\}$ is a function that assigns a nonnegative distance c_{ij} to each edge $(i, j) \in E$.

The aim of the problem is to find a subset $P \subseteq V$ of size p such that its weight, defined as

$$\mathbb{C}(P) = \max_{i \in U} \min_{j \in P} c_{ij} \tag{4.1}$$

is minimized. The minimum value is called *radius*. Although it is not a restrictive hypothesis, we consider the special case where $V = U$ is the node set of a complete graph $G = (V, E)$, each distance c_{ij} represents the length of a shortest path between nodes i and j ($c_{ii} = 0$), and hence the triangle inequality is satisfied.

In 1979, Kariv and Hakimi [73] proved that the problem is NP-hard, even in the case where the input instance has a simple structure (e.g., a planar graph of maximum node degree 3). In 1970, Minieka [91] designed the first exact method for the p-CP viewed as a series of set covering problems. His algorithm iteratively chooses a threshold r for the radius and checks whether all clients can be covered within distance r using no more than p facilities. If so, the threshold r is decreased; otherwise, it is increased. Inspired by Minieka's idea, in 1995 Daskin [23] proposed a recursive bisection algorithm that systematically reduces the gap between upper and lower bounds on the radius. More recently, in 2010 Salhi and Al-Khedhairi [114] proposed a faster exact approach based on Daskin's algorithm that obtains tighter upper and lower bounds by incorporating information from a three-level heuristic that uses a variable neighborhood strategy in the first two levels and at the third level a perturbation mechanism for diversification purposes.

Recently, several facility location problems similar to the p-center have been used to model scenarios arising in financial markets. The main steps to use such techniques are the following: first, to describe the considered financial market via a correlation matrix of stock prices; second, to model the matrix as a graph, stocks and correlation coefficients between them are represented by nodes and edges, respectively. With this idea, Goldengorin et al. [67] used the p-median problem to analyze stock markets. Another interesting area where these problems arise is the manufacturing system with the aim of lowering production costs [66].

Due to the computational complexity of the p-CP, several approximation and heuristic algorithms have been proposed for solving it. By exploiting the relationship between the p-CP and the dominating set problem [71, 83], nice approximation results were proved. With respect to inapproximability results, Hochbaum and Shmoys [71] proposed a 2-approximation algorithm for the problem with triangle inequality, showing that for any $\delta < 2$ the existence of a δ-approximation algorithm would imply that $P = NP$.

Although interesting in theory, approximation algorithms are often outperformed in practice by more straightforward heuristics with no particular performance guarantees. Local search is the main ingredient for most of the heuristic algorithms that have appeared in the literature. In conjunction with various techniques for escaping local optima, these heuristics provide solutions which exceed the theoretical upper bound of approximating the problem and derive from ideas used to solve the p-median problem, a similar NP-hard problem [74]. Given a set F of m potential facilities, a set U of n users (or customers), a distance function $d : U \times F \mapsto \mathbb{R}$, and a constant $p \leq m$, the p-median problem is to determine a subset of p facilities to open so as to minimize the sum of the distances from each user to its closest open facility. For the p-median problem, in 2004 Resende and Werneck [112] proposed a multi-start heuristic that hybridizes GRASP with Path-Relinking as both, intensification and post-optimization phases. In 1997, Hansen and Mladenovic [69] proposed three heuristics: `Greedy`, `Alternate`, and `Interchange`(node substitution). To select the first facility, `Greedy` solves a 1-center problem. The remaining $p - 1$ facilities are then iteratively added, one at a time, and at each iteration the location which most reduces the maximum cost is selected. In [34], Dyer and Frieze suggested a variant, where the first center is chosen at random. In the first iteration of `Alternate`, facilities are located at p nodes chosen in V, clients are assigned to the closest facility, and the 1-center problem is solved for each facility's set of clients. During the subsequent iterations, the process is repeated with the new locations of the facilities until no more changes in assignments occur. As for the `Interchange` procedure, a certain pattern of p facilities is initially given. Then, facilities are moved iteratively, one by one, to vacant sites with the objective of reducing the total (or maximum) cost. This local search process stops when no movement of a single facility decreases the value of the objective function. A multi-start version of `Interchange` was also proposed, where the process is repeated a given number of times and the best solution is kept. The combination of `Greedy` and `Interchange` has been most often used for solving the p-median problem. In 2003, Mladenovic et al. [93] adapted it to the p-CP and proposed a Tabu Search (TS) and a Variable Neighborhood Search (VNS), i.e., a heuristic that uses the history of the search in order to construct a new solution and a competitor that is not history sensitive, respectively. The TS is designed by extending `Interchange` to the chain-interchange move, while in the VNS, a perturbed solution is obtained from the incumbent by a k-interchange operation and `Interchange` is used to improve it. If a better solution than the incumbent is found, the search is re-centered around it. In 2011, Davidovic et al. [24] proposed a Bee Colony algorithm, a random search population-based technique, where an artificial system composed of a number of precisely defined agents, also called individuals or artificial bees.

To the best of our knowledge, most of the research effort devoted towards the development of meta-heuristics for this problem has been put into the design of efficient local search procedures. The purpose of this article is propose a new local search and to highlight how its performances are better than best-known local search proposed in literature (Mladenovic et al. [93] local search based on VNS strategy), both in terms of solutions quality and convergence speed.

4.2.1 A GRASP for the p-CP

For the p-center problem, we propose in [44] a new smart local search based on the critical node concept and embed it in a GRASP framework. This strategy obtained remarkable results on the well-known p-CP benchmark instances, while in [45] we extends this approach also considering optical networks.

Construction phase

Starting from a partial solution made of $1 \leq \mathtt{randElem} \leq p$ facilities randomly selected from V, our GRASP construction procedure iteratively selects the remaining $p - \mathtt{randElem}$ facilities in a greedy randomized fashion. The greedy function takes into account the contribution to the objective function achieved by selecting a particular candidate element. In more detail, given a partial solution P, $|P| < p$, for each $i \in V \setminus P$, we compute $w(i) = \mathbb{C}(P \cup \{i\})$. The pure greedy choice would consist in selecting the node with the smallest greedy function value. This procedure instead computes the smallest and the largest greedy function values:

$$z_{\min} = \min_{i \in V \setminus P} w(i); \quad z_{\max} = \max_{i \in V \setminus P} w(i).$$

Then, denoting by $\mu = z_{\min} + \beta(z_{\max} - z_{\min})$ the cut-off value, where β is a parameter such that $\beta \in [0, 1]$, a *restricted candidate list* (RCL) is made up of all nodes whose greedy value is less than or equal to μ. The new facility to be added to P is finally randomly selected from the RCL. The pseudocode is shown in Figure 4.3, where $\alpha \in [0, 1]$.

```
construction-GRASP-pCP (G, p, α, β)
$P \leftarrow \emptyset$ ;
randElem $\leftarrow \lfloor \alpha \cdot p \rfloor$ ;
for $k = 1, \dots,$ randElem do                                  // random component
    f ←select_node_randomly($V \setminus P$);
    $P \leftarrow P \cup \{f\}$ ;
while $|P| < p$ do
    $z_{\min} \leftarrow +\infty$ ;
    $z_{\max} \leftarrow -\infty$ ;
    for $i \in V \setminus P$ do
        if $z_{\min} > \mathbb{C}(P \cup \{i\})$ then
            $z_{\min} \leftarrow \mathbb{C}(P \cup \{i\})$ ;
        if $z_{\max} < \mathbb{C}(P \cup \{i\})$ then
            $z_{\max} \leftarrow \mathbb{C}(P \cup \{i\})$ ;
    $\mu \leftarrow z_{\min} + \beta(z_{\max} - z_{\min})$ ;
    RCL $\leftarrow \{i \in V \setminus P \mid \mathbb{C}(P \cup \{i\}) \leq \mu\}$ ;
    f ←select_node_randomly(RCL) ;
    $P \leftarrow P \cup \{f\}$;
return P;
```

Fig. 4.3 GRASP construction for p-CP.

Improvement Phase: Plateau Surfer, a New Local Search Based on the Critical Node Concept

Given a feasible solution P, the `Interchange` local search proposed by Hansen and Mladenovic [69] consists in swapping a facility $f \in P$ with a facility $\bar{f} \notin P$ which results in a decrease of the current cost function. Especially in the case of instances with many nodes, we have noticed that usually a single swap does not strictly improve the current solution, because there are several facilities whose distance is equal to the radius of the solution. In other words, the objective function is characterized by large plateaus and the `Interchange` local search cannot escape from such regions. To face this type of difficulties, inspired by Variable Formulation Search [94, 102], we have decided to use a refined way for comparing between valid solutions by introducing the concept of *critical node*. Given a solution $P \subseteq V$, let $\delta_P : V \mapsto \mathbb{R}^+ \cup \{0\}$ be a function that assigns to each node $i \in V$ the distance between i and its closest facility according to solution P. Clearly, the cost of a solution P can be equivalently written as $\mathbb{C}(P) = \max\{\, \delta_P(i) \colon i \in V\,\}$. We also give the following definition:

Definition 1 (Critical node). *Let* $P \subseteq V$ *be a solution whose cost is* $\mathbb{C}(P)$. *For each node* $i \in V$, i *is said to be a* critical node *for* P, *if and only if* $\delta_P(i) = \mathbb{C}(P)$.

In the following, we will denote with $\max_{\delta_P} = |\{ i \in V\colon \delta_P(i) = \mathbb{C}(P) \}|$ the number of nodes whose distance from their closest facility results in the objective function value corresponding to solution P. We define also the comparison operator $<_{cv}$, and we will say that $P <_{cv} P'$ if and only if $\max_{\delta_P} < \max_{\delta'_P}$.

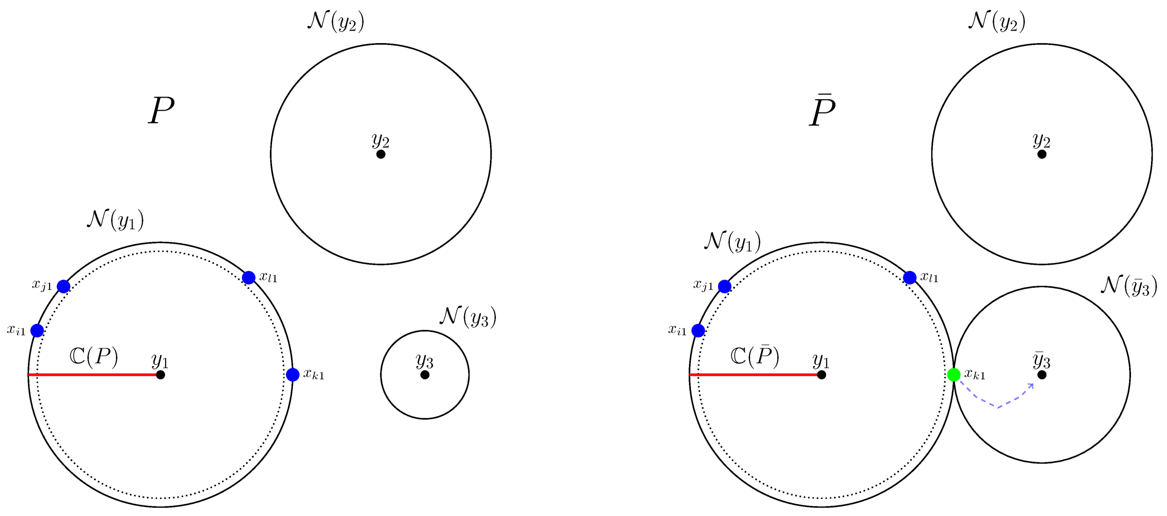


Fig. 4.4 An example of how the local search works. In this case, the algorithm switches from solution P to solution $\overline{P}$. In $\overline{P}$, a new facility $\overline{y}_3$ is selected in place of y_3 in P, $\overline{y}_3$ attracts one of the "critical nodes" from the neighborhood of the facility y_1. Although the cost of the two solutions is the same, the algorithm selects the new solution $\overline{P}$ because $\max_{\delta_{\overline{P}}} < \max_{\delta_P}$.

The main idea of our *plateau surfer* local search is to use the concept of critical node to escape from plateaus, moving to solutions that have either a better cost than the current solution or equal cost but less critical nodes. Figure 4.4 shows a simple application of the algorithm, while in Figures 4.5 and 4.6, for four benchmark instances, both Mladenovic's local search and our local search are applied once taken as input the same starting feasible solution. It is evident that both the procedures make the same first moves. However, as soon as a plateau is met, Mladenovic's local search ends, while our local search is able to escape from the plateau moving to other solutions with the same cost value, and restarting the procedure from a new solution that can lead to a strict cost function improvement.

Let us analyze in more detail the behavior of our local search, whose pseudocode is reported in Figure 4.7. The main part of the algorithm consists in the portion of

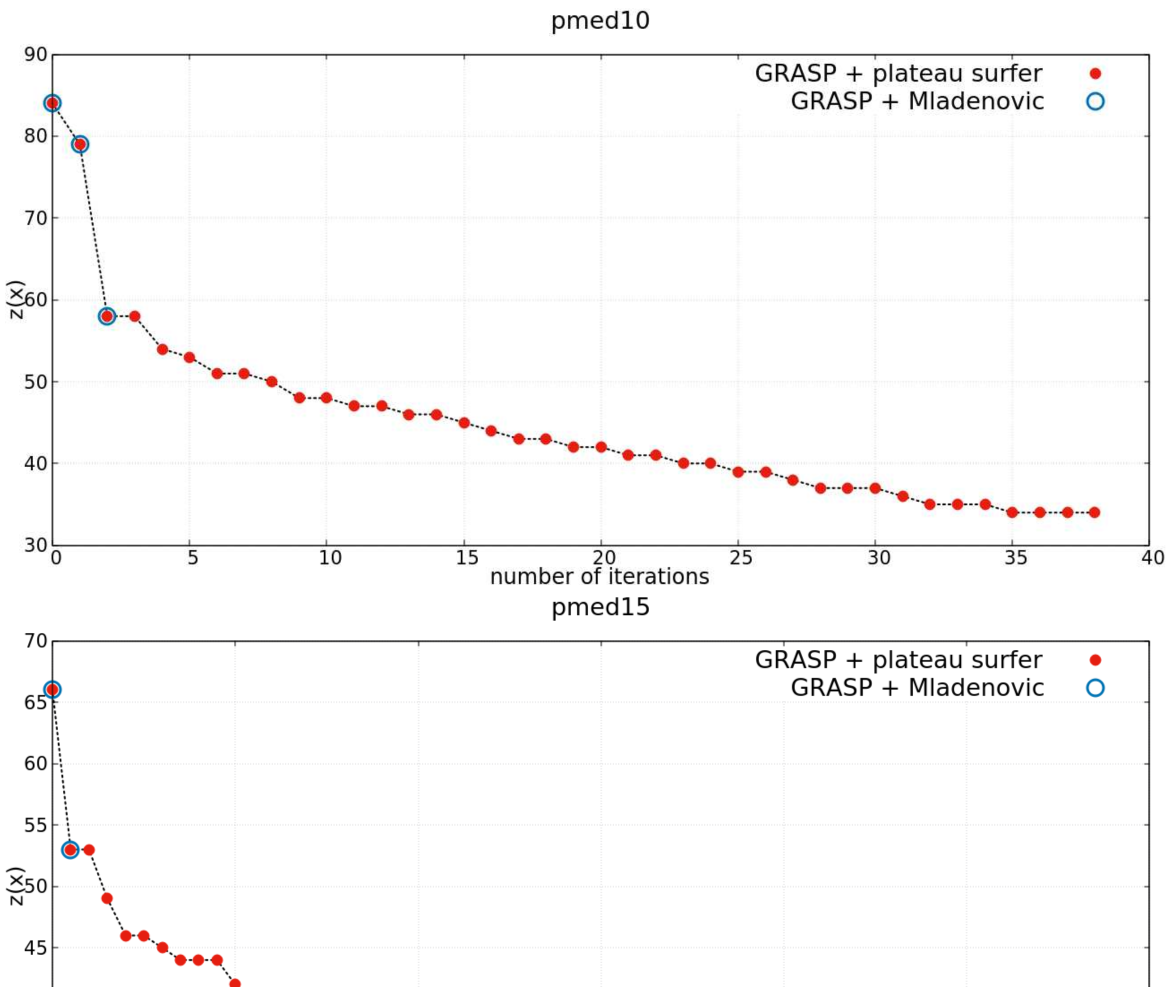


Fig. 4.5 Plateau escaping. The behavior of our *plateau surfer* local search (in red) compared with the Mladenovic's one (in blue). Both algorithms work on the same instances taking as input the same starting solution. Filled red dots and empty blue circles indicate the solutions found by the two algorithms. Mladenovic's local search stops as soon as the first plateau is met.

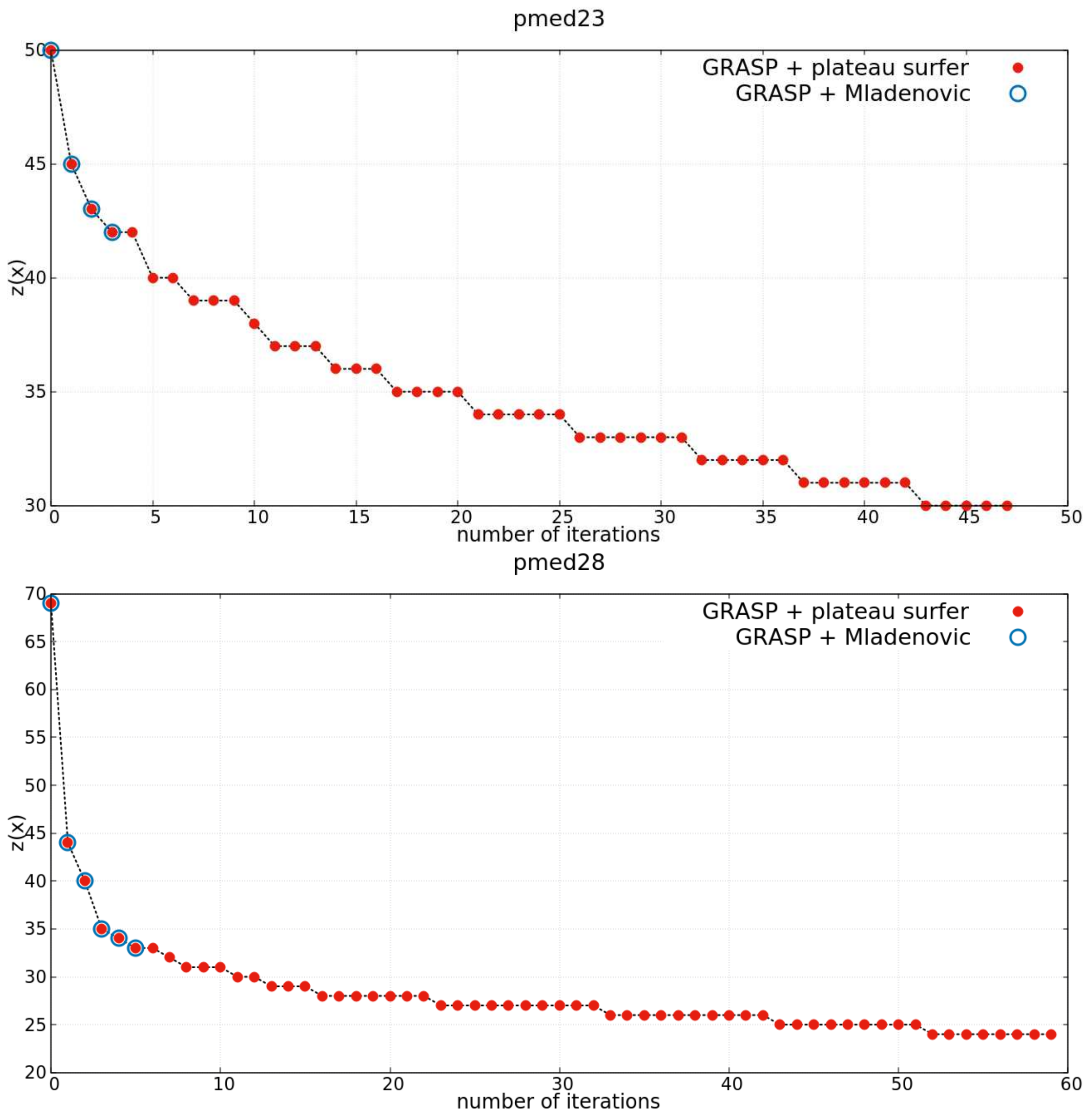


Fig. 4.6 Plateau escaping. The behavior of our *plateau surfer* local search (in red) compared with the Mladenovic's one (in blue) on other two different instances.

```
local-search-GRASP-pCP (G, P, p)
repeat
    modified ← false;
    forall $i \in P$ do
        best_flip ← best_cv_flip ← NULL;
        bestNewSolValue ← $\mathbb{C}(P)$;
        best_cv ← $\max_\delta(\overline{P})$;
        forall $j \in V \setminus P$ do
            $\overline{P} \leftarrow P \setminus \{i\} \cup \{j\}$;
            if $\mathbb{C}(\overline{P}) <$ bestNewSolValue then
                bestNewSolValue ← $\mathbb{C}(\overline{P})$;
                best_flip ← $j$;
            else if best_flip = NULL and $\max_\delta(\overline{P}) <$ best_cv then
                best_cv ← $\max_\delta(\overline{P})$;
                best_cv_flip← $j$;
        if best_flip $\neq$ NULL then
            $P \leftarrow P \setminus \{i\} \cup$ {best_flip};
            modified := true;
        else if best_cv_flip $\neq$ NULL then
            $P \leftarrow P \setminus \{i\} \cup$ {best_cv_flip};
            modified ← true;
until modified ← false;
return P;
```

Fig. 4.7 Pseudocode of the plateau surfer local search algorithm based on the critical node concept.

the pseudocode that goes from line 7 to 14. Starting from an initial solution P, the algorithm tries to improve the solution replacing a node $j \notin P$ with a facility $i \in P$. Clearly, this swap is stored as an improving move if the new solution $\overline{P} = P \setminus \{i\} \cup \{j\}$ is strictly better than P according to the cost function $\mathbb{C}$. If $\mathbb{C}(\overline{P})$ is better than the current cost $\mathbb{C}(P)$, then $\overline{P}$ is compared also with the incumbent solution and if it is the best solution found so far, the incumbent is update and the swap that led to this improvement stored (lines 9-11).

Otherwise, the algorithm checks if it is possible to reduce the number of critical nodes. If the new solution $\overline{P}$ is such that $\overline{P} <_{cv} P$, then the algorithm checks if $\overline{P}$ is the best solution found so far (line 12), the value that counts the number of critical nodes in a solution is update (line 13), and the current swap stored as an improving move (line 14).

To study the computational complexity of our local search, let be $n = |V|$ and $p = |P|$, the number of nodes in the graph and the number of facilities in a solution, respectively. The loops at lines 3 and 7 are executed p and n times, respectively. The update of the solution takes $\mathcal{O}(n)$. In conclusion, the total complexity is $\mathcal{O}(p \cdot n^2)$.

4.2.2 Experimental Results

In this section, we describe computational experience with the local search proposed above. We have compared it with the local search proposed by Mladenovic et al. [93], by embedding both in a GRASP framework. The algorithms were implemented in C++, compiled with `gcc 5.2.1` under Ubuntu with `-std=c++14` flag. The stopping criterion is $maxTime = 0.1 \cdot n + 0.5 \cdot p$. All the tests were run on a cluster of nodes, connected by 10 Gigabit Infiniband technology, each of them with two processors Intel Xeon E5-4610v2@2.30GHz.

Table 4.2 summarizes the results on a set of `ORLIB` instances, originally introduced in [5]. It consists of 40 graphs with number of nodes ranging from 100 to 900, each with a suggested value of p ranging from 5 to 200. Each node is both a user and a potential facility, and distances are given by shortest path lengths. Tables 4.3 and 4.4 report the results on the TSP set of instances. They are just sets of points on the plane. Originally proposed for the traveling salesman problem, they are available from the TSPLIB [108]. Each node can be either a user or an open facility. We used the *Mersenne Twister* random number generator by Matsumoto and Nishimura [90]. Each algorithm was run with 10 different seeds, and minimum (min), average (E) and variance (σ^2) values are listed in each table. The second to last column lists the %-Gap between average solutions. To deeper investigate the statistical significance of the results obtained by the two local searches, we performed the Wilcoxon test [21, 127].

Generally speaking, the Wilcoxon test is a ranking method that well applies in the case of a number of paired comparisons leading to a series of differences, some of which may be positive and some negative. Its basic idea is to substitute scores $1, 2, 3, \ldots, n$ with the actual numerical data, in order to obtain a rapid approximate idea of the significance of the differences in experiments of this kind.

Table 4.2 Results on ORLIB instances.

	GRASP + mladenovic			GRASP + plateau-surfer				
Instance	min	E	σ^2	min	E	σ^2	%-Gap	p-value
pmed01	127	127	0	127	127	0	0.00	
pmed02	98	98	0	98	98	0	0.00	
pmed03	93	93.14	0.12	93	93.54	0.25	0.43	
pmed04	74	76.21	1.33	74	74.02	0.04	-2.87	1.20E-16
pmed05	48	48.46	0.43	48	48	0	-0.95	
pmed06	84	84	0	84	84	0	0.00	
pmed07	64	64.15	0.27	64	64	0	-0.23	
pmed08	57	59.39	1.36	55	55.54	0.73	-6.48	3.37E-18
pmed09	42	46.87	2.83	37	37.01	0.01	-21.04	2.80E-18
pmed10	29	31.21	0.81	20	20.01	0.01	-35.89	9.38E-19
pmed11	59	59	0	59	59	0	0.00	
pmed12	51	51.89	0.1	51	51.41	0.24	-0.93	
pmed13	42	44.47	0.73	36	36.94	0.06	-16.93	1.04E-18
pmed14	35	38.59	3.24	26	26.85	0.13	-30.42	2.11E-18
pmed15	28	30.23	0.7	18	18	0	-40.46	1.09E-18
pmed16	47	47	0	47	47	0	0.00	
pmed17	39	40.71	0.23	39	39	0	-4.20	8.69E-20
pmed18	36	37.95	0.29	29	29.41	0.24	-22.50	6.37E-19
pmed19	27	29.32	0.42	19	19.13	0.11	-34.75	6.25E-19
pmed20	25	27.05	0.99	14	14	0	-48.24	1.46E-18
pmed21	40	40	0	40	40	0	0.00	
pmed22	39	40.06	0.24	38	38.94	0.06	-2.80	1.30E-18
pmed23	30	32.02	0.44	23	23.21	0.17	-27.51	7.16E-19
pmed24	24	25.38	0.34	16	16	0	-36.96	4.37E-19
pmed25	22	22.62	0.24	11	11.89	0.1	-47.44	2.77E-19
pmed26	38	38	0	38	38	0	0.00	
pmed27	33	33.96	0.06	32	32	0	-5.77	2.15E-22
pmed28	26	26.78	0.17	19	19	0	-29.05	2.20E-20
pmed29	23	23.43	0.31	13	13.68	0.22	-41.61	8.00E-19
pmed30	20	21.18	0.47	10	10	0	-52.79	6.50E-19
pmed31	30	30	0	30	30	0	0.00	
pmed32	30	30.37	0.23	29	29.62	0.24	-2.47	
pmed33	23	23.76	0.2	16	16.28	0.2	-31.48	4.31E-19
pmed34	21	22.42	0.66	11	11.56	0.25	-48.44	1.59E-18
pmed35	30	30.01	0.01	30	30	0	-0.03	
pmed36	28	29.37	0.31	27	27.65	0.23	-5.86	4.52E-18
pmed37	23	24.07	0.37	16	16	0	-33.53	2.74E-19
pmed38	29	29	0	29	29	0	0.00	
pmed39	24	25.08	0.11	23	23.98	0.02	-4.39	4.68E-21
pmed40	20	21.81	0.43	14	14	0	-35.81	5.14E-19
						Average	-16.78	

More formally, let A_1 and A_2 be two algorithms, $I_1, \ldots, I_l$ be l instances of the problem to solve, and let $\delta_{A_i}(I_j)$ be the value of the solution obtained by algorithm A_i $(i = 1, 2)$ on instance I_j $(j = 1, \ldots, l)$. For each $j = 1, \ldots, l$, the Wilcoxon test computes the differences $\Delta_j = |\delta_{A_1}(I_j) - \delta_{A_2}(I_j)|$ and sorts them in

Table 4.3 Results on TSPLIB instances (1)

		GRASP + mladenovic			GRASP + plateau-surfer				
Instance	p	min	E	σ^2	min	E	σ^2	%-Gap	p-value
pcb3038	50	534.48	608.49	1068.09	355.68	374.66	51.05	-38.43	3.90E-18
	100	399.49	481.75	1285.58	259.67	270.2	17.56	-43.91	3.90E-18
	150	331.62	428.69	1741.11	206.71	215.78	23.73	-49.67	3.90E-18
	200	301.01	386.56	3161.87	177.79	190.88	10.4	-50.62	3.90E-18
	250	292.48	359.59	3323.62	155.03	163.75	19.24	-54.46	3.90E-18
	300	261.28	349.42	2902.71	143.39	151.89	10.04	-56.53	3.90E-18
	350	258.82	336.08	3755.72	123.85	136.22	22.45	-59.47	3.90E-18
	400	249.78	337.14	4033.46	119.07	122.31	1.2	-63.72	3.90E-18
	450	214.97	321.36	3373.23	115	117	0.6	-63.59	3.90E-18
	500	209.35	299.4	3378.98	102	110.38	5.78	-63.13	3.90E-18
pr1002	10	3056.55	3313.49	10132.77	2616.3	2727.45	2260.95	-17.69	3.90E-18
	20	2404.16	2668.29	8244.65	1806.93	1886.89	1516.07	-29.28	3.90E-18
	30	2124.26	2358.07	4432.11	1456.02	1505.55	910.93	-36.15	3.89E-18
	40	1960.23	2172.63	7831.77	1253.99	1302.76	751.62	-40.04	3.90E-18
	50	1755.7	1992.08	5842.66	1097.72	1156.77	815.35	-41.93	3.90E-18
	60	1697.79	1865.5	5872.47	1001.25	1042.82	257.42	-44.1	3.89E-18
	70	1569.24	1736.41	4078.39	900	954.04	307.65	-45.06	3.89E-18
	80	1486.61	1633.87	3278.4	851.47	889.5	407.29	-45.56	3.88E-18
	90	1350.93	1543.17	3922.25	764.85	809.78	382.29	-47.52	3.89E-18
	100	1312.44	1472.47	2616	743.3	767.62	77.4	-47.87	3.89E-18
pr439	10	2575.12	2931.83	38470.59	1971.83	1972.28	19.61	-32.73	3.79E-18
	20	1940.52	2577.03	23638.88	1185.59	1194.12	124.58	-53.66	3.71E-18
	30	1792.34	2510.91	23692.47	886	919.1	442.37	-63.4	3.89E-18
	40	1525.2	2413.33	53876.4	704.45	728.19	39.31	-69.83	3.88E-18
	50	1358.54	2252.46	89633.71	575	595.4	64.21	-73.57	3.82E-18
	60	1386.09	2170.85	110065.93	515.39	537.66	75.43	-75.23	3.89E-18
	70	1370.45	1898.53	116167.77	480.23	499.65	4.93	-73.68	3.73E-18
	80	1140.18	1815.1	118394.68	424.26	440.27	166.06	-75.74	3.89E-18
	90	1191.9	1699.64	91388.99	400	406.17	31.71	-76.1	3.88E-18
	100	1190.85	1679.73	94076.45	375	384.27	98.91	-77.12	3.89E-18
rat575	10	81.32	92.98	9.27	73	74.71	0.79	-19.65	3.90E-18
	20	68.07	73.86	3.7	50.54	53.04	0.63	-28.19	3.90E-18
	30	59.81	64.61	3.67	41.79	43.53	0.47	-32.63	3.90E-18
	40	54.13	58.37	3.43	36.12	37.43	0.29	-35.87	3.90E-18
	50	47.68	53.78	3.56	32.45	33.36	0.17	-37.97	3.90E-18
	60	45.62	50.03	3.21	29.15	30.17	0.19	-39.7	3.90E-18
	70	43.68	46.96	2.97	27	27.78	0.13	-40.84	3.90E-18
	80	39.81	44.2	2.75	25.02	25.99	0.11	-41.2	3.90E-18
	90	38.48	41.98	2.06	23.85	24.4	0.07	-41.88	3.90E-18
	100	37.01	39.93	1.4	22.2	23.01	0.08	-42.37	3.89E-18
rat783	10	102.22	110.93	13.17	83.49	87.82	1.65	-20.83	3.90E-18
	20	80.53	88.56	7.68	59.68	62.8	1.41	-29.09	3.90E-18
	30	69.58	76.92	7.87	49.25	51.48	0.74	-33.07	3.90E-18
	40	62.97	69.63	4.62	42.05	44.27	0.53	-36.42	3.90E-18
	50	59.41	65.26	5.59	38.29	39.6	0.42	-39.32	3.90E-18
	60	54.82	60.35	4.77	34.48	35.92	0.24	-40.48	3.90E-18
	70	49.4	56.56	7.48	32.06	33.11	0.24	-41.46	3.90E-18
	80	48.51	53.76	4.03	29.55	30.94	0.21	-42.45	3.90E-18
	90	46.07	51.82	3.53	28.18	28.85	0.11	-44.33	3.90E-18
	100	43.97	49.5	4.68	26.31	27.49	0.14	-44.46	3.90E-18
							Average	-46.84	

Table 4.4 Results on TSPLIB instances (2)

		GRASP + mladenovic			GRASP + plateau-surfer				
Instace	p	min	E	σ^2	min	E	σ^2	%-Gap	p-value
rl1323	10	3810.84	4185.89	24655.46	3110.57	3241.79	3290.56	-22.55	3.90E-18
	20	2996.4	3348.31	23183.21	2090.87	2236.28	2798.56	-33.21	3.90E-18
	30	2689.44	2979.79	14205.75	1730.78	1808.94	1544.85	-39.29	3.90E-18
	40	2337.92	2712.93	14193.05	1479.24	1576.25	1710.4	-41.9	3.90E-18
	50	2195.91	2462.95	9835.09	1300	1363.88	950.66	-44.62	3.90E-18
	60	2021.87	2278.94	16400.27	1181.3	1244.03	657.55	-45.41	3.90E-18
	70	1900.77	2128.45	11883.58	1076.2	1127.98	475.13	-47	3.90E-18
	80	1866.8	2033.24	4501.73	988.87	1048.87	438.82	-48.41	3.89E-18
	90	1634.37	1966.13	4643.42	935.02	978.6	289.18	-50.23	3.89E-18
	100	1631.5	1909.56	8483.55	886.85	914	238.2	-52.14	3.89E-18
u1060	10	3110.65	3373.87	7541.61	2301.7	2440	599.42	-27.68	3.86E-18
	20	2652.6	2818.37	5787.51	1650.34	1749.15	2814.03	-37.94	3.90E-18
	30	2501.72	2684.87	3811.23	1302.94	1373.21	912.92	-48.85	3.90E-18
	40	2442.07	2616.15	5267.85	1118.59	1176.14	593.6	-55.04	3.90E-18
	50	2378.36	2591.96	7266.77	950.66	1021.55	418.91	-60.59	3.90E-18
	60	2301.83	2602.13	13579.82	860.49	919.97	374.54	-64.65	3.90E-18
	70	2378.36	2606.64	10944.09	790.13	828.16	441.03	-68.23	3.90E-18
	80	2351.82	2622.32	12980.39	720.94	753.64	306.94	-71.26	3.90E-18
	90	2248.61	2562.01	10260.36	667.55	708.04	107.79	-72.36	3.90E-18
	100	2060.29	2494.08	11025.91	632.11	653.15	110.65	-73.81	3.90E-18
	110	2049.18	2444.22	10385.95	570.49	613.02	148.7	-74.92	3.90E-18
	120	2122.97	2406.19	9191.4	570	579.93	96.23	-75.9	3.90E-18
	130	1839.55	2390.82	12029.95	538.82	561.62	78.78	-76.51	3.90E-18
	140	1924.48	2316.25	12982.87	500.39	527.66	172.51	-77.22	3.90E-18
	150	1942.27	2300.45	13245.06	499.65	503.26	20.49	-78.12	3.90E-18
u1817	10	592.97	646.89	325	466.96	485.44	104.33	-24.96	3.90E-18
	20	462.3	564.44	560.9	330.2	348.15	53.96	-38.32	3.90E-18
	30	418.91	530.34	1018.29	265.19	283.4	58.43	-46.56	3.90E-18
	40	407.19	526.44	956.01	232.25	245.78	43.42	-53.31	3.90E-18
	50	330.21	507.52	2889.76	204.79	217.05	26.96	-57.23	3.90E-18
	60	352.88	497.35	3539.09	184.91	197.26	21.79	-60.34	3.90E-18
	70	321.27	477.43	4139.93	170.39	181.53	13.67	-61.98	3.90E-18
	80	289.61	445.35	4866.81	154.5	166.46	22.68	-62.62	3.90E-18
	90	283.99	422.34	3828.2	148.11	153.5	13.72	-63.65	3.90E-18
	100	283.99	416.69	2660.21	136.79	146.67	7.4	-64.8	3.90E-18
							Average	-54.9	

non decreasing order. Accordingly, starting with a smallest rank equal to 1, to each difference Δ_j, it assigns a non decreasing rank R_j. Ties receive a rank equal to the average of the sorted positions they span. Then, the following quantities are computed

$$W^+ = \sum_{j=1,\ldots,l:\ \Delta_j>0} R_j,$$
$$W^- = \sum_{j=1,\ldots,l:\ \Delta_j<0} R_j.$$

Under the null hypothesis that $\delta_{A_1}(I_j)$ and $\delta_{A_2}(I_j)$ have the same median value, it should result that $W^+ = W^-$. If the p-value associated to the experiment is less than an a priori fixed significance level α, then the null hypothesis is rejected and the difference between W^+ and W^- is considered significant. The last column of each table lists the p-values where the %-Gap is significant, all the values are less than $\alpha = 0.01$. This outcome of the Wilcoxon test further confirms that our local search is better performing than the local search proposed by Mladenovic et al.

4.3 A GRASP for the Minimum Cost Satisfiability Problem (MinCost-SAT)

Propositional Satisfiability (SAT) is a well known problem in logic and optimization belong to the NP-complete problems [58]. This problem plays an important role in the theory of complexity, where it is used to model complex problems. In logic supervised learning, for example, a dataset of samples is given, each of which represented by a finite number of logic variables, and a particular extension of the classic SAT problem - the Minimum Cost Satisfiability Problem (MinCost-SAT) - can be used to iteratively identify the different clauses of a compact formula in Disjunctive Normal Form (DNF) that possesses the desirable property of assuming the value *True* on one specific subset of the dataset and the value *False* on the rest. The MinCost-SAT for learning propositional formula from data is described in [37] and [122]. There are several reasons that motivate the validity of this approach for the supervised learning, in fact it is proved to be very effective in several applications, particularly on those derived from biological and medical data analysis [6, 2, 125, 126, 124, 16].

One of the main drawbacks of the approach described in [37] lies in the difficulty of solving MinCost-SAT exactly or with an appropriate quality level. Such drawback is becoming more and more evident as, in the era of Big Data, the size of the datasets that one is to analyze steadily increases. Feature selection techniques may be used to reduce the space in which the samples are represented (one such method specifically designed for data in logic form is described in [7]). While the literature proposes both exact approaches ([54, 105], [122], and [123]) and heuristics ([119]), still the need for efficient MinCost-SAT solvers remains, and in particular for solvers that may take advantage of the specific structure of those MinCost-SAT representing supervised learning problems.

In [38] we tried to fill this gap offering a GRASP-based meta-heuristic designed to solve the MinCost-SAT problems that arise in supervised learning, although GRASP-based approaches were already proposed in literature but for different satisfiability problems [49, 109]. Furthermore, we developed a new probabilistic stopping criterion that proves to be very effective in limiting the exploration of the solution space - whose explosion is a frequent problem in meta-heuristic approaches. The method has been tested on several instances derived from artificial supervised problems in logic form, and successfully compared with four established solvers in the literature (**Z3** from Microsoft Research [25], **bsolo** [79], **MiniSat+** [36], and **PWBO** [87, 89, 88].

MinCost-SAT is a special case of the well known Boolean Satisfiability Problem. Given a set of n boolean variables $X = \{x_1, \ldots, x_n\}$, a non-negative cost function $c : X \mapsto \mathbb{R}^+$ such that $c(x_i) = c_i \geq 0$, $i = 1, \ldots, n$, and a boolean formula $\varphi(X)$ expressed in CNF, the MinCost-SAT problem consists in finding a truth assignment for the variables in X such that the total cost is minimized while $\varphi(X)$ is satisfied. Accordingly, the mathematical formulation of the problem is given as follows:

$$\text{(MinCost-SAT)} \quad z = \min \sum_{i=1}^{n} c_i x_i$$

subject to:

$$\varphi(X) = 1,$$
$$x_i \in \{0, 1\}, \qquad \forall i = 1, \ldots, n.$$

It is easy to see that a general SAT problem can be reduced to a MinCost-SAT problem whose costs c_i are all equal to 0. Furthermore, the decision version of the MinCost-SAT problem is NP-complete [54]. While the Boolean Satisfiability problem is far famed in the landscape of scientific literature, MinCost-SAT has received less attention.

4.3.1 A GRASP for the MinCost-SAT

In the implementation of our GRASP we treat the MinCost-SAT as particular covering problem with incompatibility constraints. In particular, we consider each literal $(x, \neg x)$ as a separate element, and a clause is covered if at least one literal in the clause is contained in the solution. The algorithm tries to add literals to the solution in order to cover all the clauses and, once the literal x is added to the solution, then the literal $\neg x$ cannot be inserted (and vice versa). Therefore, if the literal x is in solution, the variable x is assigned to true and all clauses covered by x are satisfied. Similarly, if the literal $\neg x$ is in solution, the variable x is assigned to false, and clauses containing $\neg x$ are satisfied.

Construction Phase

During this phase, the algorithm adds a literal at a time, until all clauses are covered or no more literals can be assigned. At each iteration, of this phase, if a clause can be covered only by a single literal x - due to the choices made in previous iterations - then x is selected to cover the clause. Otherwise, if there are not clauses covered by only a single literal, the addition of literals to the solution takes place according to a penalty function $\mathtt{penalty}(\cdot)$, which greedily sorts all the candidates literals, as described below.

Let $cr(x)$ be the number of clauses yet to be covered that contain x, then the following amount is computed:

$$\mathtt{penalty}(x) = \frac{c(x) + cr(\neg x)}{cr(x)}. \tag{4.2}$$

The penalty function evaluates both the benefits and disadvantages that can result from the choice of a literal rather than another. The benefits are proportional to the number of uncovered clauses that the chosen literal could cover, while the disadvantages are related to both the cost of the literal and the number of uncovered clauses that could be covered by $\neg x$. The smaller the penalty function $\mathtt{penalty}(x)$, the more favorable is the literal x. According to the GRASP scheme, the selection of the literal to add is not purely greedy, but a Restricted Candidate List (RCL) is created with the most promising elements, and an element is randomly selected among them. Concerning the tuning of the parameter β, whose task is to adjust the greediness of the construction phase, we performed an extensive analysis over a set of ten different random seeds. Such testing showed how a nearly totally greedy

```
1  construction-GRASP-MinCostSAT ($\mathcal{C}$, $X$, $\beta$)
   /* $\mathcal{C}$ is the set of uncovered clauses      */
   /* $X$ is the set of candidate literals               */
2  $s \leftarrow \emptyset$ ;
3  while $C \neq \emptyset$ do
4      if $c \in \mathcal{C}$ can be covered only by $x \in X$ then
5          $s \leftarrow s \cup \{x\}$;
6          $X \leftarrow X \setminus \{x, \neg x\}$;
7          $\mathcal{C} \leftarrow \mathcal{C} \setminus \{\bar{c} \mid x \in \bar{c}\}$;
8      else
9          compute $penalty(x)\ \forall\ x \in X$;
10         $th \leftarrow \min_{x \in X}\{penalty(x)\} + \beta(\max_{x \in X}\{penalty(x)\} - \min_{x \in X}\{penalty(x)\})$ ;
11         $RCL \leftarrow \{\, x \in X\colon penalty(x) \leq th \,\}$ ;
12         $\hat{x} \leftarrow$ rand(RCL) ;
13         $s \leftarrow s \cup \{\hat{x}\}$;
14         $X \leftarrow X \setminus \{\hat{x}, \neg\hat{x}\}$;
15         $\mathcal{C} \leftarrow \mathcal{C} \setminus \{\bar{c} \mid \hat{x} \in \bar{c}\}$;
16 return $s$
```

Fig. 4.8 GRASP construction phase for the MinCost-SAT.

setup ($\beta = 0.1$) allowed the algorithm to attain better quality solutions in smallest running times.

Let $|\mathcal{C}| = m$ be the number of clauses. Since $|X| = 2n$, in the worst case scenario the loop while (Figure 4.8, line 3) in the `construction-GRASP-MinCostSAT` function pseudo-coded in Figure 4.8 runs m times and in each run the most expensive operation consists in the construction of the RCL. Therefore, the total computational complexity is $\mathcal{O}(m \cdot n)$.

Improvement Phase

In the improvement phase, the algorithm uses a 1-exchange (flip) neighborhood function, where two solutions are neighbors if and only if they differ in at most one component. Therefore, if there exists a better solution $\bar{x}$ that differs only for one literal from the current solution x, the current solution s is set to $\bar{s}$ and the procedure restarts. If such a solution does not exists, the procedure ends and returns the current solution s. The local search procedure would also re-establish feasibility if the current solution is not covering all clauses of $\varphi(X)$. During our experimentation we tested the one-flip local search using two different neighborhood exploration strategies: first improvement and best improvement. With the former strategy, the current solution is replaced by the first improving solution found in its neighborhood; such improving solution is then used as a starting point for the next local exploration. On the other hand, with the best improvement strategy, the current solution x is replaced with the solution $\bar{x} \in \mathcal{N}(x)$ corresponding to the greatest improvement in terms of objective function value; $\bar{x}$ is then used as a starting point for the next local exploration. Our results showed how the first improvement strategy is slightly faster, as expected, while attaining solution of the same quality of those given by the best improvement strategy. Based on this rationale, we selected first improvement as exploration strategy in our testing phase.

Stopping Rule: Probabilistic Stop

Most meta-heuristics present a shortcoming in the effectiveness of their stopping rule. In fact, the stopping criterion is based on a bound on the maximum number of iterations, a limit on total execution time, or a given maximum number of consecutive iterations without improvement. For our GRASP we propose a probabilistic stopping criterion, inspired by [113]. The stopping rule is articulated in two phases and it can be sketched as follows. First, let $\mathcal{X}$ be a random variable representing the value of a solution obtained at the end of a generic GRASP iteration. In the first phase - the `fitting-data` procedure - the probability distribution $f_{\mathcal{X}}(\cdot)$ of $\mathcal{X}$ is estimated, while during the second phase - `improve-probability` procedure - the probability of obtaining an improvement of the current solution value is computed. Then, accordingly to a threshold, the algorithm either stops or continues its execution.

Specifically, the `fitting-data` procedure tries to represent the random variable $\mathcal{X}$ with a theoretical distribution. In fact, examining, at the end of each iteration, the trend of the objective function values and summing up their frequencies, using for example an histogram diagram, it is possible to establish a promising family of probability distributions. After that, by means of a Maximum Likelihood Estimation (`MLE`), as described in [117], a choice is made by regarding the parameters characterizing the best fitting distribution of the chosen family.

In order to conduct an empirical analysis of the objective function value we represents the data in the following way: let $\mathcal{I}$ be an instance and $\mathcal{F}$ the set of solutions obtained by the algorithm up to the current iteration, and let $\mathcal{Z}$ be the multiset of the objective function values associated to $\mathcal{F}$. Since we are facing with a minimum optimization problem during the analysis of the values in $\mathcal{Z}$ we expect to find an higher concentration of elements between the mean value μ and the $\max(\mathcal{Z})$.

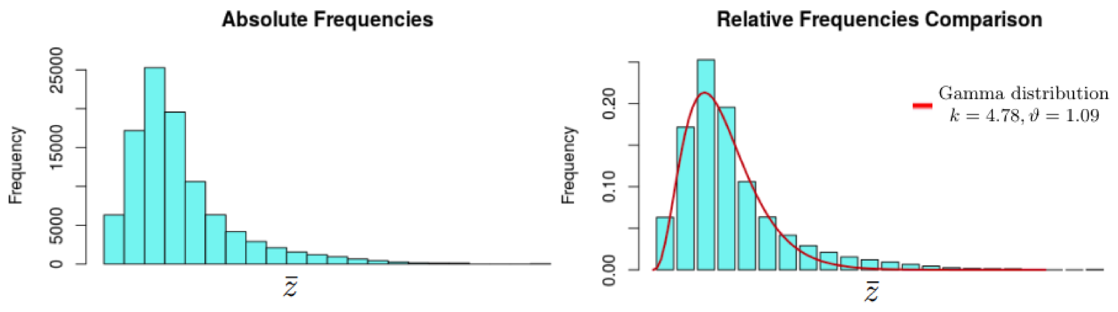


Fig. 4.9 Empirical analysis of frequencies of the solutions.

```
fitting-data ($\bar{\mathcal{Z}}$)
/* $\bar{\mathcal{Z}}$ is the initial sample of the objective function values     */
foreach $z \in \bar{\mathcal{Z}}$ do
    $z = \max(\bar{\mathcal{Z}}) - z$;
$\{k, \theta\} \leftarrow$ MLE($\mathcal{Z}$,"gamma");
return $\{k, \theta\}$
```

Fig. 4.10 Fitting data procedure.

To represent the values in $\mathcal{Z}$ with a positive distribution function, that presents higher frequencies in a right neighborhood of zero and a single tail which decays for growing values of the random variable, we perform a reflection on the data in $\mathcal{Z}$ by means the following transformation:

$$\bar{z} = \max(\mathcal{Z}) - z, \ \forall\, z \in \mathcal{Z}. \tag{4.3}$$

The ongoings of the distribution of $\bar{z}$ in our instances has then a very recognizable behavior. A representation is depicted in Figure 4.9, where the histogram of absolute and relative frequencies of $\bar{z}$ are plotted. It is easy to observe how the gamma distribution family represents a suitable guess for our random variable. In the next step, we estimate the parameters of the selected distribution, it this case the gamma distribution, performing a MLE. In order to estimate the parameters, at the beginning we collect an initial sample of objective function values and on-line execute a function, developed in R reported in , which carries out the MLE and returns the characteristics $shape$ and $scale$, k and θ, which pinpoint the specific distribution of the gamma family that best suits the data.

```
improve-probability (k, θ, z*)
    /* z* is the value of the incumbent */
p ← pgamma($z^*$, $shape = k$, $scale = \theta$);
return p
```

Fig. 4.11 Improve probability procedure.

The second phase of the probabilistic stop takes place once that the probability distribution function of the random variable $\mathcal{X}$, $f_{\mathcal{X}}(\cdot)$ has been estimated. Let $\hat{z}$ be the best solution value found so far, then, it is possible to compute an approximation of the probability of improving the incumbent solution by:

$$p = 1 - \int_0^{\max(\mathcal{Z})-\hat{z}} f_{\mathcal{X}}(t)\ dt. \tag{4.4}$$

The result of `fitting-data` and `improve-probability` consists in an estimate of the probability of incurring in an improving solution in the next iterations. Such probability is compared with a user-defined threshold, α, and if $p < \alpha$ the algorithm stops. More specifically, in our implementation the stopping criterion works as follows:

a) let q be an user-defined ìpositive integer, and let $\bar{\mathcal{Z}}$ be the sample of initial solution values obtained by the GRASP in the first q iterations;

b) call the `fitting-data` procedure, whose input is $\bar{\mathcal{Z}}$ is called one-off to estimate shape and scale parameters, k and θ, of the best fitting gamma distribution;

c) every time that an incumbent is improved, `improve-probability` procedure (pseudo-code in Figure 4.11) is performed and the probability p of further improvements is computed. If p is less than or equal to α the stopping criterion is satisfied. For the purpose of determining p, we have used the function `pgamma` of `R` package `stats`.

4.3.2 Experimental Results

Our GRASP has been implemented in C++ and compiled with `gcc 5.4.0` with the flag `-std=c++14`. All tests were run on a cluster of nodes, connected by 10 Gigabit Infiniband technology, each of them with two processors Intel Xeon E5-4610v2@2.30GHz.

We performed two different kinds of experimental tests. In the first one, we compared the algorithm with different solvers proposed in literature, without use of probabilistic stop. In particular, we used: **Z3** solver freely available from Microsoft Research [25], **bsolo** solver kindly provided by its authors [79], the **MiniSat+** [36] available at web page http://minisat.se/, and **PWBO** available at web page http://sat.inesc-id.pt/pwbo/index.html. The aim of this first set of computational experiment is the evaluation of the quality of the solutions obtained by our algorithm within a certain time limit. More specifically, the stopping criterion for GRASP, **bsolo**, and **PWBO** is a time limit of 3 hours, for **Z3** and **MiniSat+** is the reaching of an optimal solution.

Z3 is a satisfiability modulo theories (SMT) solver from Microsoft Research that generalizes boolean satisfiability by adding equality reasoning, arithmetic, fixed-size bit-vectors, arrays, quantifiers, and other useful first-order theories. **Z3** integrates modern backtracking-based search algorithm for solving the CNF-SAT problem, namely DPLL-algorithm; in addition it provides a standard search pruning methods, such as two-watching literals, lemma learning using conflict clauses, phase caching for guiding case splits, and performs non-chronological backtracking.

bsolo [79, 80] is an algorithmic scheme resulting from the integration of several features from SAT-algorithms in a branch-and-bound procedure to solve the binate covering problem. It incorporates the most important characteristics of a branch-and-bound and SAT algorithm, bounding and reduction techniques for the former, and search pruning techniques for the latter. In particular, it incorporates the search pruning techniques of the Generic seaRch Algorithm-SAT proposed in [81].

MiniSat+ [36, 120] is a minimalistic implementation of a Chaff-like SAT solver based on the two-literal watch scheme for fast boolean constraint propagation [95], and conflict clauses driven learning [81]. In fact the **MiniSat** solver provides a mechanism which allows to minimize the clauses conflicts.

PWBO [87, 89, 88] is a Parallel Weighted Boolean Optimization Solver. The algorithm uses two threads in order to simultaneously estimate a lower and an upper bound, by means of an unsatisfiability-based procedure and a linear search, respectively. Moreover, learned clauses are shared between threads during the search.

For testing, we have initially considered the datasets used to test feature selection methods in [7], where an extensive description of the generation procedure can be found. Such testbed is composed of 4 types of problems (A,B,C,D), for each of which 10 random repetitions have been generated. Problems of type A and B are of moderate size (100 positive examples, 100 negative examples, 100 logic features), but differ in the form of the formula used to classify the samples into the positive and negative classes (the formula being more complex for B than for A). Problems of type C and D are much larger (200 positive examples, 200 negative examples, 2500 logic features), and D has a more complex generating logic formula than C.

Table 4.5 reports both the value of the solutions and the time needed to achieve them (in the case of GRASP, it is average over ten runs).[1] For problems of moderate size (A and B), the results show that GRASP finds an optimal solution whenever one of the exact solvers converges. Moreover, GRASP is very fast in finding the optimal solution, although here it runs the full allotted time before stopping the search. For larger instances (C and D), GRASP always provides a solution within the bounds, while two of the other tested solvers fail in doing so and the two that are successful (**bsolo**, **PWBO**) always obtain values of inferior quality.

The second set of experimental tests was performed for the purpose of evaluating the impact of the probabilistic stopping rule. In order to do so, we have chosen five different values for threshold α, two distinct sizes for the set $\bar{\mathcal{Z}}$ of initial solution, and executed GRASP using ten different random seeds imposing a maximum number of iterations as stopping criterion. This experimental setup yielded for each instance, and for each threshold value, 20 executions of the algorithm. About such runs, the data collected were: the number of executions in which the probabilistic stopping rule was verified ("stops"), the mean value of the objective function of the best solution found (μ_z), and the average computational time needed (μ_t). To carry out the evaluation of the stopping rule, we executed the algorithm only using the maximum number of iterations as stopping criterion for each instance and for each random seed. About this second setup, the data collected are, as for the first one, the objective function of the best solution found ($\mu_{\hat{z}}$) and the average computational time needed ($\mu_{\hat{t}}$). For the sake of comparison, we considered the percentage gaps between the results collected with and without the probabilistic stopping rule. The second set of experimental tests is summarized in Table 4.6 and in Figure 4.13. For each pair of columns (3,4), (6,7), (9,10), (12, 13), the table reports the percentage of loss in terms of objective function value and the percentage of gain in terms of computation times using the probabilistic stopping criterion, respectively. The analysis of the gaps shows how the probabilistic stop yields little or no changes in the objective function value while bringing dramatic improvements in the total computational time.

[1]For missing values, the algorithm was not able to find the optimal solution in 24 hours.

Table 4.5 Comparison between GRASP and other solvers.

	GRASP		Z3		bsolo		MiniSat+		pwbo-2T	
Inst.	Time	Value	Time	Value	Time	Value	Time	Value	Time	Value
A1	6.56	78.0	10767.75	78.0	0.09	78.0	0.19	78.0	0.03	78.0
A2	1.71	71.0	611.29	71.0	109.59	71.0	75.46	71.0	121.58	71.0
A3	0.64	65.0	49.75	65.0	598.71	65.0	10.22	65.0	5.14	65.0
A4	0.18	58.0	4.00	58.0	205.77	58.0	137.82	58.0	56.64	58.0
A5	0.29	66.0	69.31	66.0	331.51	66.0	9.03	66.0	30.64	66.0
A6	21.97	77.0	5500.17	77.0	328.93	77.0	32.82	77.0	359.97	77.0
A7	0.21	63.0	30.57	63.0	134.20	63.0	19.34	63.0	24.12	63.0
A8	0.25	62.0	6.57	62.0	307.69	62.0	16.84	62.0	11.81	62.0
A9	12.79	72.0	1088.83	72.0	3118.32	72.0	288.76	72.0	208.63	72.0
A10	0.33	66.0	42.23	66.0	62.03	66.0	37.75	66.0	1.81	66.0
B1	6.17	78.0	8600.60	78.0	304.36	78.0	121.25	78.0	20.01	78.0
B2	493.56	80.0	18789.20	80.0	4107.41	80.0	48.21	80.0	823.66	80.0
B3	205.37	77.0	7037.00	77.0	515.25	77.0	132.74	77.0	1.69	77.0
B4	38.26	77.0	7762.03	77.0	376.00	77.0	119.49	77.0	1462.18	77.0
B5	19.89	79.0	15785.35	79.0	3025.26	79.0	214.52	79.0	45.05	79.0
B6	28.45	76.0	4087.14	76.0	394.45	76.0	162.31	76.0	83.72	76.0
B7	129.76	78.0	10114.84	78.0	490.30	78.0	266.25	78.0	455.92	81.0*
B8	44.42	76.0	5186.45	76.0	5821.19	76.0	1319.21	76.0	259.07	76.0
B9	152.77	80.0	14802.00	80.0	5216.95	82.0	36.28	80.0	557.02	80.0
B10	7.55	73.0	1632.87	73.0	760.28	79.0	370.30	73.0	72.09	73.0
C1	366.24	132.0	86400	–	8616.25	178.0*	86400	–	343.38	178.0*
C2	543.11	131.0	86400	–	323.90	150.0*	86400	–	1742.68	174.0*
C3	5883.6	174.1	86400	–	6166.06	177.0*	86400	–	421.64	177.0*
C4	4507.63	176.3	86400	–	6209.69	178.0*	86400	–	2443.20	177.0*
C5	5707.51	171.2	86400	–	314.18	179.0*	86400	–	67.73	178.0*
C6	6269.91	172.1	86400	–	1547.90	177.0*	86400	–	2188.82	177.0*
C7	6193.15	165.9	86400	–	794.90	177.0*	86400	–	730.36	178.0*
C8	596.58	137.0	86400	–	306.27	169.0*	86400	–	837.71	178.0*
C9	466.3	136.0	86400	–	433.32	179.0*	86400	–	3455.92	178.0*
C10	938.54	136.0	86400	–	3703.94	180.0*	86400	–	4617.24	179.0*
D1	3801.61	145.3	86400	–	307.25	175.0*	86400	–	127.69	180.0*
D2	2040.64	139.0	86400	–	7704.92	177.0*	86400	–	2327.23	177.0*
D3	1742.78	143.0	86400	–	309.10	145.0*	86400	–	345.97	178.0*
D4	1741.95	135.0	86400	–	6457.79	177.0*	86400	–	295.76	178.0*
D5	1506.22	134.0	86400	–	6283.27	178.0*	86400	–	238.81	173.0*
D6	1960.87	144.5	86400	–	309.11	173.0*	86400	–	2413.42	178.0*
D7	1544.42	143.0	86400	–	4378.73	179.0*	86400	–	1250.07	178.0*
D8	1756.15	144.0	86400	–	1214.97	179.0*	86400	–	248.85	179.0*
D9	2779.38	137.0	86400	–	303.11	146.0*	86400	–	4.73	179.0*
D10	5896.86	149.0	86400	–	319.45	170.0*	86400	–	1239.93	176.0*
Y	16.05	0.0	0.73	0.0	9411.06	974*	1.96	0	0.23	0.0

*sub-optimal solution
– no optimal solution found in 24 hours

The experimental evaluation of the probabilistic stop is summarized in the three distinct boxplots of Figure 4.12. Each boxplot reports a sensible information related to the impact of the probabilistic stop, namely: the number of times the probabilistic criterion has been satisfied, the gaps in the objective function values, and the gaps in the computation times obtained comparing the solutions obtained with and without the use of the probabilistic stopping rule. Such information are collected, for each instance, as averages of the data obtained over 20 trials in the experimental setup

Table 4.6 Probabilistic stop on instances A, B, C and D.

threshold α	inst	%-gap z	%-gap t(s)	inst	%-gap z	%-gap t(s)	inst	%-gap z	%-gap t(s)	inst	%gap z	%gap t(s)
$5 \cdot 10^{-2}$	**A1**	-0.0	83.1	**B1**	-2.1	87.1	**C1**	-6.6	76.0	**D1**	-5.0	79.3
$1 \cdot 10^{-2}$	**A1**	-0.0	83.1	**B1**	-2.1	87.1	**C1**	-6.6	76.1	**D1**	-5.0	79.3
$5 \cdot 10^{-3}$	**A1**	-0.0	83.0	**B1**	-2.1	87.1	**C1**	-5.0	74.8	**D1**	-4.9	78.7
$1 \cdot 10^{-3}$	**A1**	-0.0	2.5	**B1**	-2.1	87.1	**C1**	-3.8	70.7	**D1**	-1.7	58.9
$5 \cdot 10^{-4}$	**A1**	-0.0	-15.3	**B1**	-2.1	87.2	**C1**	-2.6	70.2	**D1**	-1.2	49.0
$1 \cdot 10^{-4}$	**A1**	-0.0	-11.8	**B1**	-0.5	86.1	**C1**	-1.3	52.5	**D1**	-0.2	31.6
$5 \cdot 10^{-2}$	**A2**	-0.0	84.0	**B2**	-0.7	87.0	**C2**	-3.5	76.0	**D2**	-0.1	79.1
$1 \cdot 10^{-2}$	**A2**	-0.0	84.1	**B2**	-0.7	87.0	**C2**	-3.5	76.2	**D2**	-0.1	79.1
$5 \cdot 10^{-3}$	**A2**	-0.0	83.6	**B2**	-0.7	86.9	**C2**	-3.5	76.7	**D2**	-0.1	79.1
$1 \cdot 10^{-3}$	**A2**	-0.0	84.0	**B2**	-0.7	87.0	**C2**	-1.9	76.4	**D2**	-0.1	79.1
$5 \cdot 10^{-4}$	**A2**	-0.0	84.9	**B2**	-0.7	87.0	**C2**	-1.9	76.1	**D2**	-0.1	75.7
$1 \cdot 10^{-4}$	**A2**	-0.0	57.9	**B2**	-0.1	71.3	**C2**	-1.9	65.2	**D2**	-0.1	53.5
$5 \cdot 10^{-2}$	**A3**	-0.0	83.4	**B3**	-2.7	87.0	**C3**	-2.7	76.3	**D3**	-1.8	75.2
$1 \cdot 10^{-2}$	**A3**	-0.0	83.8	**B3**	-2.7	87.0	**C3**	-2.1	73.0	**D3**	-1.8	75.2
$5 \cdot 10^{-3}$	**A3**	-0.0	82.9	**B3**	-2.7	87.0	**C3**	-1.7	68.0	**D3**	-1.7	74.8
$1 \cdot 10^{-3}$	**A3**	-0.0	8.3	**B3**	-2.6	86.6	**C3**	-0.6	40.9	**D3**	-0.8	38.5
$5 \cdot 10^{-4}$	**A3**	-0.0	-1.6	**B3**	-2.0	84.1	**C3**	-0.0	28.3	**D3**	-0.5	19.1
$1 \cdot 10^{-4}$	**A3**	-0.0	-6.8	**B3**	-0.7	58.4	**C3**	-0.0	9.9	**D3**	-0.3	14.5
$5 \cdot 10^{-2}$	**A4**	-0.0	86.4	**B4**	-2.3	86.9	**C4**	-4.3	78.8	**D4**	-2.2	75.0
$1 \cdot 10^{-2}$	**A4**	-0.0	6.4	**B4**	-2.3	86.9	**C4**	-3.3	68.0	**D4**	-2.2	70.9
$5 \cdot 10^{-3}$	**A4**	-0.0	3.5	**B4**	-2.3	86.9	**C4**	-2.2	63.9	**D4**	-2.2	66.8
$1 \cdot 10^{-3}$	**A4**	-0.0	1.4	**B4**	-2.3	87.0	**C4**	-1.0	51.2	**D4**	-2.0	41.0
$5 \cdot 10^{-4}$	**A4**	-0.0	5.6	**B4**	-2.3	86.9	**C4**	-0.8	48.6	**D4**	-1.2	29.1
$1 \cdot 10^{-4}$	**A4**	-0.0	6.4	**B4**	-0.6	74.8	**C4**	-0.3	38.1	**D4**	-1.2	18.9
$5 \cdot 10^{-2}$	**A5**	-0.0	87.6	**B5**	-0.7	86.6	**C5**	-2.6	79.7	**D5**	-5.6	75.2
$1 \cdot 10^{-2}$	**A5**	-0.0	12.2	**B5**	-0.7	86.6	**C5**	-1.5	71.5	**D5**	-4.9	75.1
$5 \cdot 10^{-3}$	**A5**	-0.0	12.5	**B5**	-0.7	86.6	**C5**	-0.4	68.1	**D5**	-4.9	75.2
$1 \cdot 10^{-3}$	**A5**	-0.0	12.4	**B5**	-0.7	86.6	**C5**	-0.2	53.2	**D5**	-4.7	67.6
$5 \cdot 10^{-4}$	**A5**	-0.0	12.3	**B5**	-0.6	86.3	**C5**	-0.0	46.8	**D5**	-3.8	60.0
$1 \cdot 10^{-4}$	**A5**	-0.0	12.5	**B5**	-0.1	19.0	**C5**	-0.0	33.2	**D5**	-3.3	49.8
$5 \cdot 10^{-2}$	**A6**	-0.9	87.2	**B6**	-0.8	86.6	**C6**	-3.3	79.9	**D6**	-7.9	76.0
$1 \cdot 10^{-2}$	**A6**	-0.9	87.2	**B6**	-0.8	86.6	**C6**	-2.0	70.5	**D6**	-5.9	74.8
$5 \cdot 10^{-3}$	**A6**	-0.9	87.2	**B6**	-0.8	86.6	**C6**	-1.3	65.4	**D6**	-5.0	74.0
$1 \cdot 10^{-3}$	**A6**	-0.8	87.1	**B6**	-0.7	86.3	**C6**	-0.2	49.6	**D6**	-2.5	71.1
$5 \cdot 10^{-4}$	**A6**	-0.5	86.8	**B6**	-0.1	72.1	**C6**	-0.2	39.9	**D6**	-2.5	71.2
$1 \cdot 10^{-4}$	**A6**	-0.0	66.1	**B6**	-0.0	7.6	**C6**	-0.0	36.6	**D6**	-2.5	67.3
$5 \cdot 10^{-2}$	**A7**	-0.0	87.5	**B7**	-3.1	86.2	**C7**	-3.8	74.4	**D7**	-6.5	75.5
$1 \cdot 10^{-2}$	**A7**	-0.0	11.7	**B7**	-3.1	86.2	**C7**	-2.4	65.7	**D7**	-5.3	72.1
$5 \cdot 10^{-3}$	**A7**	-0.0	11.7	**B7**	-3.1	86.2	**C7**	-1.9	60.7	**D7**	-4.0	68.0
$1 \cdot 10^{-3}$	**A7**	-0.0	11.3	**B7**	-3.1	86.2	**C7**	-0.8	43.0	**D7**	-2.8	61.2
$5 \cdot 10^{-4}$	**A7**	-0.0	11.5	**B7**	-3.0	86.0	**C7**	-0.0	36.4	**D7**	-2.2	60.6
$1 \cdot 10^{-4}$	**A7**	-0.0	11.4	**B7**	-0.8	75.8	**C7**	-0.0	14.0	**D7**	-2.2	57.4
$5 \cdot 10^{-2}$	**A8**	-0.0	88.1	**B8**	-1.5	86.7	**C8**	-3.6	73.9	**D8**	-11.5	76.2
$1 \cdot 10^{-2}$	**A8**	-0.0	88.1	**B8**	-1.5	86.7	**C8**	-3.3	74.7	**D8**	-6.7	73.4
$5 \cdot 10^{-3}$	**A8**	-0.0	88.1	**B8**	-1.5	86.7	**C8**	-3.3	74.4	**D8**	-6.7	73.4
$1 \cdot 10^{-3}$	**A8**	-0.0	16.4	**B8**	-1.2	86.4	**C8**	-3.3	73.7	**D8**	-4.4	68.2
$5 \cdot 10^{-4}$	**A8**	-0.0	16.6	**B8**	-0.8	74.5	**C8**	-3.2	65.6	**D8**	-3.4	67.9
$1 \cdot 10^{-4}$	**A8**	-0.0	16.5	**B8**	-0.0	7.8	**C8**	-2.2	60.5	**D8**	-2.4	64.9
$5 \cdot 10^{-2}$	**A9**	-0.0	88.0	**B9**	-1.9	85.9	**C9**	-4.1	75.3	**D9**	-2.1	75.2
$1 \cdot 10^{-2}$	**A9**	-0.0	88.0	**B9**	-1.9	85.9	**C9**	-2.7	74.8	**D9**	-2.1	75.2
$5 \cdot 10^{-3}$	**A9**	-0.0	88.0	**B9**	-1.9	85.9	**C9**	-1.1	74.4	**D9**	-2.1	75.2
$1 \cdot 10^{-3}$	**A9**	-0.0	16.0	**B9**	-1.9	85.9	**C9**	-1.1	66.6	**D9**	-2.1	75.2
$5 \cdot 10^{-4}$	**A9**	-0.0	16.0	**B9**	-1.7	84.9	**C9**	-0.2	56.5	**D9**	-2.1	67.7
$1 \cdot 10^{-4}$	**A9**	-0.0	15.9	**B9**	-0.5	45.2	**C9**	-0.2	55.7	**D9**	-1.9	60.4
$5 \cdot 10^{-2}$	**A10**	-0.0	83.3	**B10**	-0.3	87.7	**C10**	-0.4	76.3	**D10**	-7.1	73.7
$1 \cdot 10^{-2}$	**A10**	-0.0	75.4	**B10**	-0.3	87.6	**C10**	-0.4	76.2	**D10**	-6.9	73.8
$5 \cdot 10^{-3}$	**A10**	-0.0	0.5	**B10**	-0.3	87.7	**C10**	-0.3	67.9	**D10**	-6.4	73.1
$1 \cdot 10^{-3}$	**A10**	-0.0	-5.4	**B10**	-0.3	87.6	**C10**	-0.3	48.0	**D10**	-4.5	62.0
$5 \cdot 10^{-4}$	**A10**	-0.0	-4.8	**B10**	-0.0	87.4	**C10**	-0.3	48.0	**D10**	-4.3	57.3
$1 \cdot 10^{-4}$	**A10**	-0.0	-4.7	**B10**	-0.0	35.7	**C10**	-0.2	27.0	**D10**	-3.1	38.6

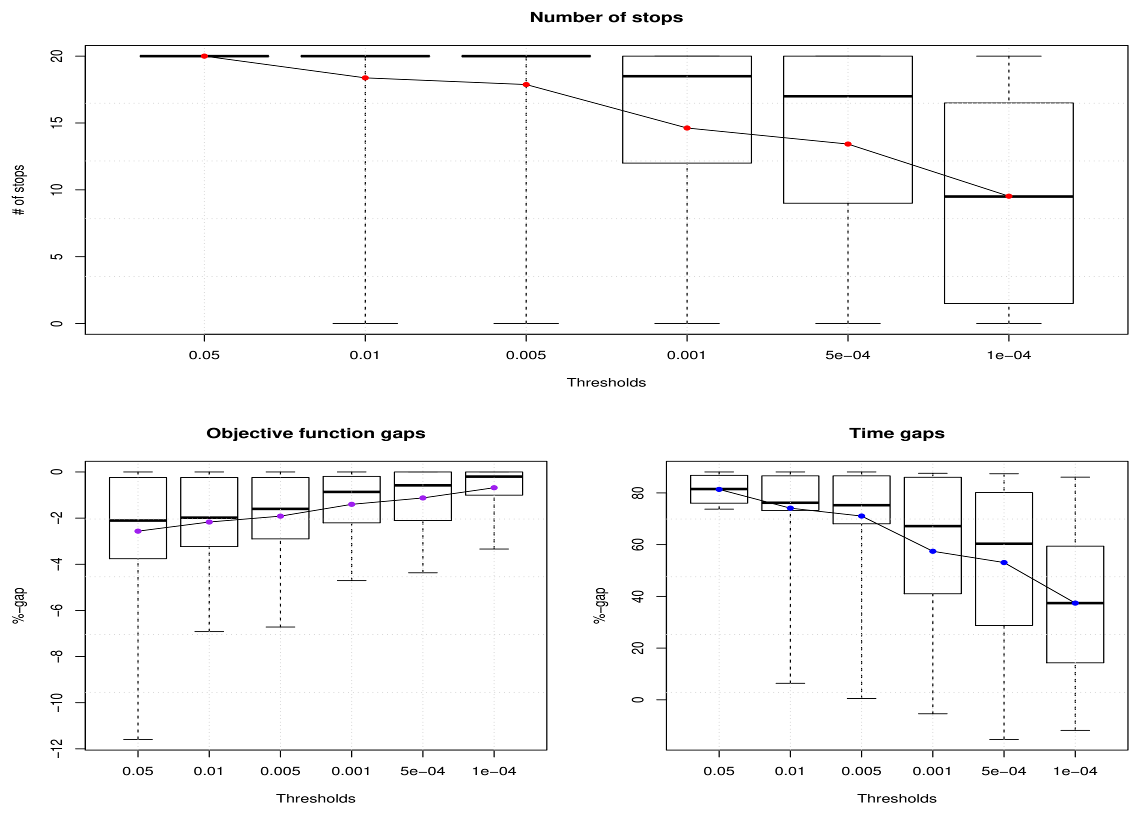


Fig. 4.12 Experimental evaluation of the probabilistic stopping rule. In each boxplot, the boxes represent the first and the second quartile; solid line represent median while dotted vertical line is the full variation range. Plots vary for each threshold α. The dots connected by a line represent the mean values.

described above. The first boxplot depicts the number of total stops recorded for different values of threshold α. Larger values of α, indeed, yield a less coercive stopping rule, thus recording an higher number of stops. Anyhow, even for the smallest, most conservative α, the average number of stops recorded is close to 50% of the tests performed. In the second boxplot, the objective function gap is reported. Such gap quantifies the qualitative worsening in quality of the solutions obtained with the probabilistic stopping rule. The gaps yielded show how even with the highest α, the difference in solution quality is extremely small, with a single minimum of -11.5% for the instance D8, and a very promising average gap, slightly below -2%. As expected, decreasing the α values the solutions obtained with and without the probabilistic stopping rule will align with each other, and the negative gaps will accordingly grow up to approximately -1%. The third boxplot shows the gaps obtained in the computation times. The analysis of such gaps is the key to realistically appraise the actual benefit provided by the use of the probabilistic stopping rule. Observing the results reported, it is possible to note how even in the case of the smallest threshold, i.e., using the most strict probabilistic stopping criterion, the stops recorded are such that an average time discount close to the 40% is encountered. A more direct display of this time gaps can be obtained straightly considering the total time discount in seconds: with the smallest α we have experienced a time discount of 4847.6 seconds over the 11595.9 total seconds needed for the execution without the probabilistic stop. Analyzing in the same fashion the values obtained under the largest threshold, we observed an excellent average discount just over 80%, which quantified in seconds amounts to an astonishing total discount of 8919.64 seconds over the 11595.9 total seconds registered for the execution without the probabilistic stop.

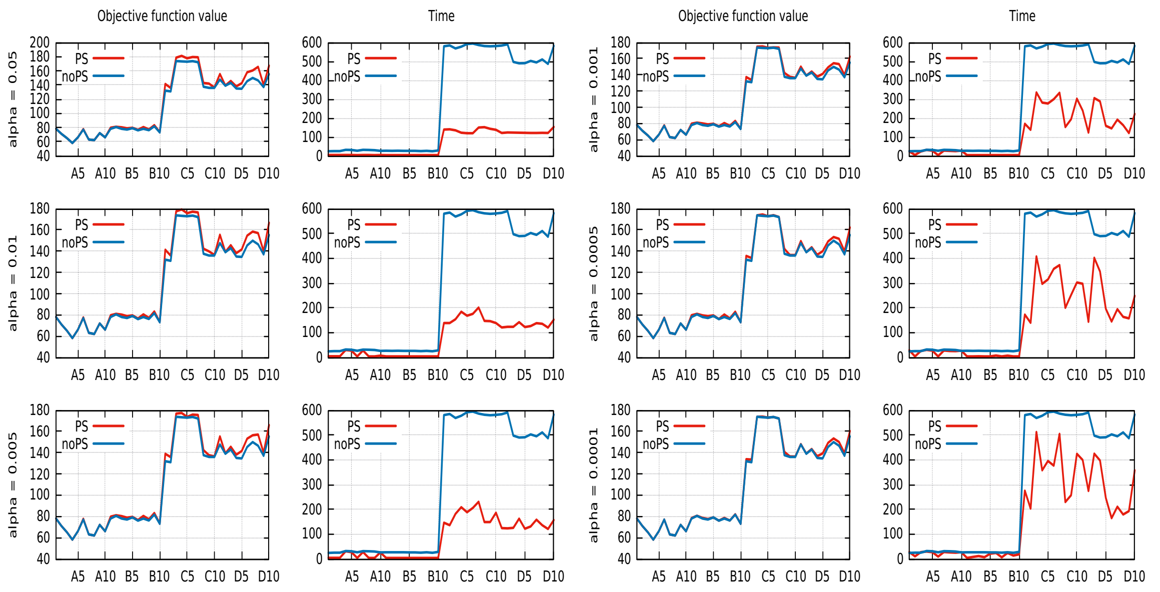


Fig. 4.13 Comparison of objective function values and computation times obtained with and without probabilistic stopping rule for different threshold values.

Chapter 5

Conclusions

5.1 Re-optimization Shortest Path Tree

In Chapter 2, the re-optimization shortest path tree problem, in the case of source node change, has been addressed. To handle the problem of interest, a dual approach, based on a strongly polynomial auction algorithm, has been defined. The theoretical complexity of the proposed approach has been investigated and an extensive computational study has been carried out to assess the performance of the proposed solution strategies. A comparison with a Dijkstra-like approach and a from scratch procedure has been also conducted on different types of networks and real instances. The computational results have shown that the proposed approach always outperforms the others on grid and random networks, and performs better than Dijkstra-like approach on real instances when an ad-hoc priority queue is used.

5.2 Constrained Incremental Graph Drawing

In Chapter 3, we presented a new dynamic version of the classical graph drawing problem, inspired by the incremental graph drawing, with the aim of preserving the mental map. In fact, in the novel formulation, the displacement of the original nodes, taking into account their original position, is limited by an upper and lower bound. This was made possible thanks to the introduction of an additional constraint, the *position constraint*. Furthermore, several resolutive strategies were designed ad-hoc for the problem, based on the GRASP and Tabu Search paradigms. In particular, three different constructive phases for the GRASP, and another one which adopts a

Tabu matrix in order to diversify the range of the initial solutions were presented. To support the constructive phases, were elaborated two improvement strategies, a local search for the GRASP and a Tabu Search for the Tabu-like constructive. Finally, a post-optimization based on Path Relinking was considered to accurately refine the solutions after the improvement phase.

All the designed algorithms obtain extremely accurate solutions, very close to the optimum, in very low computational times.

5.3 Maximum Cut-Clique

In Section 4.1, an hybrid meta-heuristic based on a GRASP and a Phased Local Search for MCCP was proposed. The method has been tested and compared with the most recent heuristic approaches, R-ILS and D-ILS [86]. The preliminary computational results suggest that, compared with the other competitor algorithms, our proposal produces good-quality solutions for all instances demonstrating that it is a well-suited approach for Max Cut-Clique.

5.4 p-Center

In Section 4.2, a new local search heuristic for the p-center problem is presented, whose potential applications appear in telecommunications, in transportation logistics, and whenever one must design a system to organize some sort of public facilities, such as, for example, schools or emergency services. The testing phase underlines that the *plateau surfer* is able to reduce the number of local optimal solutions using the concept of *critical node*, and it outperforms the results of the best known local search for the problem.

Future lines of investigation will be focused on a deeper analysis of the robustness of our proposal by applying it on further instances coming from financial markets and manufacturing systems.

5.5 Minimum Cost Satisfiability

In Section 4.3, we have investigated a strategy for a GRASP heuristic that solves large sized MinCost-SAT. The method adopts a straight-forward implementation of the main ingredients of the heuristic, but proposes a new probabilistic stopping rule. Experimental results show that, for instances belonging to particular class of

MinCost-SAT problems, the method performs very well and the new stopping rule provides a very effective way to reduce the number of iterations of the algorithm without observing any significant decay in the value of the objective function.

The work presented has to be considered preliminary, but it clearly indicates several research directions that we intend to pursue: the refinement of the dynamic estimate of the probability distribution of the solutions found by the algorithm, the comparative testing of instances of larger size, and the extension to other classes of problems. Last, but not least, attention will be directed toward the incorporation of the proposed heuristic into methods that are specifically designed to extract logic formulas from data and to the test of the performances of the proposed algorithm in this setting.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Chapter IV network flows. *Handbooks in operations research and management science*, 1:211–369, 1989.

[2] I. Arisi, M. D'Onofrio, R. Brandi, A. Felsani, S. Capsoni, G. Drovandi, G. Felici, E. Weitschek, P. Bertolazzi, and A. Cattaneo. Gene expression biomarkers in the brain of a mouse model for alzheimer's disease: Mining of microarray data by logic classification and feature selection. *Journal of Alzheimer's Disease*, 24 (4):721–738, 2011.

[3] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall PTR, 1998.

[4] M. Bazaraa and R. Langley. A dual shortest path algorithm. *SIAM Journal on Applied Mathematics*, 26(3):496–501, 1974.

[5] J. Beasley. A note on solving large p-median problems. *European Journal of Operational Research*, 21:270–273, 1985.

[6] P. Bertolazzi, G. Felici, and E. Weitschek. Learning to classify species with barcodes. *BMC Bioinformatics*, 10(14):S7, 2009.

[7] P. Bertolazzi, G. Felici, P. Festa, G. Fiscon, and E. Weitschek. Integer programming models for feature selection: New extensions and a randomized solution algorithm. *European Journal of Operational Research*, 250(2):389 - 399, 2016.

[8] D. P. Bertsekas. A distributed algorithm for the assignment problem. *Lab. for Information and Decision Systems Working Paper, M.I.T., Cambridge, MA*, 1979.

[9] D. P. Bertsekas. An auction algorithm for shortest paths. *SIAM Journal on Optimization*, 1(4):425–447, 1991.

[10] D. P. Bertsekas. *Linear network optimization: algorithms and codes*. MIT Press, 1991.

[11] D. P. Bertsekas and P. Tseng. *RELAX-IV: A faster version of the RELAX code for solving minimum cost flow problems*. Massachusetts Institute of Technology, Laboratory for Information and Decision Systems Cambridge, MA, 1994.

[12] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. The maximum clique problem. In *Handbook of combinatorial optimization*, pages 1–74. Springer, 1999.

[13] L. S. Buriol, M. G. C. Resende, and M. Thorup. Speeding up dynamic shortest-path algorithms. *INFORMS Journal on Computing*, 20(2):191–204, 2008.

[14] R. Cerulli, P. Festa, and G. Raiconi. Graph collapsing in shortest path auction algorithms. *Computational Optimization and Applications*, 18(3):199–220, 2001.

[15] R. Cerulli, P. Festa, and G. Raiconi. Shortest path auction algorithm without contractions using virtual source concept. *Computational Optimization and Applications*, 26(2):191–208, 2003.

[16] V. Cestarelli, G. Fiscon, G. Felici, P. Bertolazzi, and E. Weitschek. Camur: Knowledge extraction from rna-seq cancer data through equivalent classification rules. *Bioinformatics*, 32(5):697–704, Mar 2016.

[17] E. P. Chan and Y. Yang. Shortest path tree computation in dynamic graphs. *IEEE Transactions on Computers*, 58(4):541–557, 2009.

[18] B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical programming*, 73(2):129–174, 1996.

[19] B. V. Cherkassky, A. V. Goldberg, and C. Silverstein. Buckets, heaps, lists, and monotone priority queues. *SIAM Journal on Computing*, 28(4):1326–1346, 1999.

[20] N. Christofides. *An algorithmic approach*. Academic Press, Inc., New York, 1975.

[21] M. Coffin and M. Saltzman. Statistical analysis of computational tests of algorithms and heuristics. *INFORMS Journal on Computing*, 12(1):24–44, 2000.

[22] G. Dantzig. *Linear Programming And Extensions*. Princeton University Press, 1963.

[23] M. Daskin. *Network and Discrete Location: Models, Algorithms, and Applications*. Wiley, New York, 1995.

[24] T. Davidovic, D. Ramljak, M. Šelmic, and D. Teodorovic. Bee colony optimization for the p-center problem. *Computers & Operations Research*, 38(10):1367–1376, 2011.

[25] L. de Moura and N. Bjørner. Z3: An efficient smt solver. *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 337–340, 2008.

[26] F. Della Croce and R. Tadei. A multi-kp modeling for the maximum-clique problem. *European Journal of Operational Research*, 73(3):555–561, 1994.

[27] C. Demetrescu, A. V. Goldberg, and D. S. Johnson. 9th dimacs implementation challenge-shortest paths. *American Mathematical Society*, 2006.

[28] L. Di Puglia Pugliese and F. Guerriero. A computational study of solution approaches for the resource constrained elementary shortest path problem. *Annals of Operations Research*, 201(1):131–157, 2012.

[29] L. Di Puglia Pugliese and F. Guerriero. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200, 2013.

[30] R. Dial, F. Glover, D. Karney, and D. Klingman. A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees. *Networks*, 9(3):215–248, 1979.

[31] R. B. Dial. Algorithm 360: Shortest-path forest with topological ordering [h]. *Communications of the ACM*, 12(11):632–633, 1969.

[32] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[33] S. E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations research*, 17(3):395–412, 1969.

[34] M. Dyer and A. Frieze. A simple heuristic for the p-centre problem. *Operations Research Letters*, 3(6):285–288, 1985. ISSN 0167-6377.

[35] A. D'Andrea, M. D'Emidio, D. Frigioni, S. Leucci, and G. Proietti. Dynamically maintaining shortest path trees under batches of updates. In *International Colloquium on Structural Information and Communication Complexity*, pages 286–297. Springer, 2013.

[36] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[37] G. Felici and K. Truemper. A minsat approach for learning in logic domains. *INFORMS Journal on computing*, 14(1):20–36, 2002.

[38] G. Felici, D. Ferone, P. Festa, A. Napoletano, and T. Pastore. A GRASP for the minimum cost sat problem. In *International Conference on Learning and Intelligent Optimization*, pages 64–78. Springer, 2017.

[39] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[40] D. Ferone, P. Festa, F. Guerriero, and D. Laganá. The constrained shortest path tour problem. *Computers & Operations Research*, 74:64 – 77, 2016.

[41] D. Ferone, P. Festa, A. Napoletano, and T. Pastore. Reoptimizing shortest paths: From state of the art to new recent perspectives. In *2016 18th International Conference on Transparent Optical Networks (ICTON)*, pages 1–5, July 2016.

[42] D. Ferone, P. Festa, D. S. Marino, A. Napoletano, and T. Pastore. A GRASP for the max cut-clique problem. In *Proceedings of the 2017 Metaheuristics International Conference, 1-3*, 2017.

[43] D. Ferone, P. Festa, A. Napoletano, and T. Pastore. Shortest paths on dynamic networks: a survey. *Pesquisa Operacional*, 2017. inpress.

[44] D. Ferone, P. Festa, A. Napoletano, and M. G. C. Resende. A new local search for the p-center problem based on the critical vertex concept. In *International Conference on Learning and Intelligent Optimization*, pages 79–92. Springer, 2017.

[45] D. Ferone, P. Festa, A. Napoletano, and M. G. C. Resende. On the fast solution of the p-center problem. In *Transparent Optical Networks (ICTON), 2017 19th International Conference on*, pages 1–4. IEEE, 2017.

[46] P. Festa and S. Pallottino. A pseudo-random networks generator. Technical report, Department of Mathematics and Applications "R. Caccioppoli", University of Napoli FEDERICO II, Italy, 2003.

[47] P. Festa and M. G. C. Resende. An annotated bibliography of GRASP - Part I: Algorithms. *International Transactions in Operational Research*, 16(1):1–24, 2009.

[48] P. Festa and M. G. C. Resende. An annotated bibliography of GRASP - Part II: Applications. *International Transactions in Operational Research*, 16(2):131–172, 2009.

[49] P. Festa, P. M. Pardalos, L. S. Pitsoulis, and M. G. C. Resende. GRASP with path-relinking for the weighted maximum satisfiability problem. In *WEA*, volume 3503, pages 367–379. Springer, 2005.

[50] P. Festa, F. Guerriero, and A. Napoletano. An auction-based approach for the re-optimization shortest path tree problem. *European Journal of Operational Research*, 2017. submitted.

[51] M. Florian, S. Nguyen, and S. Pallottino. A dual simplex algorithm for finding all shortest paths. *Networks*, 11(4):367–378, 1981.

[52] L. R. Ford Jr. Network flow theory. Technical report, DTIC Document, 1956.

[53] L. R. Ford Jr and D. R. Fulkerson. *Flows in networks*. Princeton university press, 2015.

[54] Z. Fu and S. Malik. Solving the minimum-cost Satisfiability problem using sat based branch-and-bound search. In *2006 IEEE/ACM International Conference on Computer Aided Design*, pages 852–859, Nov 2006.

[55] G. Gallo. Reoptimization procedures in shortest path problem. *Rivista di Matematica per le Scienze Economiche e Sociali*, 3(1):3–13, 1980.

[56] G. Gallo and S. Pallottino. A new algorithm to find the shortest paths between all pairs of nodes. *Discrete Applied Mathematics*, 4(1):23–35, 1982.

[57] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.

[58] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. Wh Freeman New York, 2002.

[59] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533-549, 1986.

[60] F. Glover. Tabu search-part i. *ORSA Journal on Computing*, 1(3):190-206, 1989.

[61] F. Glover. Tabu search-part ii. *ORSA Journal on Computing*, 2(1):4-32, 1990.

[62] F. Glover. Tabu search and adaptive memory programming—Advances, applications and challenges. In *Interfaces in Computer Science and Operations Research*, pages 1-75. Springer, 1997.

[63] F. Glover and M. Laguna. Tabu search*. In *Handbook of Combinatorial Optimization*, pages 3261-3362. Springer, 2013.

[64] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653-684, 2000.

[65] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156-165. Society for Industrial and Applied Mathematics, 2005.

[66] B. Goldengorin, D. Krushinsky, and P. M. Pardalos. *Application of the PMP to Cell Formation in Group Technology*, pages 75-99. Springer New York, 2013.

[67] B. Goldengorin, A. Kocheturov, and P. M. Pardalos. *A Pseudo-Boolean Approach to the Market Graph Analysis by Means of the p-Median Model*, pages 77-89. Springer New York, 2014.

[68] S. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12(3):450-459, 1964.

[69] P. Hansen and N. Mladenovic. Variable neighborhood search for the p-median. *Location Science*, 5(4):207-226, 1997.

[70] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100-107, 1968.

[71] D. Hochbaum and D. Shmoys. A best possible heuristic for the k-Center problem. *Mathematics of Operations Research*, 10(2):180-184, 1985.

[72] I. Ioachim, S. Gelinas, F. Soumis, and J. Desrosiers. A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, 31(3):193-204, 1998.

[73] O. Kariv and S. Hakimi. An Algorithmic Approach to Network Location Problems. Part I: The p-Centers. *SIAM Journal on Applied Mathematics*, 37(3):513-538, 1979.

[74] O. Kariv and S. Hakimi. An Algorithmic Approach to Network Location Problems. Part II: The p-Medians. *SIAM Journal on Applied Mathematics*, 37(3):539-560, 1979.

[75] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85-103. Springer, 1972.

[76] M. Laguna and R. Marti. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11(1):44-52, 1999.

[77] M. Laguna, R. Martí, and V. Valls. Arc crossing minimization in hierarchical digraphs with tabu search. *Computers & operations research*, 24(12):1175-1186, 1997.

[78] T. L. Magnanti and R. T. Wong. Network design and transportation planning: Models and algorithms. *Transportation science*, 18(1):1-55, 1984.

[79] V. M. Manquinho and J. P. Marques-Silva. Search pruning techniques in SAT-based branch-and-bound algorithms for the binate covering problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21 (5):505-516, May 2002.

[80] V. M. Manquinho, P. F. Flores, J. P. M. Silva, and A. L. Oliveira. Prime implicant computation using Satisfiability algorithms. In *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on*, pages 232-239. IEEE, 1997.

[81] J. P. Marques-Silva and K. A. Sakallah. Grasp: a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506-521, May 1999.

[82] R. Martí and V. Estruch. Incremental bipartite drawing problem. *Computers & Operations Research*, 28(13):1287-1298, 2001.

[83] J. S. Martinich. A vertex-closing approach to the p-center problem. *Naval Research Logistics*, 35(2):185-201, 1988.

[84] P. Martins. Extended and discretized formulations for the maximum clique problem. *Computers & Operations Research*, 37(7):1348-1358, 2010.

[85] P. Martins. Cliques with maximum/minimum edge neighborhood and neighborhood density. *Computers & Operations Research*, 39(3):594-608, 2012.

[86] P. Martins, A. Ladrón, and H. Ramalhinho. Maximum cut-clique problem: Ils heuristics and a data analysis application. *International Transactions in Operational Research*, 22(5):775-809, 2015.

[87] R. Martins, V. Manquinho, and I. Lynce. Clause Sharing in Deterministic Parallel Maximum Satisfiability. In *RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*, 2012.

[88] R. Martins, V. M. Manquinho, and I. Lynce. Parallel search for maximum satisfiability. *AI Commun.*, 25(2):75–95, 2012.

[89] R. Martins, V. M. Manquinho, and I. Lynce. Clause sharing in parallel maxsat. In *Learning and Intelligent Optimization - 6th International Conference,LION 6, Paris, France, January 16-20, 2012, Revised Selected Papers*, pages 455–460, 2012.

[90] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, Jan. 1998.

[91] E. Minieka. The m-Center Problem. *SIAM Rev.*, 12(1):138–139, 1970.

[92] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210, 1995.

[93] N. Mladenovic, M. Labbé, and P. Hansen. Solving the p-Center Problem with Tabu Search and Variable Neighborhood Search. *Networks*, 42(April):48–64, 2003.

[94] N. Mladenovic, D. Urosevic, D. Pérez-Brito, and C. G. García-González. Variable neighbourhood search for bandwidth reduction. *European Journal of Operational Research*, 200(1):14 – 27, 2010.

[95] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA, 2001. ACM.

[96] G. Nannicini, P. Baptiste, D. Krob, and L. Liberti. Fast paths in dynamic road networks. *Proceedings of ROADEF*, 8:1–14, 2008.

[97] G. Nannicini, D. Delling, D. Schultes, and L. Liberti. Bidirectional a* search on time-dependent road networks. *Networks*, 59(2):240–251, 2012.

[98] A. Napoletano, A. Martínez-Gavara, P. Festa, and R. Martí. Heuristics for the constrained incremental graph drawing problem. *European Journal of Operational Research*, 2017. submitted.

[99] T. A. J. Nicholson. Finding the shortest route between two points in a network. *The computer journal*, 9(3):275–280, 1966.

[100] S. Pallottino and M. G. Scutellá. Shortest path algorithms in transportation models: classical and innovative aspects. In *Equilibrium and advanced transportation modelling*, pages 245–281. Springer, 1998.

[101] S. Pallottino and M. G. Scutellá. A new algorithm for reoptimizing shortest paths when the arc costs change. *Operations Research Letters*, 31(2):149–160, 2003.

[102] E. G. Pardo, N. Mladenovic, J. J. Pantrigo, and A. Duarte. Variable formulation search for the cutwidth minimization problem. *Applied Soft Computing*, 13(5): 2242 – 2252, 2013.

[103] S. Pettie and V. Ramachandran. Command line tools generating various families of random graphs. *American Mathematical Society*, 2006.

[104] P. P. Pham and S. Perreau. Performance analysis of reactive shortest path and multipath routing mechanism with load balance. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 1, pages 251–259. IEEE, 2003.

[105] M. Pipponzi and F. Somenzi. An iterative algorithm for the binate covering problem. In *Proceedings of the European Design Automation Conference, 1990., EDAC.*, pages 208–211, Mar 1990.

[106] W. Pullan. Phased local search for the maximum clique problem. *Journal of Combinatorial Optimization*, 12(3):303–323, 2006.

[107] G. Ramalingam and T. Reps. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms*, 21(2):267–305, 1996.

[108] G. Reinelt. Tsplib—a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[109] M. G. C. Resende and T. A. Feo. A GRASP for Satisfiability. In *Cliques, Coloring and Satisfiability: the second DIMACS implementation challenge*. Citeseer, 1996.

[110] M. G. C. Resende and C. C. Ribeiro. GRASP with path-relinking: Recent advances and applications. In *Metaheuristics: progress as real problem solvers*, pages 29–63. Springer, 2005.

[111] M. G. C. Resende and C. C. Ribeiro. *Optimization by GRASP: Greedy randomized adaptive search procedures*. Springer, 2016.

[112] M. G. C. Resende and R. F. Werneck. A Hybrid Heuristic for the p-Median Problem. *Journal of Heuristics*, 10(1):59–88, 2004.

[113] C. C. Ribeiro, I. Rosseti, and R. C. Souza. Probabilistic stopping rules for GRASP heuristics and extensions. *International Transactions in Operational Research*, 20(3):301–323, 2013.

[114] S. Salhi and A. Al-Khedhairi. Integrating heuristic information into exact methods: The case of the vertex p-centre problem. *Journal of the Operational Research Society*, 61(11):1619–1631, 2010.

[115] J. Sánchez-Oro, A. Martínez-Gavara, M. Laguna, A. Duarte, and R. Martí. Variable neighborhood descent for the incremental graph drawing. *Electronic Notes in Discrete Mathematics*, 58:183–190, 2017.

[116] J. Sánchez-Oro, A. Martínez-Gavara, M. Laguna, R. Martí, and A. Duarte. Variable neighborhood scatter search for the incremental graph drawing problem. *Computational Optimization and Applications*, pages 1–23, 2017.

[117] F. W. Scholz. Maximum likelihood estimation. 2004.

[118] M. Schwartz. *Telecommunication networks: protocols, modeling and analysis*, volume 7. Addison-Wesley Reading, MA, 1987.

[119] M. Servit and J. Zamazal. Heuristic approach to binate covering problem. In *Proceedings The European Conference on Design Automation*, pages 123–129, Mar 1992.

[120] N. Sorensson and N. Een. Minisat v1.13 - a sat solver with conflict-clause minimization. Technical report, 2005(53).

[121] P. Toth and D. Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

[122] K. Truemper. *Design of Logic-based Intelligent Systems*. Wiley-Interscience publication. Wiley, 2004.

[123] T. Villa, T. Kam, R. K. Brayton, and A. L. Sangiovanni-Vincenteili. Explicit and implicit algorithms for binate covering problems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(7):677–691, Jul 1997.

[124] E. Weitschek, G. Felici, and P. Bertolazzi. Mala: A microarray clustering and classification software. In *2012 23rd International Workshop on Database and Expert Systems Applications*, pages 201–205, Sept 2012.

[125] E. Weitschek, A. Lo Presti, G. Drovandi, G. Felici, M. Ciccozzi, M. Ciotti, and P. Bertolazzi. Human polyomaviruses identification by logic mining techniques. *Virology Journal*, 9(1):58, 2012.

[126] E. Weitschek, G. Fiscon, and G. Felici. Supervised dna barcodes species classification: analysis, comparisons and results. *BioData Mining*, 7(1):4, 2014.

[127] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1 (6):80–83, 1945.

[128] X. Zhang, Z. Zhang, Y. Zhang, D. Wei, and Y. Deng. Route selection for emergency logistics management: A bio-inspired algorithm. *Safety science*, 54:87–91, 2013.