Logic Learning and Optimized Drawing: Two Hard Combinatorial Problems

Tommaso Pastore

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 2018

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Whether it be while surfing the Internet, registering financial transactions or processing medical records, we are constantly collecting new data. The process of information gathering, as acquisition of derived knowledge, substantially represents the basis of modern science and the foundation of the learning process itself. At the same time, the amount of data we produce nowadays is unprecedented. According to a column featured on Forbes magazine[1], there are $2.5 \cdot 10^{18}$ bytes of data created each day, with the last two years alone that generated the 90 percent of the data in the world. As a consequence of this, information extraction from large datasets or networks is a recurring operation in countless fields, establishing itself as a cornerstone process in an organic revolution that over the course of time brought us from tally sticks[2] to modern data centers.

The purpose leading this thesis is to ideally follow the data flow along its journey, describing some hard combinatorial problems that arise from two key processes, one consecutive to the other: information extraction and representation. Moreover, having in mind the growing size of information networks, the approaches here considered will focus mainly on metaheuristic algorithms, to address the need for fast and effective optimization methods.

The core of this thesis is devoted to the study of Supervised Learning in Logic Domains and the Max Cut-Clique Problem, as examples of NP-hard data extraction processes, and two different Graph Drawing Problems. Moreover, stemming from these main topics, other additional themes will be addressed, namely two different approaches to handle Information Variability in Combinatorial Optimization Problems (COPs), and Topology Optimization

[1] Bernard Marr - “How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read”, Forbes Magazine - July 2018

[2] Grandell, Axel. "The reckoning board and tally stick." The Accounting Historians Journal 4.1 (1977): 101-105.

for the design of stress-constrained lightweight structures. The structure of the thesis and the interplays among its topics are depicted in Figure 1.1 and described in the following.

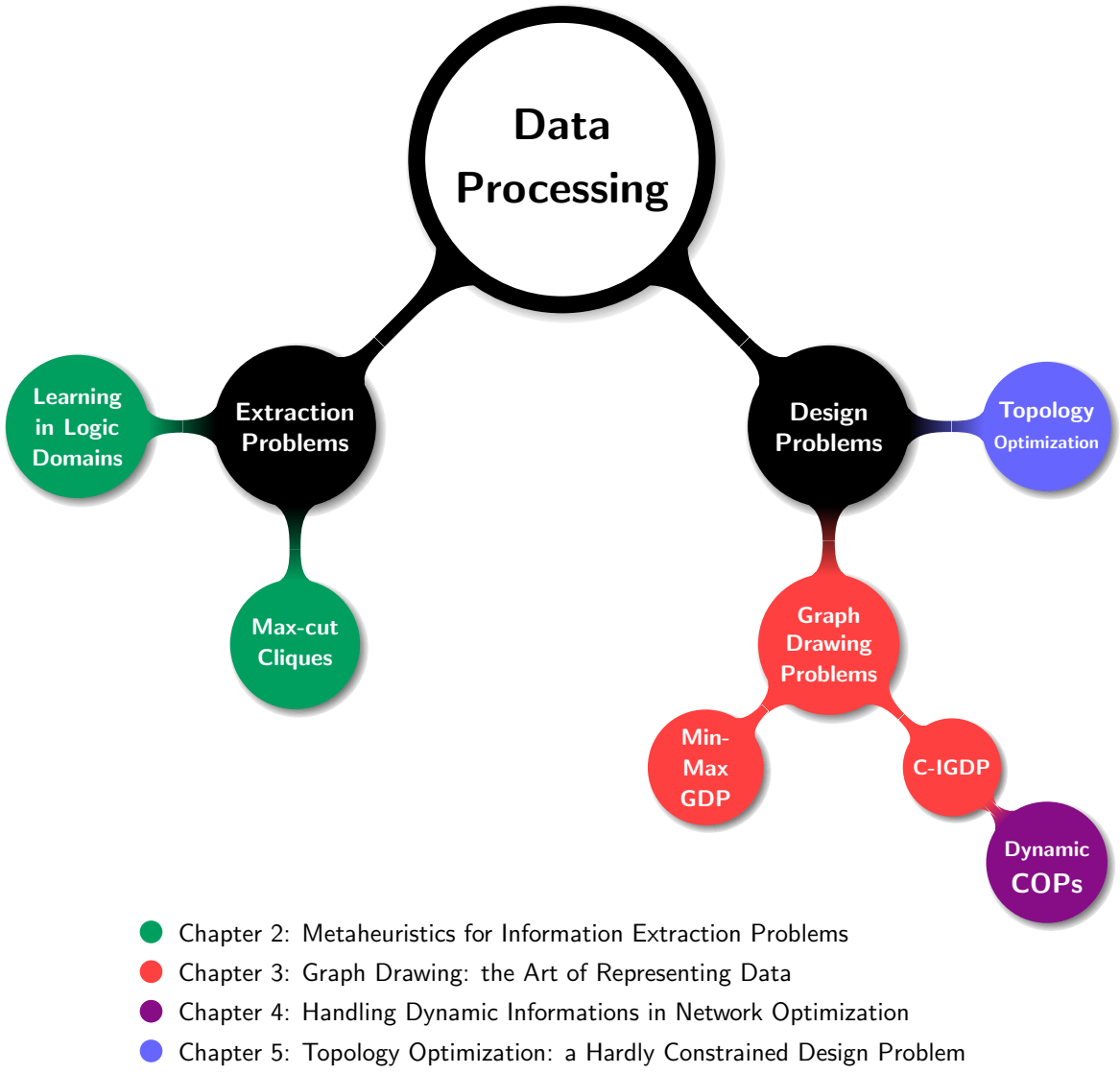

Fig. 1.1 Mindmap representing the structure of the thesis.

The first problem addressed in this thesis is logic supervised learning. In this scenario, studied in Chapter 2, we have a dataset of samples, each represented by a finite number of logic variables, and we show how 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 use of MinCost-SAT for learning propositional formulae from data is described in [48] and [151], and it proved to be very effective in several applications, particularly on those derived from biological and medical data analysis [10, 3, 155, 156, 154].

One of the main drawbacks of this approach 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 to analyze steadily increases. While the literature proposes both exact approaches ([63, 127], [151], and [153]) and heuristics ([138]), 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.

Chapter 2 will describe a GRASP-based metaheuristic algorithm designed to solve MinCost-SAT problems that arise in supervised learning. In doing so, 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 metaheuristic 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.

As a second example of information extraction problem, the second part of Chapter 2 studies the Max Cut-Clique Problem (MCCP). This problem is a variation of the classical Max Clique Problem with several applications in the area of market basket analysis. We formally describe the MCCP and detail the main characteristics of our hybrid heuristic solution strategy, presenting a comparison between our algorithm and the state-of-the-art.

Chapter 3 deals with data representation, introducing results obtained for two different Graph Drawing Problems (GDPs). Graph drawing is a well established area of computer science that consists in obtaining an automatic representation of a given graph, described in terms of its vertices and edges. In the scientific literature many aesthetic criteria have been proposed to identify the desirable properties that a good representation has to fulfill, with the most common being: edge crossings, graph area, edge length, edge bends, and symmetries. Their main objective is to achieve readable drawings in which it is easy to obtain or extract information. This goal is particularly critical in graphs with hundreds of vertices and edges, in which an improper layout could be extremely hard to analyze. The graph drawing area is very active, and an excellent resource on the topic is the book by Di Battista et al. [6], where many graph drawing models and related applications are introduced.

There are different drawing conventions to represent a graph. In some settings, the vertices of the graph are typically drawn in a circle, such as for example in genetic interaction networks. Figure 1.2 represents the two hundred and forty-five interactions found among 16

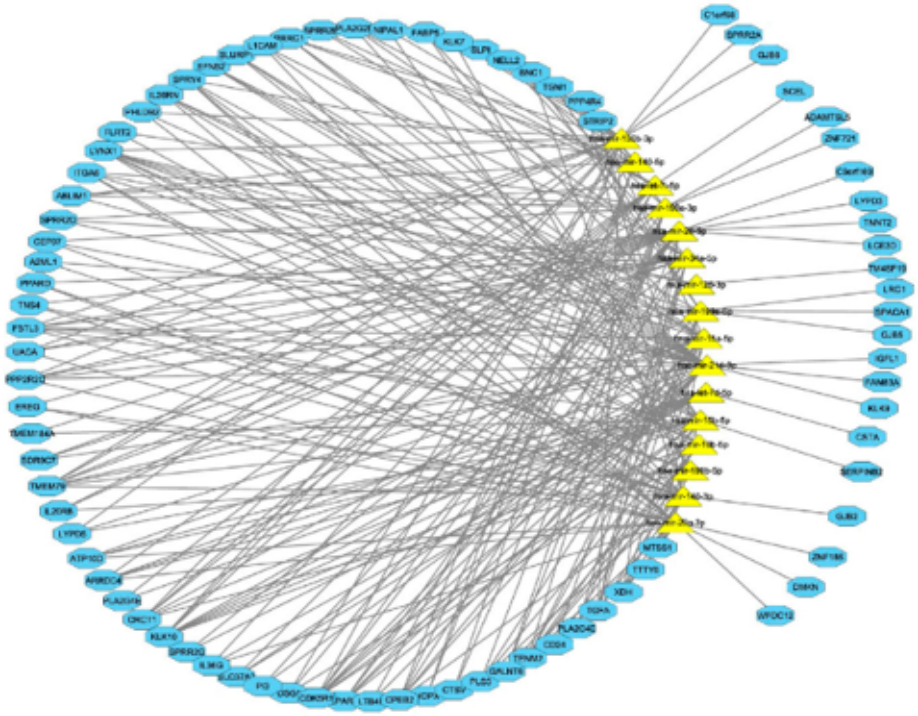

Fig. 1.2 Circular representation of a graph [27].

micro-RNAs and 84 genes (see [27]). In other cases, the data has a clear sequential nature, like in workforce scheduling problems, where a set of tasks cannot be undertaken before the completion of some previous ones. Whenever the data is characterized by this kind of precedence relationships, the natural representation for the corresponding network is given by a hierarchical graph. Application fields which benefits from this graph layout can be found in Table 1.1. For example, in the field of operational inter-period material activities, a set of operational multi-period scenarios can be represented with a special graph rooted with replicas of the related strategic node, as it can be seen in Figure 1.3 (e.g see [47]).

The Hierarchical Directed Acyclic Graph (HDAG) representation is obtained by arranging the vertices on a series of equidistant vertical lines called layers in such a way that all edges point in the same direction. Note that working with hierarchies is not a limitation, since there exists a number of procedures to transform any directed acyclic graph (DAG) into a layered or hierarchical graph [5, 145].

The crossing minimization problem in hierarchical digraphs has received a lot of attention. Even the problem in bipartite graphs has been extensively studied for more than 40 years, beginning with the Relative Degree Algorithm introduced in Carpano [25]. Early heuristics were based on simple ordering rules, reflecting the goal of researchers and practitioners of

Context	References	Description
Workflow visualization	[152]	Representation of the work to be executed by the project team.
Software engineering	[29, 22]	Representation of calling relationships between subroutines in a computer program.
Database modeling	[85]	Definition of data connections and system processing and storing diagrams.
Bioinformatics	[98]	Representations of proteins and other structured molecules with multiple functional components.
Process modeling	[81, 51]	Analytical representation or illustration of an organization's business processes.
Network management	[122, 92]	Representation of the set of actions that ensures that all network resources are put to productive use as best as possible.
VLSI circuit design	[13]	Representation of the design of integrated circuits (ICs) which are essential to the production of new semiconductor chips.
Decision diagrams	[115, 116]	Definition of logic synthesis and formal verification of logic circuits.

Table 1.1 Hierarchical Graphs applications as listed in [119].

quickly obtaining solutions of reasonable quality. However, the field of optimization has recently evolved introducing complex methods, both in exact and heuristic domains. The crossing problem has benefited from these techniques, and advanced solution strategies have been proposed in the last 20 years to solve it. We refer the reader to Martí [113], Di Battista et al. [6], or Chimani et al. [30] to mention relatively recent developments.

In Chapter 3 we focus on two different variants of the crossing problem: (i) the min-max GDP [143], and (ii) the Constrained Incremental GDP.

The interest in the min-max GDP, originally called the *bottleneck crossing minimization*, arose in the context of VLSI circuits design in which it is more appropriate to minimize the maximum number of crossings over all edges (min-max) rather than the sum of the edge crossings over all the graph (min-sum). As stated in Bhatt and Leighton [13], an undesirable feature of VLSI layouts is the presence of a large number of wire crossings. More specifically, wires that are crossed by many others are susceptible to cross-talk, when all the crossing wires simultaneously carry the same signal, thus deteriorating the circuit

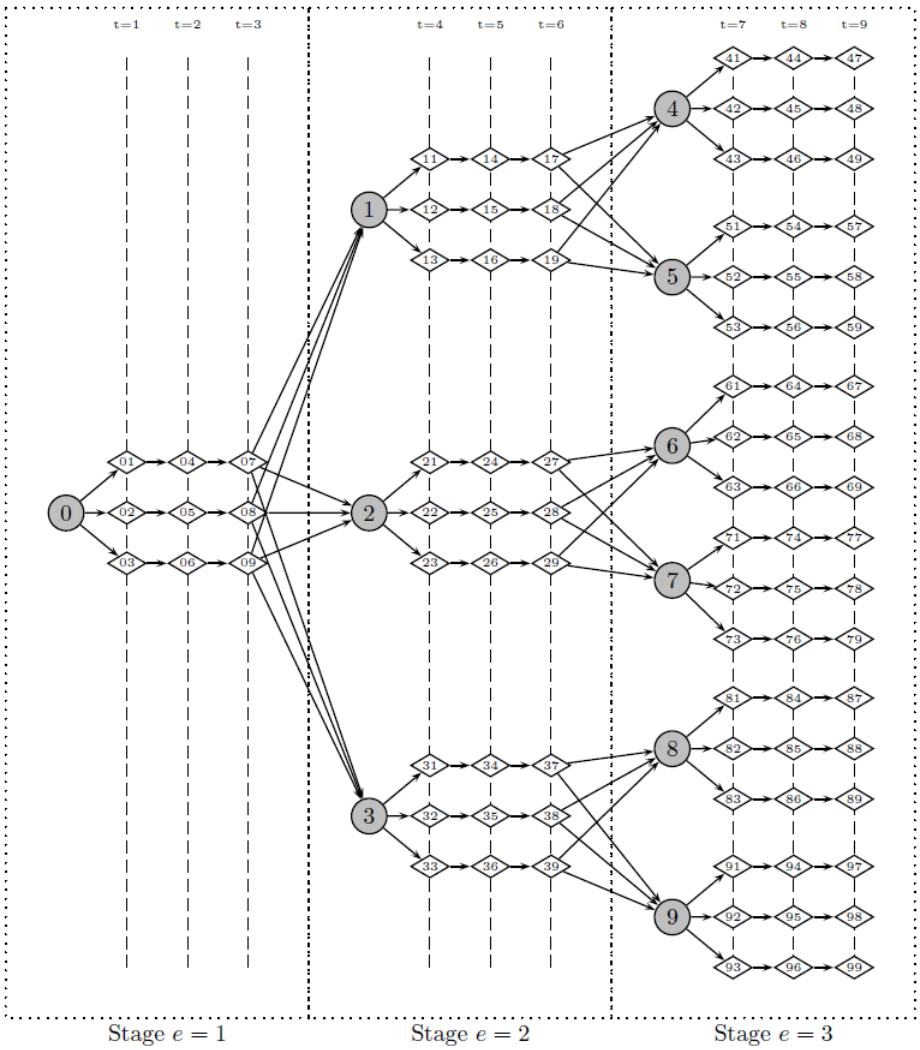

Fig. 1.3 Strategic multistage scenario tree with tactical multiperiod graphs rooted with strategic nodes.

performance. On the other hand, if the number of wire crossings is small, the number of contact-cuts is also small, thus providing a better signal. Therefore, in order to attain a good performance over the network it is critical that no edge has a large number of crossings, more than the overall sum of crossings is small. Remarkably, the solution of the min-max problem is also useful in general graph drawing softwares, where zooming highlights a specific area of the graph where it is then desirable to have locally a low number of crossings.

On the other hand, the Constrained Incremental GDP (C-IGDP) is a variation of the classical min-sum GDP, that addresses the need of properly handling graph drawing in areas such as project management, production planning or CAD software, where changes

in project structure result in successive drawings of similar graphs. The so-called mental map of a drawing reflects the user's ability to create a mental structure with the elements in the graph. When elements are added to or deleted from a graph, the user has to adjust their mental map to become familiar with the new graph. The dynamic graph drawing area is devoted to minimizing this effort. As described in [18], considering that a graph has been slightly modified, applying a graph drawing method from scratch would be inefficient and could provide a completely different drawing, thus resulting in a significant effort for the user to re-familiarize her/himself with the new map. Therefore, models to work with dynamic or incremental graphs have to be used in this context.

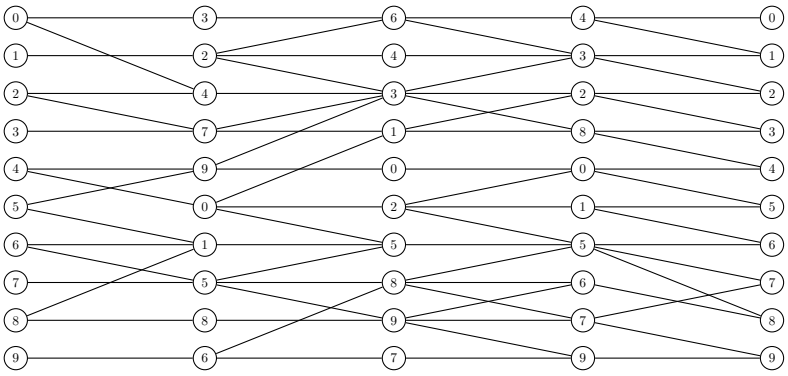

Fig. 1.4 Optimal drawing for crossing minimization of a given graph.

To illustrate the incremental problem, we consider the hierarchical drawing in Figure 1.4, which shows the optimal solution of the edge crossing minimization problem of a graph with 50 vertices and 5 layers. We increment this graph now by adding 20 vertices (4 in each layer), and their incident edges. Figure 1.5 shows the optimal solution of the edge crossing minimization problem of the new graph, where the new vertices and edges are represented with dotted lines.

Although the number of crossings in Figure 1.5 is minimum, 99, this new drawing was created from scratch, and it ignores the position of the vertices in the original drawing (the one in Figure 1.4). For example, vertex 6 in the first layer is in position 7 in Figure 1.4, but in position 10 in Figure 1.5. We can say that Figure 1.5 does not keep the mental map of the user familiarized with Figure 1.4. Therefore, in line with the dynamic drawing conventions [18], we propose reducing the number of crossings of the new graph while keeping the original vertices close to their positions in Figure 1.4.

Regarding hierarchical graphs, previous efforts only preserve the relative positions of the original vertices [106]. As will be shown, this can result in poor incremental drawings (i.e., the mental map of the drawing is not properly kept). The approach described in this thesis

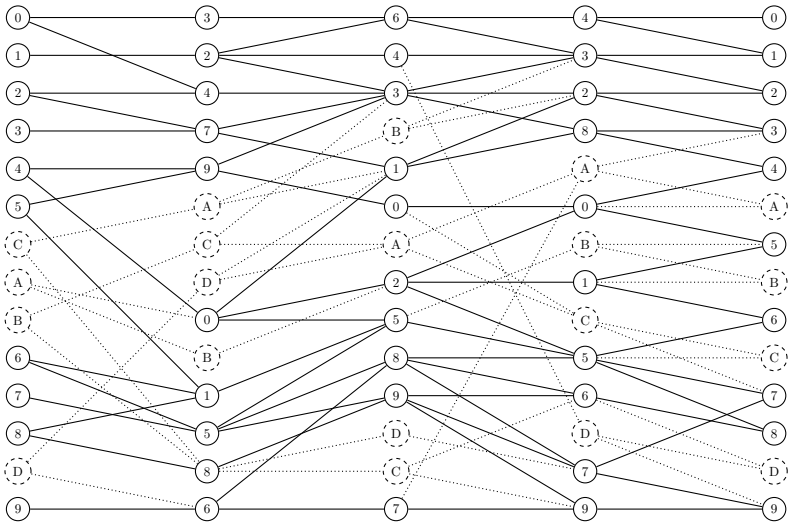

Fig. 1.5 Optimal drawing for crossing minimization of the incremented graph.

considers a robust model with constraints on both the relative and the absolute position of the original vertices when minimizing the number of edge crossings in a sequence of drawings. In this way, we help the user to keep his or her mental map when working with a drawing where successive changes occur. In particular, our model restricts the relative position of the original vertices with respect to their position in the initial drawing (as in [106]), and also restricts their absolute position within a short distance of their initial position (as in [6] in the context of orthogonal graphs).

As in the case of the C-IGDP, a challenge of growing consideration in the operations research community consists in the inclusion of information variability in classical Combinatorial Optimization Problems. Given the non-deterministic nature of real-world scenarios, properly addressing information variability (or stochasticity) is essential for embedding optimization techniques into each of the supply-chain systems that determine production, scheduling, distribution, location-allocation, etc. Chapter 4 studies two iconic problems in the landscape of Dynamic Network Optimization: the Vehicle Routing Problem with stochastic demands, and the Reoptimization of Shortest Paths; presenting a simheuristic approach for the former, and surveying the research efforts made in the latter.

The vast majority of combinatorial problems are computationally intractable by nature, and the Vehicle Routing Problem (VRP) is no exception. For a summary of the existing approaches to solving the classical VRP, we refer the reader to three excellent resources on the topic: [149], [95], and [75]. Due to the complexity of the VRP, the typical solution techniques for large-scale instances mainly belong to the class of heuristic and metaheuristic algorithms. In our approach we embrace the Simheuristic paradigm, in which heuristic

algorithms are hybridized with a multi-stage Monte Carlo simulation to properly evaluate solution quality. More specifically, we couple a GRASP with biased randomization (BR-GRASP) with stochastic simulation in order to obtain reliable and competitive routing plans when considering customers with stochastic demands.

On the other hand, the Shortest Path Problem (SPP) established itself as one of the most representative and widespread polynomially solvable problems of operations research. Reoptimizing shortest paths on dynamic graphs consists in the solution of a sequence of shortest path problems, where each problem differs only slightly from the previous one. 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, ad-hoc algorithms can efficiently use information gathered in previous computations and speed up the solution process.

If in Chapter 3 we address the problem of designing an optimized representation for information networks, Chapter 5 presents a solution for a different kind of design problem: the shaping of lightweight stress-constrained structures by means of Topology Optimization. Topology optimization is the most broad form of structural optimization. In general terms it aims at determining efficient material layouts within a given design space, starting from very few prescribed specifications such as load case and boundary conditions [134].

The development of additive manufacturing (AM) and 3D printing technologies makes possible to produce extremely complex structures which until recently, using traditional production methods, would have been impossible to accomplish or would require unreliable efforts and unacceptable costs. Component design by means of topology optimization from the earliest stages of their conceptual development could lead to major breakthroughs, maximizing the potential of 3D printing.

More specifically, in Chapter 5 we consider the topology optimization of Reinforced Concrete (RC) elements. In this scope, the primary goal for designers is to obtain the lightest structure while having a straight control on the stress levels. It is possible to notice how most of the optimization methods are limited to compliance minimization problems. This problem is considered classical in the optimization community and simpler to be solved if compared to the stress constrained one. Stress problems, indeed, bear more challenging difficulties, such as high non-linearity [124, 97]. On the other hand, even if some studies tackle the stress problem, in most of those cases the stress constraints there defined are based on the classical Von Mises stress, which is more appropriate to describe isotropic materials, such as steel, or other grouping strategies. See for example [43, 125, 144].

Chapter 5 provides a formal description of the structural optimization problem of interest, with particular emphasis on the form used in the imposition of the stress constraints, and

presents an iterative heuristic algorithm with its solutions for a wide range of load cases and material properties.

Lastly, conclusions are drawn in Chapter 6.

Chapter 2

Metaheuristics for Information Extraction Problems

Right after being collected and appropriately stored, a data stream is ready to start its journey and be processed, to extract its most relevant informations and recurring patterns. The focus of the present Chapter consists in the study of the contributions that operations research –and more specifically metaheuristic techniques– can bring in the solutions of two widely different mining problems. The problems of interest are respectively concerned with the extraction of boolean formulae in supervised learning, and the study of special cliques in graphs of large size.

2.1 From Logic Learning to Minimum Cost Satisfiability

The goal of logic supervised learning consists in inductively determining separations among sets of data, starting from a set of labeled training observations. The approach adopted in this thesis, as described in [48] and applied in [49], reduces the learning problem to a sequence of satifiability instances.

The logic data, or the observations to be used in the in the inductive learning process, are defined as vectors $r \in \{-1, 0, 1\}^n$, and called records. In contrast with classical Boolean vectors, the components r_i can not only indicate the presence ($r_i = 1$) or absence ($r_i = -1$) of a certain feature i, but additionally include the possibility of an absence of informations ($r_i = 0$).

For each of such records, we define index sets r^+, r^-, and r^0, containing the indices i for which the elements r_i are respectively equal to 1, -1, and 0. Along with a record

r comes a logic outcome, either True or False, to represent the presence or absence of a given property in the observation. We collect the records r for which the property is absent in a set A, and those for which it is present in a set B.

The aim in this supervised learning scenario is to extract a Boolean formula from the observation records, that completely separates the set of positive examples B from the set A. This separation process takes places by means of a set of vectors, called "separating set". To properly define the concept of separation, we introduce the property of being *nested*.

Let f be a $\{-1,0,1\}^n$ vector, then f is said to be *nested* in a $\{-1,0,1\}^n$ vector g, if and only if for any f_i equal to 1 or -1, the corresponding entry of g , g_i, is such that $g_i = f_i$. Conversely, f is not nested in g if and only if there is a $f_i = 1$ in f, such that $g_i = -f_i$ or $g_i = 0$.

On the base of the nested relation we are able to define separation.

Let A and B be sets of records of the same size n. A record s *separates* $b \in B$ from A if and only if

S1. s is not nested in a, $\forall a \in A$;

S2. s is nested in b.

Then, a set of records S separates the whole set B from A if, for each record $s \in S$, s is not nested in any $a \in A$, and for each $b \in B$ it exists a $s_b \in S$ such that s_b separates b from A. As proven in [48], the existence of such a separating set S of B from A is guaranteed if and only if no record $b \in B$ is nested in any record $a \in A$.

The use of satisfiability models for learning propositional formulae gathered several successes, as reported in [10, 3, 155, 156, 154]. In the following we describe how to obtain a specific variation of the classical SAT, the Minimum Cost Satisfiability Problem (MinCostSAT), to model supervised learning in Logic Fields.

Let A and B be non-empty sets of $\{-1,0,1\}^n$ records. The core idea of this approach is to iteratively find logic records s to separate a growing portion $B^* \subseteq B$ from A, until a separating set S is obtained.

To write a proper satisfiability model we introduce for each $i = 1,\ldots,n$ two Boolean variables, p_i and q_i. These variables are tightly connected with the component of the separating vector s to be found, in the sense that

$$\begin{aligned} &s_i = 1, \quad \text{if } p_i = \mathrm{True} \wedge q_i = \mathrm{False};\\ &s_i = -1, \quad \text{if } p_i = \mathrm{False} \wedge q_i = \mathrm{True};\\ &s_i = 0 \quad \text{if } p_i = q_i = \mathrm{False}. \end{aligned} \tag{2.1}$$

The case $p_i = q_i = True$ has not a clear interpretation, so it is ruled out enforcing

$$\neg p_i \vee \neg q_i, \quad \forall i = 1, \ldots, n. \tag{2.2}$$

Since s is a separating vector, it has to fulfill conditions S1 and S2. In the following we take into account those two separately, and obtain two different set of constraints to be included in our SAT model.

Condition S1 requires that s is not nested in any $a \in A$. This means that exists at least an index i such that one of the three following conditions happens:

- $a_i = 1$ and $s_i = -1$;
- $a_i = -1$ and $s_i = 1$;
- $a_i = 0$ and $s_i = 1 \vee s_i = -1$.

So, for this index i, in terms of p_i and q_i, using 2.1 and 2.2 we obtain

$$\begin{aligned} &a_i = 1 \Rightarrow q_i \\ &a_i = -1 \Rightarrow p_i \\ &a_i = 0 \Rightarrow q_i \vee p_i. \end{aligned} \tag{2.3}$$

Since these three conditions have to hold for each $a \in A$ and at least an index i, they can be collected in the following disjunctions:

$$\left(\bigvee_{i \in a^+ \cup a^0} q_i \right) \vee \left(\bigvee_{i \in a^- \cup a^0} p_i \right), \quad \forall a \in A \tag{2.4}$$

that completely enclose condition S1. On the other hand, to take into account condition S2, we have that if s separates a record $b \in B$, s has to be nested in b. Following a similar approach to the one presented for condition S1, we have

$$\begin{aligned} &b_i = 1 \Rightarrow \neg q_i \\ &b_i = -1 \Rightarrow \neg p_i \\ &b_i = 0 \Rightarrow \neg p_i \vee \neg q_i \end{aligned} \tag{2.5}$$

At the same time, in our pursuit for a separating record s, there is no guarantee that a single record is able to separate all the elements of B from A. For this reason, it is not possible to include, for each $b \in B$, (2.5) as constraint in a satisfiability model. As a workaround, we define for each b an additional Boolean variable, d_b, that determines

whether s must separate b from A. More specifically, $d_b = True$ means that s needs not to separate b from A, while $d_b = False$ requires that separation. Using variable d_b in (2.5), we get to the new following form

$$\begin{aligned} &\neg q_i \vee d_b, \quad \forall i \in b^+ \cup b^0 \\ &\neg p_i \vee d_b, \quad \forall i \in b^- \cup b^0. \end{aligned} \tag{2.6}$$

As mentioned earlier, the idea behind this approach is to decompose the problem of finding a separating set into a sequence of subproblems, each of which determines a vector s that separates a nonempty subset of B from A. This idea can be translated in an optimization problem if at each step we try to maximize the number of elements of B separated from A. Rendering this insight in terms of variables d_b, we want a separating record s, i.e. a vector that satisfies (2.4) and (2.6), and that does so with the minimal number of d_b variables with a $True$ value.

For each $b \in B$, we define a cost function $c_b(d_b)$ that is equal to 1 if d_b is $True$, and 0 otherwise. Using the costs c_b and in light of constraints (2.4) and (2.6), we obtain the following Satisfiability Problem:

$$z = \min \sum_{b \in B} c_b(d_b)$$

subject to:

$$\begin{aligned} \bigvee_{i \in a^+ \cup a^0} q_i \vee \bigvee_{i \in a^- \cup a^0} p_i \quad & \forall a \in A \\ \neg q_i \vee d_b, \quad & \forall i \in b^+ \cup b^0 \\ \neg p_i \vee d_b, \quad & \forall i \in b^- \cup b^0. \end{aligned} \tag{2.7}$$

Problems of this family are called Minimum Cost Satisfiability Problems. In their most general form they can be stated as follows.

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 MinCostSAT 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:

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

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 MinCostSAT problem whose costs c_i are all equal to 0. Furthermore, the decision version of the MinCostSAT problem is NP-complete [63]. While the Boolean satisfiability problem is an evergreen in the landscape of scientific literature, MinCostSAT has received less attention.

2.2 A GRASP for MinCostSAT

GRASP is a well established iterative multistart metaheuristic method for difficult combinatorial optimization problems [50]. The reader can refer to [57, 58] for a study of a generic GRASP metaheuristic 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 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 2.1 depicts the pseudo-code of a generic GRASP for a minimization problem.

In order to allow a better and easier implementation of our GRASP, we treat the MinCost-SAT as particular covering problem with incompatibility constraints. Indeed, we consider each literal $(x, \neg x)$ as a separate element, and a clause of the CNF 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.

```
1  Algorithm GRASP(β)
2      x* ← Nil ;
3      z(x*) ← +∞ ;
4      while a stopping criterion is not satisfied do
5          Build a greedy randomized solution x ;
6          x ← LocalSearch(x) ;
7          if z(x) < z(x*) then
8              x* ← x ;
9              z(x*) ← z(x) ;
10     return x*
```

Fig. 2.1 A generic GRASP for a minimization problem.

The construction phase adds a literal at a time, until all clauses are covered or no more literals can be assigned. At each iteration of the construction, 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, $penalty(\cdot)$, 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. We then compute:

$$penalty(x) = \frac{c(x) + cr(\neg x)}{cr(x)}. \tag{2.8}$$

This 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 $penalty(x)$, 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 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 2.2, line 3) in the `construct-solution` function pseudo-coded

```
1  Function construct-solution(𝒞, X, β)
       /* 𝒞 is the set of uncovered clauses                          */
       /* X is the set of candidate literals                         */
2      s ← ∅ ;
3      while C ≠ ∅ do
4          if c ∈ 𝒞 can be covered only by x ∈ X then
5              s ← s ∪ {x};
6              X ← X \ {x, ¬x};
7              𝒞 ← 𝒞 \ {c̄ | x ∈ c̄};
8          else
9              compute penalty(x) ∀ x ∈ X;
10             th ← min_{x∈X}{penalty(x)} + β(max_{x∈X}{penalty(x)} − min_{x∈X}{penalty(x)}) ;
11             RCL ← { x ∈ X: penalty(x) ≤ th } ;
12             x̂ ← rand(RCL) ;
13             s ← s ∪ {x̂};
14             X ← X \ {x̂, ¬x̂};
15             𝒞 ← 𝒞 \ {c̄ | x̂ ∈ c̄};
16     return s
```

Fig. 2.2 Pseudo-code of the GRASP construction phase.

in Fig. 2.2 runs m times and in each run the most expensive operation consists in the construction of the RCL. Therefore, the total computational complexity is $\mathcal{O}(m \cdot n)$.

In the local search 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.

2.3 Bernoulli Take the Wheel! A probabilistic Stopping Rule

Although being very fast and powerful, most metaheuristics present a shortcoming in the effectiveness of their stopping rule. Usually, 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. In this algorithm, we propose a probabilistic stopping criterion, inspired by [132].

The stopping criterion is composed of two phases, described in the next subsections. 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 $\mathrm{f}_{\mathcal{X}}(\cdot)$ 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.

The first step to be performed in order to properly represent the random variable $\mathcal{X}$ with a theoretical distribution consists in an empirical observation of the algorithm. Examining the objective function values obtained at the end of each iteration, and counting up the respective frequencies, it is possible to select a promising parametric family of distributions. Afterwards, by means of a Maximum Likelihood Estimation (`MLE`), see for example [137], a choice is made regarding the parameters characterizing the best fitting distribution of the chosen family.

In order to carry on the empirical analysis of the objective function value obtained in a generic iteration of GRASP, which will result in a first guess concerning the parametric family of distributions, we represent the data obtained in the following way.

Let $\mathcal{I}$ be a fixed 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 dealing with a minimization problem, it is harder to find good quality solutions, whose cost is small in term of objective function, rather than expensive ones. This means that 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 $\max(\mathcal{Z})$. In order to represent the values in $\mathcal{Z}$ with a positive distribution function, that presents higher frequencies in a right neighborhood

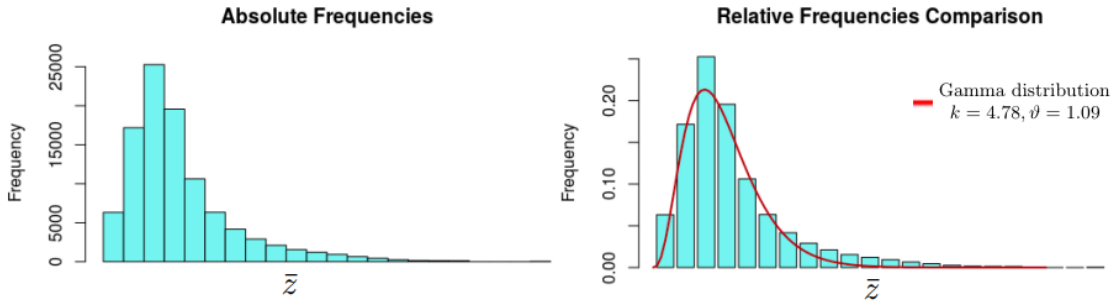

Fig. 2.3 Empirical analysis of frequencies of the solutions.

Function `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. 2.4 Fitting data procedure.

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 of the following transformation:

$$\bar{z} = \max(\mathcal{Z}) - z, \ \forall \ z \in \mathcal{Z}. \tag{2.9}$$

The behavior of the distribution of $\bar{z}$ in our instances has then a very recognizable behavior. A representative of such distribution is given in Figure 2.3 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 reasonable educated guess for our random variable.

Once we have chosen the gamma distribution family, we estimate its parameters performing a `MLE`. In order to accomplish the estimation, we collect an initial sample of solution values and on-line execute a function, developed in `R` (whose pseudo-code is reported in Figure 2.4), which carries out the `MLE` and returns the characteristic shape and scale parameters, k and θ, which pinpoint the specific distribution of the gamma family that best suits the data.

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.

```
1 Function improve-probability(k, θ, z*)
    | /* z* is the value of the incumbent                    */
2   | p ← pgamma(z*, shape = k, scale = θ);
3   | return p
```

Fig. 2.5 Improve probability procedure.

Let $\hat{z}$ be the best solution value found so far. It is possible to compute an approximation of the probability of improving the incumbent solution by

$$p = 1 - \int_0^{\max(\mathcal{Z})-\hat{z}} f_{\mathcal{X}}(t)\ dt. \tag{2.10}$$

The result of the procedures `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 positive 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 2.5) 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`.

2.4 Computational Testing I

Our GRASP has been implemented in C++ and compiled with `gcc5.4.0` 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 [34], `bsolo` solver kindly provided by its authors [101], the `MiniSat+` [46] 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` [101, 102] 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 [104].

`MiniSat+` [46, 142] is a minimalistic implementation of a Chaff-like SAT solver based on the two-literal watch scheme for fast Boolean constraint propagation [117], and conflict clauses driven learning [104]. In fact the `MiniSat` solver provides a mechanism which allows to minimize the clauses conflicts. `PWBO` [110, 112, 111] 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.

In our testing, we have initially considered the datasets used to test feature selection methods in [11], 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 2.1 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 with 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 2.2 and in Figure 2.7. 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.

The experimental evaluation of the probabilistic stop is summarized in the three distinct boxplots of Figure 2.6. 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 described above. The first boxplot depicts

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

Table 2.1 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 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

Table 2.2 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

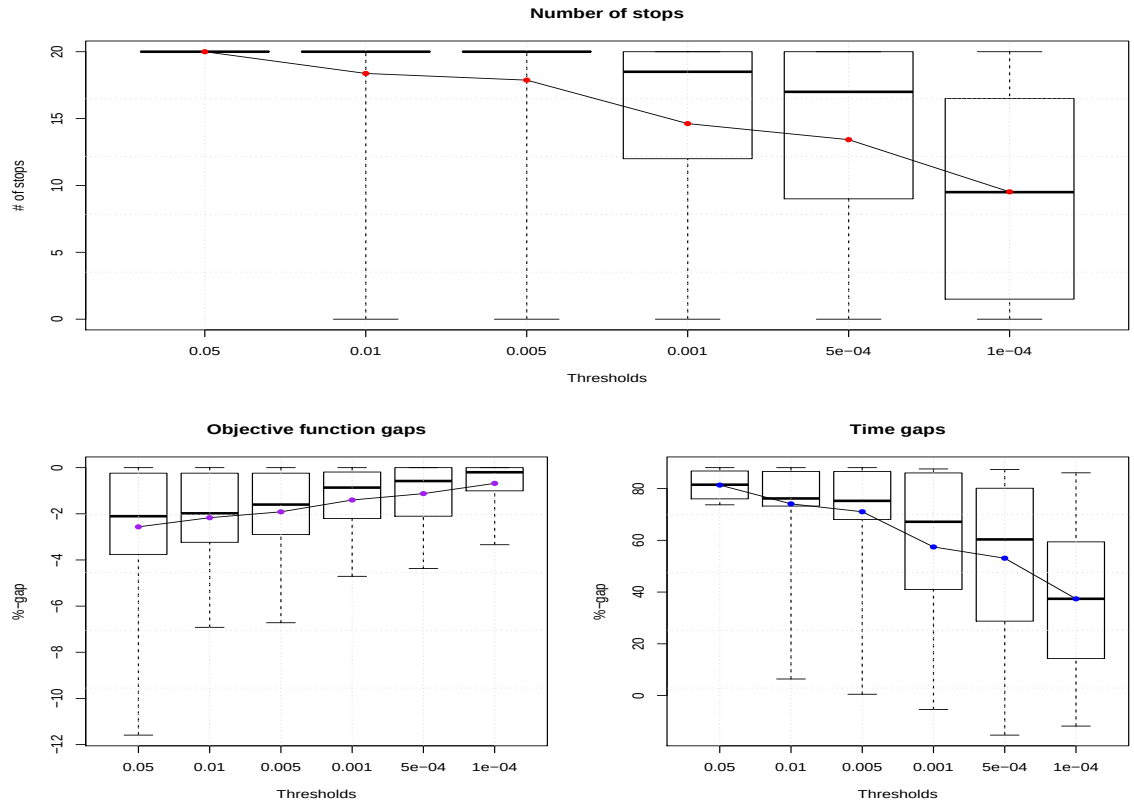

Fig. 2.6 Experimental evaluation of the probabilistic stopping rule. In each boxplot, the boxes represent the first and the second quartile; solid line represent median while dotted vertical line is the full variation range. Plots vary for each threshold α. The dots connected by a line represent the mean values.

-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.

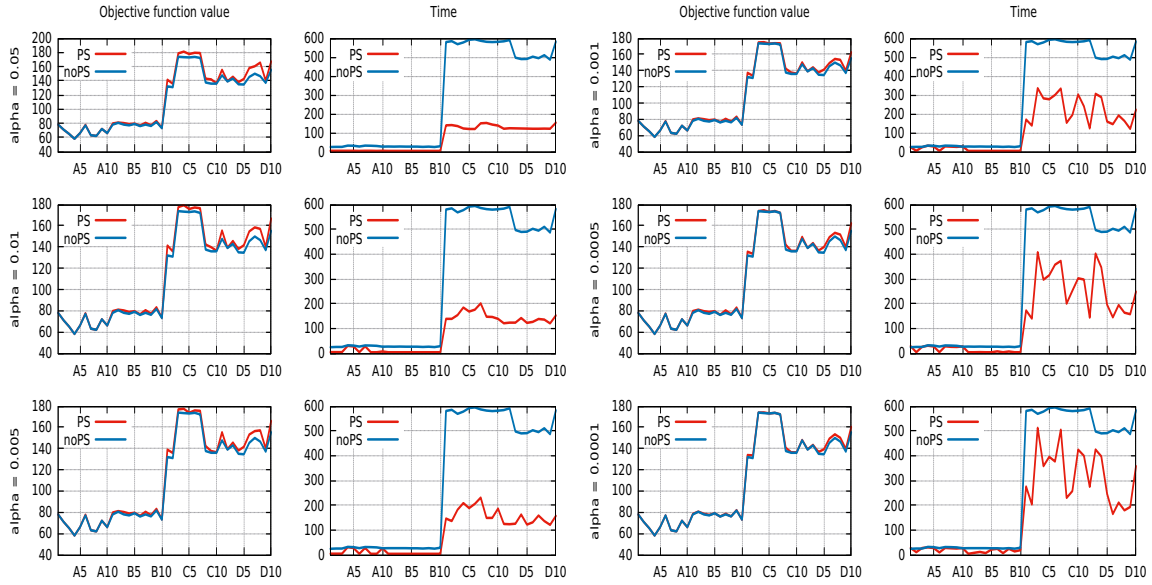

Fig. 2.7 Comparison of objective function values and computation times obtained with and without probabilistic stopping rule for different threshold values.

2.5 A Hybrid Metaheuristic Algorithm for the Max Cut-Clique Problem

The study of Cliques in graphs is a classical field of interest in combinatorial optimization literature. This family of problems –whose most famous member is the Maximum Clique Problem (MCP)– finds a wide variety of applications, ranging from bioinformatics [24] to computer vision [17].

This part of the thesis is concerned with the study of a recent variation of the classical MCP, called the Max Cut-Clique Problem (MCCP), firstly introduced in [108]. In this case, rather than the construction of a complete subgraph of maximal cardinality of a certain graph G, the focus is the extraction of a complete subgraph with maximal cut. If in the original MCP the attention was on connections within nodes of huge subsets, the focus of the MCCP is the influence that tightly connected node-subsets have on the rest of the graph. This paradigm is particularly interesting in the field of Market Basket Analysis. In fact, as argued in [129, 109], the study of cliques in customer purchase networks may

yield substantial breakthroughs in the extraction of products association rules, and business analytics in general.

The formal statement of the problem is given as follows. Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. Let C be a clique of G. The cut-clique of C, $E'(C)$, is the set of all the edges in the cut $(C, V \setminus C)$, i.e.,

$$E'(C) = \{(i, j) \in E \mid i \in C \text{ and } j \in V \setminus C\}. \tag{2.11}$$

Since by definition every clique induces a complete sub-graph, denoting by Γ the set of all the cliques of G, the Max Cut-Clique problem (MCCP) can be stated as follows:

$$\text{(MCCP)} \quad z = \max_{C \in \Gamma} \left\{ \left(\sum_{i \in C} |E'(i)| \right) - |C|(|C| - 1) \right\}.$$

Even if the MCCP is a variation of a widely studied problem of Combinatorial Optimization, it has yet to be extensively investigated. Indeed, the only effort to solve this problem is presented in [109], where the authors discuss several adaptations of the classical Iterated Local Search (ILS) Framework. The algorithms devised in [109] rely on two main operations to build cliques: `add(·)` and `swap(·)`. The former move is used to build a solution incrementally, selecting one node at a time. While `swap(·)`, on the other hand, attempts to reach better cliques swapping a node v currently in the clique C with a node $u \in V \setminus C$.

Moreover, two main strategies are developed according to the selection criteria in the `add(·)` and `swap(·)` operations: *Random*-ILS (R-ILS) and *Degree*-ILS (D-ILS), making use, respectively, of a random or a greedy node selection scheme.

In order to tackle the MCCP, we devised a meta-heuristic algorithm which, within a GRASP framework, hybridizes a greedy randomized adaptive construction phase with a Phased Local Search (PLS).

The GRASP construction phase tailored for the MCCP puts together the clique C one vertex at a time. As in the classical paradigm, such vertices are selected randomly from a *restricted candidate list* (RCL), which contains the best ones according to a greedy function. In our implementation, the RCL is made up of the vertices 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 $u \in V \setminus C$ such that $u \in \mathcal{N}(v)$, $\forall v \in C$. Where $\mathcal{N}(v)$ is the set of nodes adjacent to v.

Given the analogies with MCP, we decided to use as local search in our GRASP a suitably adapted version of the PLS proposed in [128], and depicted in Figure 2.8. Let $K_0(C)$ be the set of vertices of V connected to all the elements of the clique C and $K_1(C)$

```
1 Function PLS (C, max_sel)
2 |   C* ← C ;
3 |   for i = 1 to max_sel do
4 |   |   C ← Random_selection(n_iter_1) ;
5 |   |   C ← Degree_selection(n_iter_2) ;
6 |   |   C ← Penalty_selection(n_iter_3) ;
7 |   update_solution(C, C*) ;
8 |   return C*
```

Fig. 2.8 The phased local search procedure used in our GRASP.

be the *missing one* set, made up of vertices v connected to all but one elements of C, i.e,

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

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 vertex of $K_0(C)$ or swapping a vertex $v \in K_1(C)$ 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 vertices.

Moreover, as in [109], we implemented two different versions of our algorithm: GRASP-PLS and GRASP-PLS$_p$, where the latter implementation exploits the following Proposition.

Proposition 1. *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.*

This result allows one 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.

Table 2.3 Average of the solution values (avg-z) and execution time (avg-t) of the algorithms. The p subscript specifies when the implementation uses proposition 1.

	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

2.6 Computational Testing II

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 [109]. For the sake of scientific fairness, for our testing we implemented in C++ the algorithms of [109] -originally implemented in FORTRAN-. Each algorithm has been executed for 100 runs. The results obtained, reported in Table 2.3, show how our hybrid metaheuristic 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.

Chapter 3

Graph Drawing: the Art of Representing Data

As growing chunks of insightful informations are extracted from a dataset, the need for a representation scheme grows. The resulting visualization, which involves a controlled reduction of complexity, is a fundamental part of the learning process and human understanding as we know it.

As Alberto Cairo beautifully put it in his book *The Functional Art*[1]: "Go ahead and explain a difficult concept to a friend. In the moment she gets what you mean she will exclaim, with a sparkle of relief and happiness in her eyes: 'I see!' Her expression makes complete sense, because deep inside our minds, to see and to understand are intertwined processes. We understand because we see."

It is in this spirit that Graph Drawing aims at automatically producing network representations in a way that can enable understanding and ease of communication.

In this chapter we present two different Graph Drawing Problems (GDPs): The Min-Max GDP, and the Constrained-Incremental GDP (C-IGDP). The first problem, differently from classical crossing reduction GDPs, considers a local objective that can be a useful measure both in the case of interactive drawing softwares, and in case of VLSI circuit design. On the other hand, the C-IGDP addresses the solution of a drawing problem while introducing the matter of optimization on dynamic networks, that is further addressed in Chapter 4.

[1]Cairo, A. (2012). The Functional Art: An introduction to information graphics and visualization. New Riders.

3.1 A Local Objective: the Min-Max Graph Drawing Problem

Crossing minimization is a well-known problem in graph drawing [6], with many optimization algorithms proposed over the last twenty years to obtain good solutions. The first drawing problem that we consider is a recent variant proposed by Stallmann [143], where the goal consists in the minimization of the maximum number of crossings across all edges.

A hierarchical graph $H = (G, k, L)$ is a graph $G = (V, E)$, where the set of vertices V and the set of edges E are partitioned into k—layers in a way that:

$$V = \bigcup_{l=1,\ldots,k} V^l; \tag{3.1}$$

$$E = \bigcup_{l=1,\ldots,k-1} E^l \subset \bigcup_{1\leq l<k} V^l \times V^{l+1}, \tag{3.2}$$

where all edges connect vertices on consecutive layers. Additionally, the function $L : V \to \{1, \ldots, k\}$ indicates the layer where each vertex $v \in V$ resides, implicitly defining $V^l = \{v \in V : L(v) = l\}$. Let n_l be the number of elements in layer l ($n_l = |V^l|$). A drawing of H is defined as $D = (H, \Phi)$, where $\Phi = \{\varphi^1, \varphi^2, \ldots, \varphi^k\}$, and φ^l is the ordering (permutation) of the vertices in layer l. Let $c(u, v)$ be the number of edges that cross edge $e = (u, v) \in E$, called the *crossing number* of e. Given a drawing D, the aim of the *min-max crossing problem* is to minimize its maximum crossing number $C(D)$,

$$C(D) = \max\{c(e) : e \in E\}. \tag{3.3}$$

For the sake of simplicity, we will use C instead of $C(D)$ whenever there is no ambiguity among different drawings.

The min-max crossing problem in a bipartite graph was formulated as a linear integer model (Martí et al. [114]) by adapting the classic formulation for the min-sum problem proposed by Jünger and Mutzel [91]. To formulate the min-max problem for an arbitrary number of layers (HDGA), we define binary variables $x^l_{uu'}$ and $c^l_{uvu'v'}$ such that

$$x^l_{uu'} = \begin{cases} 1, & \text{if } \varphi^l(u) < \varphi^l(u'); \\ 0, & \text{otherwise,} \end{cases}$$

and $c^l_{uvu'v'}$ takes the value if edges (u,v) and (u',v') cross, i.e.,

$$c^l_{uvu'v'} = \begin{cases} 1, & \text{if } (\varphi^l(u) - \varphi^l(u'))(\varphi^{l+1}(v) - \varphi^{l+1}(v')) < 0; \\ 0, & \text{otherwise.} \end{cases}$$

where $u, u' \in V^l$ and $v, v' \in V^{l+1}$ for all layers $l = 1, \ldots, k-1$. The mathematical formulation for the min-max edge crossing problem is as follows:

$$\begin{aligned}
&\min \quad C \\
&\text{s.t.} \\
&\quad -c^l_{uvu'v'} \le x^{l+1}_{vv'} - x^l_{uu'} \le c^l_{uvu'v'} && (u,v),(u',v') \in E^l, u < u', v < v', l < k && (3.4)\\
&\quad 1 - c^l_{uvu'v'} \le x^{l+1}_{v'v} + x^l_{uu'} \le 1 + c^l_{uvu'v'} && (u,v),(u',v') \in E^l, u < u', v > v', l < k && (3.5)\\
&\quad 0 \le x^l_{uv} + x^l_{vw} - x^l_{uw} \le 1 && \forall\ 1 \le u < v < w \le n_l,\ \forall l && (3.6)\\
&\quad c(u,v) = \sum_{\substack{(u',v')\in E^l \\ u<u'}} c^l_{uvu'v'} + \sum_{\substack{(u',v')\in E^l \\ u'<u}} c^l_{u'v'uv} && \forall (u,v) \in E^l && (3.7)\\
&\quad c(u,v) \le C && \forall (u,v) \in E^l && (3.8)\\
&\quad x^l_{uu'} \in \{0,1\} && \forall\ u, u' \in V^l, u < u',\ \forall l && (3.9)\\
&\quad c^l_{uvu'v'} \in \{0,1\} && \forall\ (u,v),(u',v') \in E^l, u < u',\ \forall l. && (3.10)
\end{aligned}$$

Constraints (3.4)-(3.6) are straightforward adaptations of the crossing and ordering constraints appearing in classical GDP formulations [91, 90]. Constraints (3.7) compute the crossing number of all edges, and constraints (3.8), together with the objective function, force variable C to take the maximum of these crossing numbers. Note that the minimization of C in the objective function tries to reduce it as much as possible, thus matching the largest $c(u,v)$-value in constraints (3.8). In this way, it provides the optimal value and the associated orderings of the min-max edge crossing problem, as in [114] for the bilayer case. The output solution is indeed the optimal drawing of the hierarchical graph. Table 3.1 summarizes all the symbols introduced in this section.

Symbol	Definition
G	Graph: $G = (V, E)$
V	Set of vertices of G
E	Set of edges of G
k	Number of layers
H	Hierarchical graph: $H = (V, E, k, L)$
L	Function that indicates the layer in which a vertex resides
V^l	Set of vertices in layer l in H
n_l	Number of vertices in layer l
D	Drawing: $D = (H, \Phi)$
Φ	Set of permutations $\{\varphi^1, \ldots, \varphi^k\}$
$c(e)$	Number of edges that cross edge $e \in E$
$C(D)$	Maximum number of edge crossings of a drawing D

Table 3.1 Symbols and Definitions.

3.2 Solution Approaches

In this section, we review the previous methods published to solve the min-max crossing problem. As mentioned above, the min-max crossing problem has been recently proposed and, in spite of its practical applications, only two works have published to deal with it.

The first previous paper for this problem is due to Stallmann [143], who developed the `MCE` heuristic. It is a very efficient method based on the sifting principle, which performs passes over the layers of the graph, until no further improvement can be found. It is worth mentioning that most of the classic optimization methods for the min-sum problem [6], such as the barycenter or the median procedures, consider the vertices as the main element in their design. The contribution of Stallman's paper consists in the introduction of the problem and a solving method for it. This work came along with a change in paradigm in terms of considering the edge as the main element in the drawing. This novelty comes from computing the number of crossing for a single edge, instead of aggregating them as in the min-sum model. It is well documented, see [99], that min-max and max-min optimization problems constitute a challenge for heuristic algorithms due to the fact that many solutions have the same objective function value, thus making local search based methods relatively inefficient.

`MCE` identifies the edge with the largest number of crossings and tries to reduce it relocating its endpoint vertices in their layer. The method starts with an initial drawing D, determined by the barycenter method. Then, it sorts the edges $e \in E$ in descending order

according to their number of crossings $c(e)$. At each step, following this order, each edge e is examined, and its endpoints are checked in search for their best position.

As pointed out by the author, an interesting feature of the MCE method is that it performs improving moves (vertices relocation) based on a local principle. This means that, instead of using the objective function value $c(D)$ to determine the move, only edges incident to the chosen vertex are considered to find its best position. Thus, MCE determines the best location for a vertex v as the one that minimizes the maximum number of crossings among the edges incident to v. In practical terms, this implies that a drawing could be poorer after a vertex relocation since the number of crossings of a non-adjacent edge could eventually increase. The experiments showed that this local strategy achieves good results, working as a diversification element and preventing the search from being trapped in a local optimum. A post-processing procedure of exchanging adjacent vertices is also applied to further improve the solutions.

Martí et al. [114] proposed an iterated greedy heuristic based on the Strategic Oscillation (SO) methodology [135]. After the construction and improvement of an initial solution, this technique alternates between destructive and constructive phases. The SO algorithm is based on the iteration of these three steps, constructive phase, neighborhood search and destructive phase.

The construction step starts by randomly selecting a vertex v from all the vertices, and placing it in an arbitrary position in its layer. In subsequent construction steps, the next vertex v is randomly chosen from those in the restricted candidate list RCL, that consists in all unassigned vertices with a degree of no less than a percentage of the maximum degree found among unassigned vertices. Then, the selected vertex is placed in the position prescribed by the barycenter, $bc(v)$ in its layer. The barycenter is computed as the arithmetic mean of the positions of the already adjacent vertices to the chosen vertex. The construction phase ends when all the vertices have been positioned.

Important ingredients in this algorithm are the sets of near-critical edges ($NCE(p_{IG})$) and vertices ($NCV(p_{IG})$). These sets depend on a search parameter $p_{IG} \in [0,1]$, such that

$$NCE(p_{IG}) = \{e \in E : c(e) \geq p_{IG}\, C(D)\}.$$

On the other hand, the set of near-critical vertices is defined as the set of vertices that are endpoints of near-critical edges.

Once a solution D has been constructed, the neighborhood search is applied. This procedure attempts to move each vertex v in $NCV(p_{IG})$ in positions: $bc(v)-2$, $bc(v)-1$, $bc(v)$, $bc(v)+1$, $bc(v)+2$, if they exist. The vertex is placed in the position that produces the minimum of the total (sum) number of crossings. In order to save computational time,

the algorithm does not recompute the crossings and near-critical sets after every move, until the neighborhood of each vertex in $\mathrm{NCV}(\mathrm{p}_{\mathtt{IG}})$ is explored. This neighborhood search phase ends when all vertices have been considered for insertions.

During the destructive phase, all the vertices in the set $\mathrm{NCV}(\mathrm{p}_{\mathtt{IG}})$ are removed from drawing D. In order to achieve further diversification of the solution, an additional number of vertices randomly selected from the set $V \backslash \mathrm{NCV}(\mathrm{p}_{\mathtt{IG}})$ is removed. A new iteration in the SO algorithm starts by reconstructing the partial solution obtained after the destructive phase. At the end of the execution, the algorithm returns the best solution found in the entire search process.

The next section describes the Tabu Search algorithm that we devised to solve the Min-Max GDP.

3.3 How to See in the Dark: Evaluating Moves a Min-Max Problem

Tabu Search (TS) is a well-known metaheuristic for solving combinatorial optimization problems, and is designed to guide a search method to escape from local optimality. This methodology was first proposed by Glover [70], and precise descriptions appeared in many studies, as for example [71, 72, 74].

TS is based on the premise that efficient heuristic solution strategies should use the information collected during the search process. In particular, it achieves this by incorporating flexible memory structures. This feature allows the algorithm to explore the solution space in an efficient way, effectively solving many combinatorial problems.

The TS framework is built from two main principles: intensification, in which the search tries to achieve a local optimum, and diversification, applied to escape from the actual basin of attraction to find a global optimum. These two strategies let the algorithm explore the search space in different ways. In general, in the intensification phase the movements are guided by the objective function, to improve the current solution. Meanwhile, in the diversification phase, movements are evaluated using both the objective function and frequency functions recording past information, to guide the search towards unexplored regions. In this algorithm, we implement these two concepts by incorporating memory functions of different time spans, namely short and long term.

As already pointed out, in min-max problems, given a solution, most of its neighbors have the same objective function value, thus making the local search almost blind in terms of approaching to a local optima. To overcome this difficulty, we consider in the intensification

phase of our tabu search method, the δ function, proposed by Rodriguez-Tello et al. [133] in the context of the Bandwidth Minimization problem (BMP). The idea that motivates us to use this evaluation function lies in the similarities that both problems share: (i) a min-max objective function, (ii) the correspondence between a solution and a permutation of the vertices, and (iii) trivial feasibility.

In the cases of Min-Max GDP and BMP ([133]), the cardinality of the solution space presents a factorial growth in the size of the instance input. At the same time, in both problems, the objective functions can be bounded from above by a polynomial in the size of the input, linear for the BMP and quadratic for our problem. For example, in the case of the *min-max crossing* problem, a trivial bound can be obtained by counting the maximum number of edges among two consecutive layers minus one. Practically speaking, being the upper bounds low with respect to the cardinality of the search spaces, in both cases there is a plethora of solutions sharing the same cost. This means that, during the local search, a move evaluation function solely based on the *min-max* objective hardly detects improving moves, resulting eventually myopic.

The δ-evaluation function for a Drawing D, depending from its ordering Φ, is defined in equation (3.11), where d_x is the number of edges with x crossings, UB an upper bound, and β defines the current maximum crossing.

$$\delta(D) = \beta + \sum_{x=1}^{\beta} \frac{d_x}{\frac{(UB+\beta-x+1)}{UB!}}. \tag{3.11}$$

Let us consider the graph in Figure 3.1 to illustrate the rationale behind the δ-evaluation function (3.11). This drawing presents two edges with 3 crossings, which happens to be the maximum: the edge that connects vertex 0 of the first layer and vertex 3 of the second, and the edge connecting vertex 2 of the second layer with vertex 0 of the third layer. The histogram on the right hand side of Figure 3.1 shows the number of crossings (cost) distribution in the drawing, collecting for each possible cost value the number of edges with that number of crossings. For this specific graph representation the objective function (maximum number of crossings) is 3 and the δ function is 3.43959.

Swapping vertices 1 and 0 in the first and in the last layer of Figure 3.1, we achieve the drawing depicted in Figure 3.2. As the previous one, this hierarchical representation presents a maximum crossing value of 3, corresponding to the edge connecting vertex 3 of the first layer with vertex 0 of the second. The objective function in this drawing is therefore 3, however δ function attains a value of 3.20481.

Regardless of the max-crossing, which gets the same value for both drawings, the representation provided by Figure 3.2 is unambiguously clearer than the one of Figure 3.1.

This feature can be deduced by means of a comparison of the two cost distributions, since the number of edges with a relatively high cost decreased. In this way, we can consider that in the neighborhood of the new drawing it is more likely to find an improving solution. This important characteristic for a local search method is properly detected by the decrease of the δ function, but overlooked by the plain evaluation of the min-max objective function.

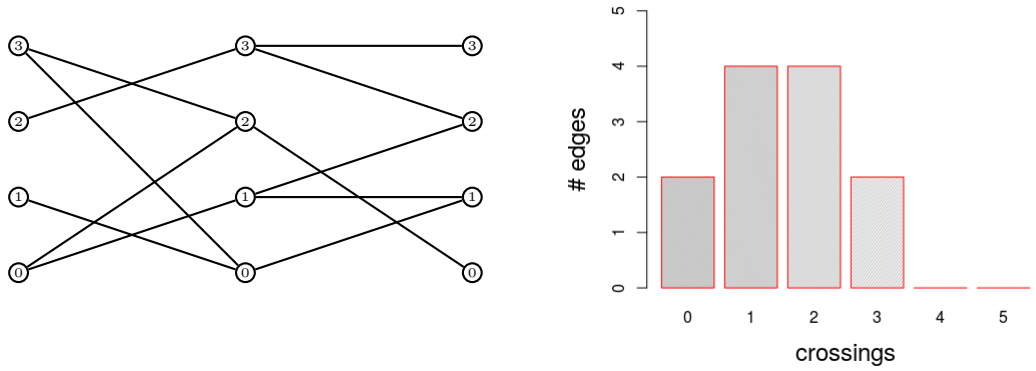

Fig. 3.1 δ-evaluation function example.

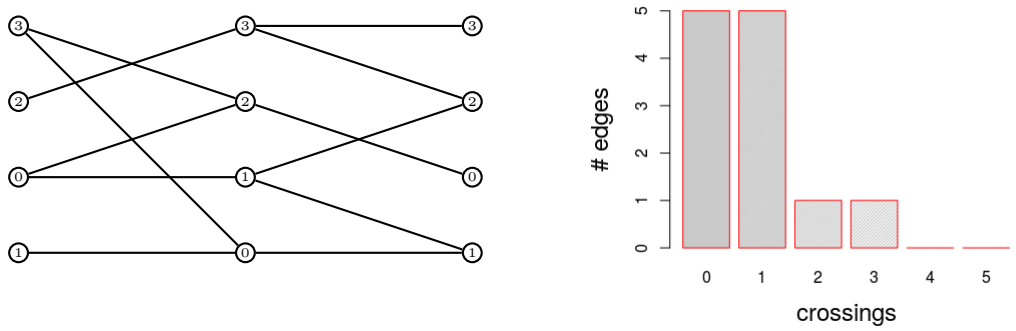

Fig. 3.2 δ-evaluation function example after swap.

Our two-phase TS method is outlined in Figure 3.3. The first phase, Short-Term TS, is focused on an intensification strategy. The second phase, Long-Term TS, complements the first one by performing search diversification to explore new regions of the solution space. Each phase is executed for a maximum number of iterations without improving the current solution, `MaxInt` and `MaxDiv` respectively. The overall TS method stops when the `TimeLimit` is reached.

```
   Data: HDAG D, α, TimeLimit, MaxInt, MaxDiv
   Result: Optimized design for the min-max crossing.
 1 bestSolution = D;
 2 noImprov = 0 ;
 3 while time spent < TimeLimit do
 4     if noImprov < MaxInt then
 5         IntensificationPhase(D,α);
 6         if the current best is improved then
 7             noImprov = 0 ;
 8         else
 9             noImprov++ ;
10     else
11         Diversification(S, MaxDiv);
12         noImprov = 0 ;
13 return bestSolution;
```

Fig. 3.3 Main Tabu algorithm.

Intensification Phase

The exploration in the intensification phase is based on swapping non-tabu vertices in the same layer. In particular, for each layer l, from $l = 1$ to $l = k$, we compute a list of candidate elements to be swapped (CL^l). This list collects all the vertices in the layer with an edge with number of crossing larger than or equal a threshold, computed as a percentage (α) of a reference maximum cost c^l_{max}. The c^l_{max} value is the current maximum number of crossings in edges incident with a non-tabu vertex in l:

$$CL^l = \{u \in V^l : c_u \geq \alpha\, c^l_{max}\}, \tag{3.12}$$

with c_u denoting the maximum number of crossings found in edges incident to vertex u,

$$c_u = \max\{c(e) : e = (u, v) \in E^l\},$$

and

$$c^l_{max} = \max\{c_u : u \in V^l \text{ non-tabu}\}.$$

Starting from a drawing D, for each $u \in CL^l$, the method searches for its best swap with another non-tabu vertex, and performs it whenever the move improves (i.e. decreases) the δ-value of the current solution D. If for all the vertices no improving solution has been found, the algorithm performs the least worsening swap. The swapped vertices become tabu-active and remain in such a status for a specified number of iterations controlled by

```
   Data: HDAG D, α
   Result: Locally optimal drawing.
 1 for l = 1, ..., k do
 2     build CL^l;
 3     for u ∈ CL^l do
 4         v ← best_swap_vertex(u);
 5         if swap(u,v) decreases δ then
 6             update solution;
 7             update δ;
 8             declare tabu u, v;
 9     if no swaps performed then
10         perform least worsening;
11         update tabu statuses;
12 return;
```

Fig. 3.4 Pseudo-code of the Intensification Phase.

the search parameter $tenure$. Swaps that involve tabu-active vertices are declared tabu and therefore are not allowed. The intensification phase stops after `MaxInt` iterations without improvement. The structure of our intensification phase is illustrated in Figure 3.4.

Diversification Phase

In some `TS` applications, short term memory strategies produce very high quality solutions by themselves. However, in general, `TS` becomes significantly stronger by including longer term memory, implementing diversification strategies that allow the search process to escape from the current basin of attraction.

As in classical implementations, our diversification procedure is executed when the intensification reaches the maximum number of iterations without improving the best-found solution, i.e., when we consider that the search is trapped in a specific region of the solution space. In order to move further away from the current local minimum, during the execution of the algorithm, the procedure records the number of times that each vertex has been moved. Then, two vertices are randomly selected for a swap according to a probability distribution defined by the frequency count of the moves that changed their position, where the lower the move frequency the larger the probability of swapping. More specifically, let M^l be the total number of swaps performed in layer l, and let m_u^l indicate the number of swaps that involved vertex u of layer l. Being the swap a pairwise move, we have $\sum_u m_u^l = 2\,M^l$. Then we define the probability of being selected for a swap in the diversification phase (pr_u^l) as:

```
   Data: HDAG D, MaxDiv
 1 step = 0 ;
 2 while step < MaxDiv do
 3 |   for l = 0, ..., k do
 4 |   |   for u ∈ V^l do
 5 |   |   |   compute pr^l_u ;
 6 |   |   u ←random_node(pr^l);
 7 |   |   v ←random_node(pr^l);
 8 |   |   swap(u, v) in V^l ;
 9 |   |   update swap frequencies;
10 |   |   update tabu statuses;
11 |   |   if the δ decreases then
12 |   |   |   return;
13 |   step++ ;
14 return;
```

Fig. 3.5 Pseudo-code of the Diversification Phase.

$$pr_u^l = \frac{M^l - m_u^l}{\sum_v (M^l - m_v^l)}, \quad \forall u \in V^l. \tag{3.13}$$

The pseudo-code of the diversification phase is shown in Figure 3.5. This phase is executed sweeping through the layers of the graph for `MaxDiv` iterations in each layer if no improving solution is encountered. If the algorithm incurs in an improving solution we force an early termination of this phase. In any case the `TS` method restarts with the intensification phase in order to obtain a locally optimal solution in the new area under exploration.

Min-Sum post-processing

As discussed previously, the optimization of the min-max crossing function can be found in different real world scenarios, such as the VLSI circuit design or the development of interactive graph drawing tools. As we pointed out in the introduction, the use of the min-max objective function in comparison with the min-sum, reasonably guarantees that edges with a high number of crossing are unlikely to be found in the layout. This feature is particularly valuable when the algorithm is embedded in a drawing tool that allows to zoom onto user-specified areas of the graph.

One of the main characteristics of the min-max problems is that we can find many solutions with the same objective function value, so we can expect that it has many

alternative optima. Therefore, it is possible to consider a post-processing procedure to reduce the total (sum) number of crossings, without increasing the min-max value. In this way, we are improving the max-sum as a secondary objective function without deteriorating the primary objective min-max function. Note that this approach is in line with Stallmann's [143] and with the classical drawing approaches that make use of the min-sum as reference function.

Our post-processing consists of a local search method guided by the min-sum function with a swap-based neighborhood structure. Note that the TS method described in the previous subsections is guided by the min-max objective, and more specifically by the δ function. This post-processing only considers moves that do not worsen the solution in terms of min-max, thus preserving solution quality, while attaining better min-sum values. In particular, the method sequentially sweeps layers (from 1 to k), and scans their vertices in search for an improving swap with respect to the sum of crossings. The procedure stops after the exploration of all the layers (and all the vertices in each one) when no improving move is performed.

3.4 Computational Experiments I

In this section, we perform extensive computational experiments to analyze the key elements in our proposed method (TS) and to compare it with previous methods. In particular, we compare TS with the maximum crossing edge heuristic method (MCE) by Stallmann [143] and the strategic oscillation method (SO) proposed by Martí et al. [114]. MCE and SO are implemented in C, and TS in C++, and the experiments are conducted on a computer with a 2.6 GHz Intel Core i7 processor with 8 GB of RAM. The MCE algorithm has been downloaded from https://people.engr.ncsu.edu/mfms. The set of instances we use is available online in the webpage http://www.optsicom.es.

We divide the experimentation into two main parts, preliminary experiments and comparative study. In the first one we set the values of the search parameters and finish with a study of the δ-function contribution in the search process. As mentioned above, in our comparative study we consider CPLEX, and the two previous heuristics. The benchmark set of instances is created with Stallmann's generator [143]. It consists in 149 instances with different number of layers and graph densities. Table 3.2 summarizes their features. Note that CPLEX is only able to target medium size instances, and for the heuristic comparison we consider the 60 instances already used and described in [114]. Additionally, for the δ-function experimentation, we keep the instance size fixed and vary the density to see how it may affect the method performance.

Type	#inst.	Vertices	Layers	Density	Class
49 instances solved with CPLEX					
low1	15	100	10	≈ 1.50	Low
low2	14	200	20	≈ 1.50	Low
low3	5	450	30	≈ 1.70	Low
low4	5	450	30	≈ 2.30	Low
low5	5	450	30	≈ 2.90	Low
low6	5	450	30	≈ 3.60	Low
60 instances run in the heuristic comparison					
noug3	10	300	15	≈ 14.00	High
noug4	10	60	3	≈ 10.00	High
noug5	10	100	5	≈ 12.00	High
noug6	10	500	25	≈ 14.50	High
noug7	10	800	40	≈ 10.00	High
noug8	10	1000	50	≈ 10.00	High
70 instances run in the δ test					
low3	5	450	30	≈ 1.70	Low
low4	5	450	30	≈ 2.30	Low
low5	5	450	30	≈ 2.90	Low
low6	5	450	30	≈ 3.60	Low
med1	5	450	30	≈ 4.30	Medium
med2	5	450	30	≈ 5.00	Medium
med3	5	450	30	≈ 5.80	Medium
med4	5	450	30	≈ 6.50	Medium
med5	5	450	30	≈ 7.20	Medium
high1	5	450	30	≈ 8.00	High
high2	5	450	30	≈ 8.70	High
high3	5	450	30	≈ 9.50	High
high4	5	450	30	≈ 10.15	High
high5	5	450	30	≈ 10.85	High

Table 3.2 Benchmark set.

We evaluate the results of our experiments with the following statistics:

- $\bar{C}$: the average of the max edge-crossing value;
- *Best*: the number of best solutions found;
- *Opt*: the number of optimal solutions found;
- $\%\ dev$: the average percent deviation with respect to the best solution found in the experiment;
- $\%\ gap$: the average percent deviation with respect to the CPLEX solution found in the experiment;
- $Time$: Average total time in seconds to execute the method.

3.4.1 Preliminary Experiments

In this section we first perform a preliminary test to fine tune the parameters of the algorithm. To avoid the over training of the methods, we consider a fraction ($< 24\%$) of the instances (26 graphs) with different number of layers and densities. We call this subset the *training set*, as opposed to the entire set of instances called the *testing set*.

The first parameter considered in our tuning is α, which controls the construction of the vertex candidate list in our intensification phase. We evaluate the results obtained by considering five different values of α: 0, 0.1, 0.5, 0.9 and 1, with the other parameters being fixed to reference values. Table 3.3 shows the mean results over the 26 instances obtained on the training set with a time limit of 60 seconds for each execution. In particular, it shows the average number of crossing ($\overline{C}$), the average percentage deviation ($\%\ dev$), the number of instances for which the method is able to match the best solution found ($Best$), and the CPU time. It is clear from these results that the best setup is obtained with $\alpha = 1$, in terms of number of best solution found, and in terms of both average max-crossings and deviations.

In our second preliminary experiment, we test the short term TS method described in Section 3.3 with several *tenure* values. The competing configuration tested are $tenure = 2$, 3, 5 and 10. The results in Table 3.4 show that there is not a clear winner. Small values of the tenure seem more performing, with $tenure = 3$ (highlited in bold in the table) having a slight edge on the others, being this our selected configuration.

In the last tuning experiment, we compare different configurations for the parameters MaxInt and MaxDiv, used to measure, respectively, the maximum lengths of the intensification and diversification phases. To achieve the right balance between intensification

training set, 26 instances				
α	$\overline{C}$	% dev	Best	Time
0	143.69	4.20	7	63.20
0.1	143.77	5.43	6	62.50
0.5	143.62	4.30	7	62.53
0.9	141.04	3.61	9	61.30
1	**139.96**	**2.21**	**18**	60.75

Table 3.3 Fine-Tune parameter α for the TS procedure.

training set, 26 instances				
tenure	$\overline{C}$	% dev	Best	Time
2	139.58	2.41	14	61.04
3	**139.42**	**1.68**	**15**	60.86
5	139.77	4.89	10	60.83
10	142.31	17.93	2	60.63

Table 3.4 Fine-Tune parameter *tenure* for the TS procedure.

and diversification, we test the parameters coupled in pairs (MaxInt, MaxDiv), generating four different configurations: $(10, 3)$, $(20, 10)$, $(30, 15)$, and $(50, 20)$. Being the training set extremely diverse in terms of density and size of the graph, as can be observed in Table 3.2 in this experiment we divide the training instances into two classes: low-density graphs and high density graphs.

MaxInt	MaxDiv	$\overline{C}$	% dev	Best	Time
10	3	15.70	6.96	3	60.14
20	10	15.30	3.11	6	60.27
30	15	15.50	7.39	5	60.27
50	20	**15.20**	**1.11**	**7**	60.20

Table 3.5 TS fine-tune on 10 low density instances.

As expected, the differences found in the training set are reflected in tuning results shown in Table 3.5 (low density) and Table 3.6 (high density). In particular, the best parameters combination seems to be $(50, 20)$ in the case of low density graphs, and $(10, 3)$ for high density graphs. In smaller graphs, characterized by low density, the algorithm is able to perform more intensification steps within the same time limits. Moreover, when the density is low, even with an high number of diversification steps, random swaps can

MaxInt	MaxDiv	$\overline{C}$	% dev	Best	Time
10	3	**216.75**	**0.66**	**9**	61.53
20	10	218.06	1.18	6	62.50
30	15	217.44	0.74	7	61.18
50	20	219.31	1.73	6	61.20

Table 3.6 TS fine-tune on 16 high density instances.

strongly diversify the solution but at the same time worsening its quality in a way that can still be repaired in the next intensification phase. On the other hand, on high density graph, a high number of random swaps could generate a worsening in the solution quality that the intensification phase can not significantly improve.

We conclude our preliminary experimentation with an empirical study on the contribution of the δ-evaluation function in the search efficiency. The tests are designed to run the TS with two different move evaluation functions in the intensification phase: the δ-function, and the plain min-max objective function. More specifically, with the former evaluation, the algorithm performs a move from solution D to the neighboring drawing $\bar{D}$ if $\delta(\bar{D}) < \delta(D)$. On the other hand, in the latter move evaluation setup, intensification moves are performed only when the objective function value decreases. We study in this experiment the percentage of the relative difference of the two solution values:

$$\%\ rel = \frac{C(D_{mM}) - C(D_{\delta})}{C(D_{\delta})} \cdot 100, \tag{3.14}$$

where D_{δ} and D_{mM} are the optimized drawings obtained respectively by using the δ function and the plain min-max objective as move evaluation functions.

As discussed in Section 3.3, the evaluation of neighboring solutions is particularly hard in large sparse graphs, so we perform our comparison with respect to the density of the graph. Henceforth, to properly relate the performances of the two setups with network density, in this experimental phase we generate 70 instances with increasing density and fixed size, both in terms of total vertices and layers. For a summary of the network features used in these experiments see Table 3.2. The algorithm is executed in both setups for a total time of 60 seconds and using the same parameter configuration for both cases.

As expected, the results in Figure 3.6-left evidence that the δ-evaluation function is of critical importance for drawing graphs with low density. Moreover, we can remarkably observe how the percentage ratio between the two solutions is always greater than zero, meaning that in all instances the use of δ as evaluation function is always favorable. In

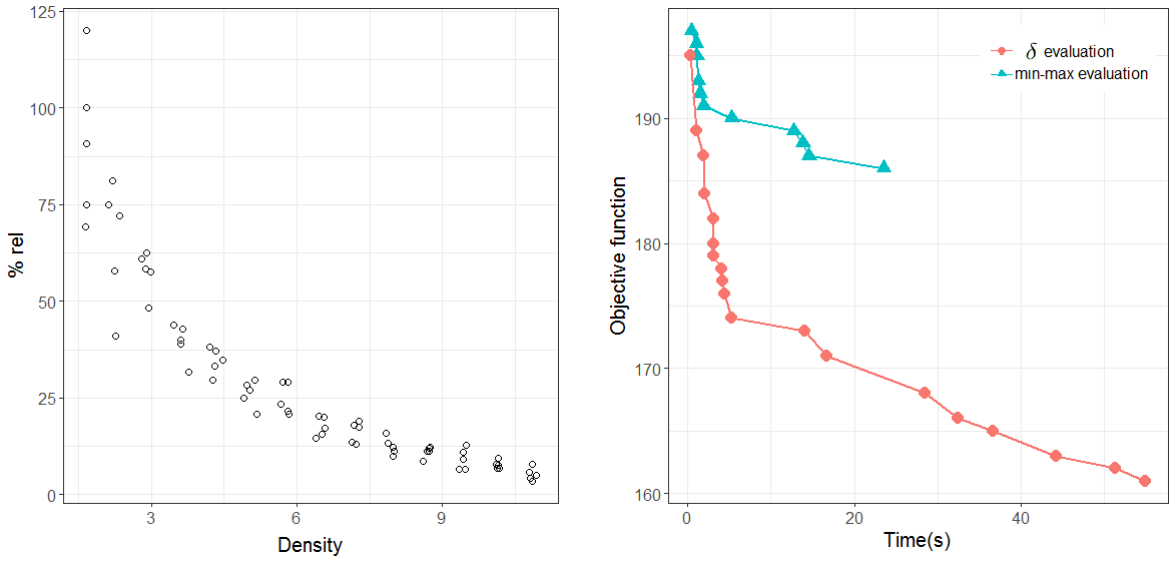

Fig. 3.6 Comparison of δ and min-max objective functions.

addition, Figure 3.6-right depicts an example of the two search profiles obtained for an instance in our benchmark set. We can note how the use of δ as evaluation function lets the algorithm improve when the plain min-max stalls, as well as allowing the procedure to reach earlier good quality solution, as argued in Section 3.3.

3.4.2 Comparative Testing

In this section, we first compare our TS method with MCE heuristic and the solutions obtained with CPLEX (v 12.8) with the linear integer formulation. The two heuristic algorithms are executed with a time limit of 60 seconds, and we run CPLEX for 1 hour. We consider the 49 instances of small-to-medium size and low density in the testing set. For a better analysis of the results, the graphs considered in this testing are divided into a sequence of six different groups: from low1 to low6, growing both in size and density.

As can be seen in Table 3.7, CPLEX is able to optimally solve almost all the smallest instances belonging to the class low1, and half of the instances in class low2. In both of those classes, even when is not able to reach an optimal solution, CPLEX is able to obtain the best solution out of the three algorithms. On the other hand, the two heuristics can reach sub-optimal solution in the very short running time of 60 seconds considered in this experiment (which is in line with graph drawing applications). When we move to larger graphs with increasing density, then CPLEX is not able to obtain competitive results while the heuristics are still obtaining good results. In particular, TS clearly outperforms

MCE in low4, low5, and low6 instances with lower percentage deviations from the best known solution. On small density graphs, MCE performs slightly better than TS. It is worth mentioning that MCE and TS are able to obtain 4 and 3 optimal solutions respectively in the low1 group.

	Instance Class					
	low1	low2	low3	low4	low5	low6
CPLEX						
$\overline{C}$	4.33	5.64	26	42.20	53.40	64.60
Time	449.04	2487.51	3600	3600	3600	3600
Opt	14/15	7/14	0/5	0/5	0/5	0/5
Best	15/15	14/14	0/5	0/5	0/5	0/5
% dev	0.00	0.00	191.44	144.04	105.31	87.90
MCE						
$\overline{C}$	5.20	6.36	9.20	18.80	29.80	41
Best	4/15	4/14	5/5	1/5	0/5	0/5
% gap	21.44	13.06	-60.67	-55.32	-44.22	-36.44
% dev	21.44	13.06	0.00	8.61	14.53	19.30
TS						
$\overline{C}$	5.27	6.50	11.20	17.80	26	34.40
Best	3/15	3/14	0/5	4/5	5/5	5/5
% gap	22.22	16.55	-52.40	-57.99	-51.27	-46.66
% dev	22.22	16.55	22.25	2.00	0.00	0.00

Table 3.7 Comparison of TS, MCE, and CPLEX on 49 instances (low density).

In our second experiment in this section, we undertake to evaluate the performance of our TS with respect to the two previous heuristics, MCE and SO. We consider 60 instances of mixed size and high density of the *testing set* (subsets nough3 to nough8), so according to our previous results, we cannot include CPLEX in this experiment. The two previous heuristics are run for 60 seconds in noug3 to noug6, and 300 seconds on the harder noug7 and noug8, following what is reported in [114]. Our TS is run for 60 seconds in all the cases (i.e., in each instance of the *testing set*). Table 3.8 reports the solutions of this round of experiments, summed up in Table 3.9. For the three algorithms we report the average number of crossings ($\overline{C}$), the average total time, the number of best solutions found, and the average percentage deviation from the best solution found.

Analyzing the computational results, we can see how TS obtains the best solution in 57 out of the 60 instances, and shows the minimum average number of crossings among the three procedures. These remarkable performance is reflected in the average percent deviation, which amounts to zero percent in 5 out of 6 instance types, and always lower than 0.60 percent. In particular, we can see in Table 3.9 how the average deviation over the whole *testing set* is 0.01 percent, compared to 4.77 and 5.76, respectively for MCE and SO. Moreover, we observe how the TS achieves high quality solutions in a short computational time even in the case of graphs of large size (such as in instance classes noug7 and noug8).

	Instance Class					
	noug3	noug4	noug5	noug6	noug7	noug8
MCE						
$\overline{C}$	269.30	246.80	254.00	270.30	175.00	176.00
Time	60.00	60.00	60.00	60.00	300.00	300.00
Best	0/10	2/10	1/10	1/10	0/10	0/10
% dev	3.85	4.31	2.51	3.27	6.69	7.99
SO						
$\overline{C}$	267.20	241.80	254.70	273.30	180.70	182.60
Time	80.60	32.20	68.50	145.80	312.50	476.20
Best	0/10	2/10	1/10	0/10	0/10	0/10
% dev	3.04	2.12	2.79	4.41	10.17	12.05
TS						
$\overline{C}$	259.30	238.20	247.80	261.80	164.10	163.00
Time	61.25	60.53	60.74	62.02	62.02	61.87
Best	10/10	7/10	10/10	10/10	10/10	10/10
% dev	0.00	0.59	0.00	0.00	0.00	0.00

Table 3.8 Comparison of MCE, SO, and TS on 60 instances (high density).

Procedure	$\overline{C}$	% dev	Best	Time
MCE	231.90	4.77	4	140.00
SO	233.38	5.76	3	185.97
TS	**222.37**	**0.01**	**57**	61.40

Table 3.9 Summary of heuristics performance.

We conclude our experimentation by analyzing the effect of the post-processing procedure described in Section 3.3 to reduce the sum of crossings. In particular, we compare the

previous heuristics MCE and SO with our Tabu Search with post-processing (TS-p). Table 3.10 reports the results for the three methods in terms of the average sum of edge-crossings *cross-sum*, its associated average percent deviation, $\%\ dev_s$ with respect to the best crossing-sum value found in the experiment, average of the max edge-crossing value, $\bar{C}$, its associated average percent deviation, $\%\ dev$, and average total time in seconds to execute the method, $Time$.

The analysis of the computational results shown in Table 3.10 evidences how, out of the three algorithms, TS-p always achieves the best max-crossing value, while SO shows the lowest cross-sum over all types of instances. This is expected since SO was meant to simultaneously optimize both objectives, while our tabu search is designed to minimize the maximum number of crossings, and the post-processing reduces the total sum without deteriorating the min.max achieved, which is indeed a hard constraint. Nevertheless, we can observe how TS-p obtains the second best results in terms of average cross-sum, with a maximum percentage deviation of 3% with respect to SO. Moreover, analyzing the global averages of the two percentage deviations, we observe averages of 0.01% and 2.25%, respectively for the deviation from the best $\overline{C}$ and from the best *cross-sum*.

Considering the first heuristic proposed for this problem, MCE, we can observe in Table 3.10 that it presents a relatively good performance considering that it was based on ordering rules and not in metaheuristic methodologies as its competitors in this table. On the other hand, TS-p achieves a good trade-off performance between the two objectives: the main max edge-crossing function, and the secondary edge-crossing sum function. Indeed, it is worth mentioning that while $\%\ dev$ in TS-p is comparable with $\%\ dev_s$ in SO, the value of $\%\ dev$ in SO almost doubles the value of $\%\ dev_s$ achieved by TS-p.

3.5 Drawing Dynamic Informations: Mental Map and Crossing Reduction

It is well documented that incremental drawing is a very important area in graph representations. We can find many references highlighting this problem, or more precisely, this family of problems. We refer the reader to [121], [37], or [126] to mention a few. The graph drawing textbook [6] –a reference in the field– devotes an entire chapter to incremental constructions. The range of applications of incremental techniques is also vast, from on-line problems, such as affiliation networks or on-line advertisement, to the well-known project management diagrams in business administration. However, in spite of its importance and practical significance, there are just a few incremental graph drawing models, and, as

	Instance Class					
	noug3	noug4	noug5	noug6	noug7	noug8
MCE						
cross-sum	276040.8	37764.6	76643.5	475970.2	334284.9	421127.6
% dev_s	10.28	8.55	8.58	10.78	20.05	20.21
$\overline{C}$	269.30	246.80	254.00	270.30	175.00	176.00
% dev	3.85	4.31	2.51	3.27	6.69	8.00
$Time$	60.00	60.00	60.00	60.00	300.00	300.00
SO						
cross-sum	250323.1	34821.3	70594.3	429655.6	278458.1	349641.2
% dev_s	0.00	0.00	0.00	0.00	0.00	0.00
$\overline{C}$	267.20	241.80	254.70	273.30	180.70	182.60
% dev	3.04	2.12	2.79	4.41	10.17	12.05
$Time$	80.60	32.20	68.50	145.80	312.50	476.20
TS-p						
cross-sum	254133.4	35704	72282.8	434861.6	286283.5	260115.7
% dev_s	1.52	2.54	2.39	1.21	2.81	3.00
$\overline{C}$	259.30	238.20	247.80	261.80	164.10	163.00
% dev	0.00	0.59	0.00	0.00	0.00	0.00
$Time$	61.25	60.53	60.74	62.02	62.02	61.87

Table 3.10 Analysis of both objective on 60 high-density instances.

argued in the following, to some extent not entirely satisfactory. In this second part of Chapter 3, we review the incremental graph drawing background including previous efforts in both software and academic context. We discuss the limitations of the existing model for hierarchical drawings and how the proposal here described, and originally presented in [119], overcomes them.

Currently, there exists a wide variety of software devised for graph representation. For example, Graphviz [66] is a free, flexible software, accessible with an easy-to-use web version. These features altogether make Graphviz one of the most popular drawing systems. As the vast majority of its competitors, Graphviz incorporates optimization criteria in order to obtain aesthetically pleasing drawings. We illustrate this point in Figure 3.7, which shows how Graphviz is able to obtain a clear and aesthetically pleasing layout for a simple hierarchical graph.

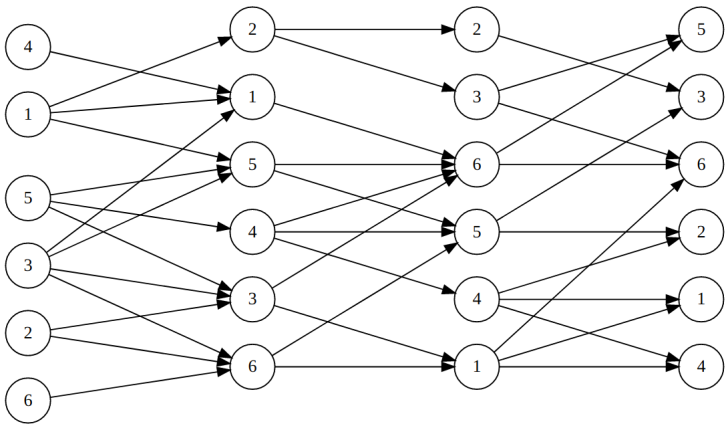

Fig. 3.7 Graphviz drawing of a simple hierarchical graph.

However, Graphviz is not able to properly represent objects and relations characterized by dynamic nature. Indeed, whenever new vertices and edges are added to the network, the software does not support the identification of incremental elements, and it draws the graph from scratch. For example, in Figure 3.8, we can observe how the addition of a new set of vertices and connections forces in the software a completely different representation, thus "destroying" the mental map of the reader.

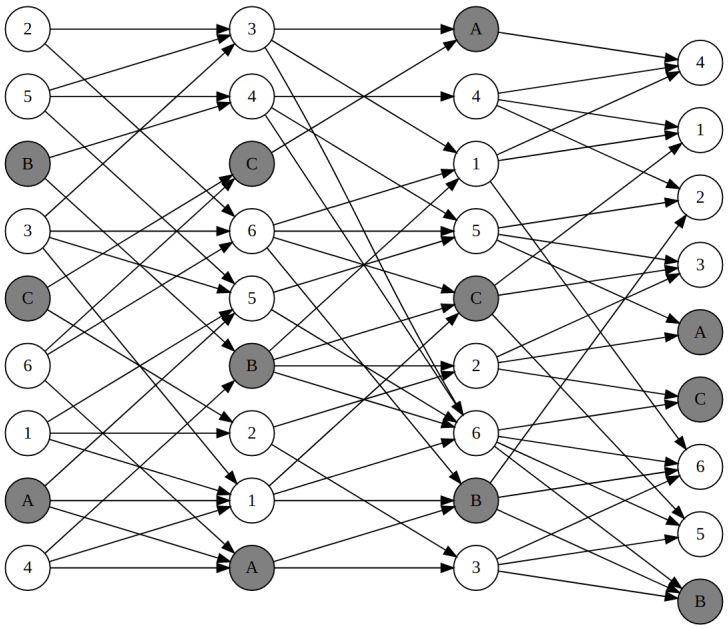

Fig. 3.8 Example optimized with new method.

Recently, in the scientific literature, some efforts were carried out in order to solve incremental graph drawing problems. The heuristic algorithms proposed there, are able to solve large instances, but the resulting drawings present shortcomings in terms of mental map, as can be observed in the following example, arising from *project management*. In these projects, tasks are represented with vertices and edges model their precedence relationships. Many changes occur during the development of a large project and they have to be reflected in the associated graph or chart. Dynamic graph drawing is a demand of project managers who need a stable sequence of drawings as the project evolves. The project is usually represented as hierarchical or layered graph, and constitutes a good example for the applicability of our incremental graph drawing model. Figure 3.9 shows a representation of such a graph on a medium size project. Since it is a large graph with 6 layers, we made some simplifications to draw it. In particular, we do not include the first vertex, which represents the beginning of the project. It would be allocated in the left part (say in layer 0) and connected to all the vertices in the first layer. Similarly, we do not represent the final project's vertex, which would be allocated in layer 7, and connected with an edge to each vertex in layer 6. To reduce the size of the vertices in the drawing, we renumbered them, starting from 1 in each layer. We color in light gray the original vertices and edges, which were in the initial design of the project, and with black the new vertices and edges that have been added in a later stage.

The manager is used to work with the initial project, in which only the light gray vertices were included. In Figure 3.9 the new vertices are all placed in the bottom part of the diagram, leaving in this way the original vertices in their initial position. This is good in terms of the stability of the drawing. In other words, it preserves the manager mental's map of the project. However, it contains a large number of crossings, since no optimization has been performed after the addition of the new vertices. This drawing has 6963 edge crossings. The challenge is therefore to reduce the number of crossings while trying to keep the placement of the original vertices as much as possible. That is esentially the objective of the incremental graph drawing problem.

We have identified two types of approaches in dynamic graph drawing algorithms with the objective of creating a sequence of stable representations, as introduced by Böhringer and Paulisch in [16]. The first type consists of multi-objective methods, which optimize both the aesthetic criteria and a stability distance function. In this category we can find the early work by North in [121], who proposed a graph drawing system, Branke in [18], who adapted Sugiyama's heuristic to include the stability conditions in the drawing method, and [126], who based their stability measure in terms of the number of pairs of vertices that are inverted in the new drawing with respect to their relative position in the previous drawing.

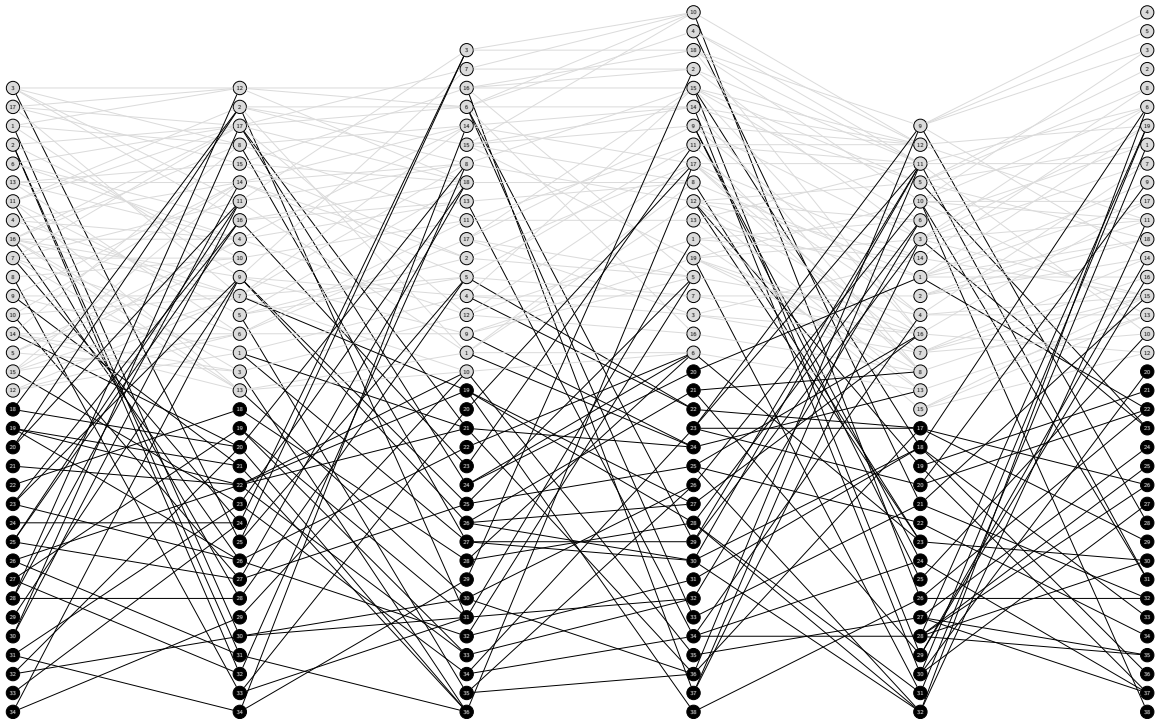

Fig. 3.9 Project management example.

In the second type of approaches, called incremental graph drawing algorithms, we can find the models based on the inclusion of additional constraints in the standard crossing reduction problem. In these approaches, hard constraints about the position of the vertices in the new graph are established beforehand. In particular, Martí and Estruch considered in [105] the relative position between vertices in the original drawing as a constraint to create a new incremental drawing when new vertices and edges are added to the bipartite graph (their GRASP approach was limited to optimize 2-layered graphs). Martí et al. proposed in [106] a Tabu Search for the same problem obtaining better solution than the previous GRASP. Recently, in [136], these authors extended this approach to the general case of multi-layer graphs and proposed a new heuristic based on the Scatter Search methodology, which outperforms the previous GRASP. We apply this method to the example in Figure 3.9, obtaining the drawing shown in Figure 3.10 with 4647 edge crossings.

The methods described above, and applied in Figure 3.10, solve the incremental or dynamic problem based on the relative position of the original vertices. It is easy to check that these vertices, depicted in gray color, keep the same ordering among them in this figure than in the original drawing shown in Figure 3.9. However, we believe that this example also illustrates a serious drawback of this model based on the relative ordering: the location of the original vertices changes significantly, thus endangering the user's mental

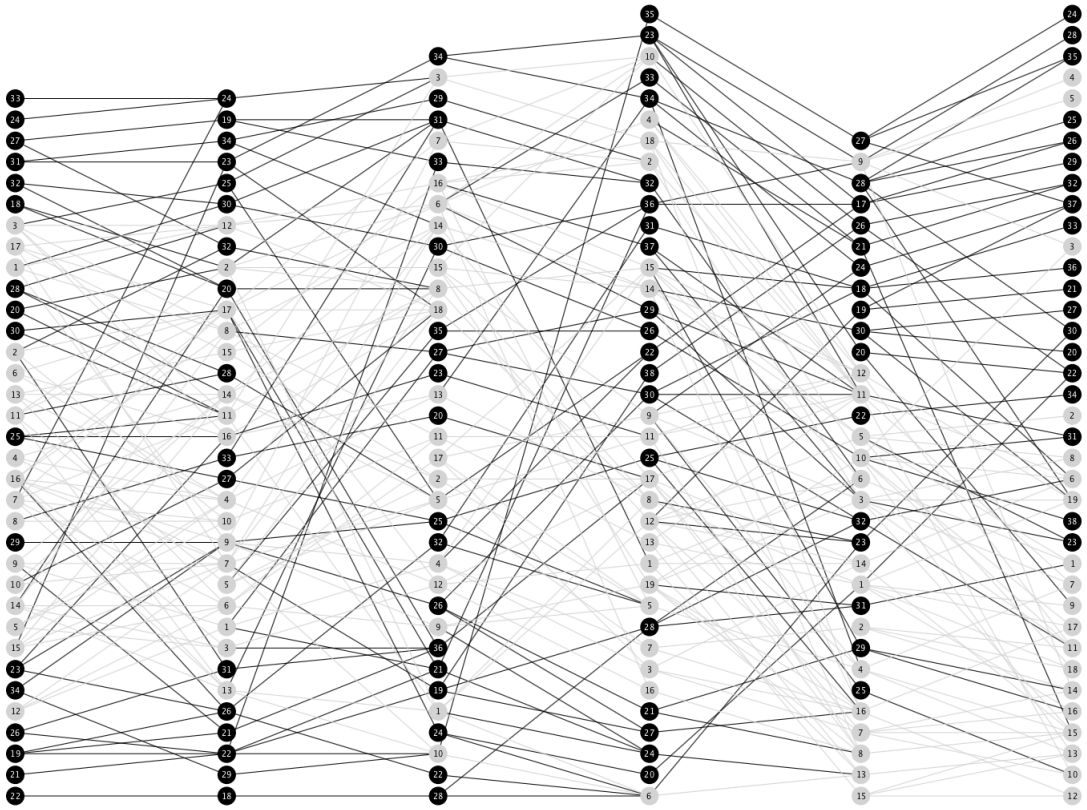

Fig. 3.10 Example optimized with a previous method.

map of the original drawing. For example, in the drawing of Figure 3.9, vertex 3 of the first layer is in the first position, while in Figure 3.10, it is in the sixth position. This is due to five new vertices (33, 24, 27, 31, 32, and 18) inserted in previous positions to reduce the number of crossings. Similar and even worse situations can be easily identified in many original vertices that are now (in Figure 3.10) in positions far from their original placement (in Figure 3.9). If for example we consider the last original vertex in layer 1, number 12, which occupied position 17th in Figure 3.9, we can see that it is now (in Figure 3.10) in position 30. We believe that this difference of 13 positions alters the layout too much and forces the user to make important adjustments in his or her mental map of the graph, thus making the reading of the graph a time consuming task. In Section 3.9, to complement the experimentation, we will show the solution of our method on the example depicted in Figure 3.10.

An in-depth study of the relations of the mental map and several formal measures can be found in [20], in the case of orthogonal graphs. The authors define mathematical criteria to reflect the idea of stability across drawings based on the comparison of different layouts.

In particular, they define measures and conduct a student survey to evaluate them. The authors conclude this study with the analysis of the correspondence between theoretical results and the survey, thus leading to a ranking of the measures defined, where among the most important are the relative distance, the nearest neighbor-within position, and the average and maximum distances.

Considering that the method based on the relative position [106] is somehow limited, we approach the stability of hierarchical graphs taking into account both relative and absolute distances of the vertices position by including additional constraints. Our proposal is in line with previous studies. In the case of orthogonal graphs [20], the authors consider different measures based on the distances to achieve stability. Similarly, Di Battista et al. [6] proposed to approach the incremental problem with the *coordinates scenario*, in which the coordinates of some vertices and edges may change by a small constant, because of the insertion of a new vertex and its incident edges. Branke [18] pointed out that we cannot state that relative position is better than absolute position in terms of preserving the user's mental map, and both can be of interest.

To sum it up, based on the previous studies and the examination of practical examples, in Section 3.6 we describe a model to preserve the mental map of incremental hierarchical drawings, while minimizing the number of crossings. In particular, we fix the relative position between the original vertices, and keep their absolute position within a short bounded distance, K, of their original position.

3.6 A Mathematical Programing Model for the Constrained-IGDP

With in mind the notations summed up in Table 3.1, let $H = (G, k, L)$ be a hierarchical graph and $D = (H, \Phi_0)$ its drawing. Moreover, given a vertex v in layer t, let $\Lambda_{t-1}(v) = \{u \in V : (u, v) \in E\}$ be the set of vertices in layer $t-1$ adjacent to v, and symmetrically, denote with $\Lambda_{t+1}(v) = \{u \in V : (v, u) \in E\}$ the set of vertices in layer $t+1$ adjacent to v, such that $\Lambda(v) = \Lambda_{t-1}(v) \cup \Lambda_{t+1}(v)$.

Keeping the same number of layers, we can consider the addition of some vertices $\hat{V}$ and edges $\hat{E}$ to the original graph G, obtaining respectively an incremental graph $IG = (IV, IE)$ and an incremental hierarchical graph $IH = (IG, p, L)$, where $IV = V \cup \hat{V}$ and $IE = E \cup \hat{E}$. As previously, the sets of vertices and edges can be written as sequences of disjoint sets $IV = IV^1 \cup \ldots \cup IV^k$ and $IE = IE^1 \cup \ldots \cup IE^{k-1}$. Let's denote m_t the number of vertices in the incremental graph in layer t, i.e., $m_t = |IV^t|$. The Incremental Graph Drawing

Problem (IGDP) consists of finding a drawing $ID = (IH, \Phi)$ that minimizes the number of edge crossings while keeping the same relative position between the original vertices as in the original drawing D.

Literature dealing with IGDP is scarce. In fact, we are only aware of paper [105], which is limited to a bipartite graph, and more recently, the work reported in [136] for the multilayer case. In [105], the authors describe a branch-&-bound procedure that is tested on a relatively small graph and a meta-heuristic procedure based on GRASP [50] applied to medium- and large- sized instances. This seminal work is improved in [106] and extended to more than 2 layers in [136], where the authors propose a Variable Neighborhood Scatter Search for the IGDP.

In the following we describe an alternative approach, which, together with the relative position constraints, also requires that the positions of the original vertices are constrained to be close to their positions in the original drawing. The mathematical model for the Constrained Incremental Graph Drawing Problem (C-IGDP) is as follows:

$$\text{(C-IGDP)} \quad \min \quad \sum_{t=1}^{k-1} \sum_{(u,v),(u',v') \in IE^t} c^t_{uvu'v'}$$

s.t.

$$-c^t_{uvu'v'} \leq x^{t+1}_{vv'} - x^t_{uu'} \leq c^t_{uvu'v'}, \qquad \forall\, (u,v),(u',v') \in IE^t, v < v',\ \forall t \tag{3.15}$$

$$1 - c^t_{uvu'v'} \leq x^{t+1}_{v'v} + x^t_{uu'} \leq 1 + c^t_{uvu'v'}, \quad \forall\, (u,v),(u',v') \in IE^t, v > v'\ \forall t \tag{3.16}$$

$$0 \leq x^t_{uv} + x^t_{vu'} - x^t_{uu'} \leq 1, \qquad \forall\, u,v,u' \in IV^t, u < v < u',\ \forall t \tag{3.17}$$

$$x^t_{uv} + x^t_{vu} = 1, \qquad \forall\, 1 \leq i < j \leq m_t,\ \forall t \tag{3.18}$$

$$x^t_{uv} = 1, \qquad \forall\, u,v \in V^t,\ \varphi^t_0(u) < \varphi^t_0(v),\ \forall t \tag{3.19}$$

$$\max\{1, \varphi^t_0(u) - K\} \leq \varphi^t(u), \qquad \forall\, u \in V^t,\ \forall t \tag{3.20}$$

$$\min\{\varphi^t_0(u) + K, m_t\} \geq \varphi^t(u), \qquad \forall\, u \in V^t,\ \forall t \tag{3.21}$$

$$x^t_{uv}, c^t_{uvu'v'} \in \{0,1\}, \qquad \forall\, (u,v),(u',v') \in IE^t, \forall t. \tag{3.22}$$

Constraints (3.18) and (3.19) preserve the ordering of the original vertices, while (3.20) and (3.21) are required to restrict the position of the original vertices. Let K be a parameter representing a distance slack between the original position and the new one. Without loss of generality, we suppose that u is in layer t, and $\varphi_0^t(u)$ is the original position of vertex u. The new position $\varphi^t(u)$ where vertex u can be relocated in the solution must be such that:

$$\max\{1, \varphi_0^t(u) - K\} \leq \varphi^t(u) \leq \min\{\varphi_0^t(u) + K, m_t\}, \ \forall\, u \in V, \tag{3.23}$$

where m_t represents the number of vertices in the incremental graph in layer t. Table 3.11 summarizes all the definition used in this section.

Symbol	Definition
G	Original graph: $G = (V, E)$
V	Set of original vertices of G
E	Set of original edges of G
k	Number of layers
H	Hierarchical graph: $H = (G, k, L)$
IH	Incremental graph: $IH = (IG, k, L)$
L	Function that indicates the label of the layer that contains each vertex
$\hat{V}$	Set of incremental vertices
IV	Set of vertices in the incremental graph IH: $IV = \hat{V} \cup V$
V^t	Set of vertices in layer t in H: $V = V^1 \cup \cdots \cup V^k$
E^t	Set of edges from V^t to V^{t+1} in H: $E = E^1 \cup \cdots \cup E^k - 1$
n_t	Number of original vertices in layer t: $n_t = \lvert V^t \rvert$
IV^t	Set of vertices in layer t in IH: $IV = IV^1 \cup \cdots \cup IV^k$
m_t	Number of vertices in layer t: $m_t = \lvert IV^t \rvert$
IE	Set of edges in the incremental graph IH
D	Drawing: $D = (H, \Phi_0)$
Φ_0	Set of permutation $\{\varphi_0^1, \ldots, \varphi_0^p\}$
ID	Incremental graph drawing: $ID = (IH, \Phi)$
Φ	Set of permutation $\{\varphi^1, \ldots, \varphi^k\}$
$C(D)$	Number of crossings of a drawing D
$\Lambda(v)$	Set of all vertices adjacent to v
K	Constraint limit

Table 3.11 Symbols and Definitions.

3.7 Solution Methods

In this section, we propose three GRASP constructions, C1, C2, C3, a memory-based construction, C4, and two improving methods: a local search and a tabu search. We consider their combination in four different algorithms for the C-IGDP problem:

- GRASP1: C1 + Local Search.
- GRASP2: C2 + Local Search.
- GRASP3: C3 + Local Search.
- TS: C4 + Tabu Search.

3.7.1 GRASP constructive methods

We propose three different ways to obtain an initial solution in the GRASP construction phase. While applying these methods, we should keep in mind that each original vertex v has to be placed in a position between $\max\{1, \varphi_0(v) - K\}$ and $\min\{\varphi_0(v) + K, m_{L(v)}\}$.

In a first analysis, we may say that the greedy function is expected to be based on the objective function and therefore the semi-greedy selection in GRASP has to reflect good values in terms of the objective. Note however that if the objective function is relatively time consuming to evaluate, it is a common practice in heuristic search to employ an alternative evaluation to guide a method. This evaluation has to be fast and somehow connected with the objective value. In this section we describe two types of approaches to design a constructive method. The first one employs an alternative evaluation based on the vertex degree, and the second one is based on the direct number of crossings.

Our first method, called C1, constructs an initial solution from scratch. The method starts with the random selection of a vertex v among those with maximal degree. This vertex is placed in a random position in its layer, taking into account that if it is an original vertex, the position cannot be greater than $\min\{(\varphi_0(v) + K, m_{L(v)}\}$ or less than $\max\{(1, \varphi_0(v) - K)\}$. In the next steps, the candidate list CL is formed by all the unassigned vertices, where the degree $\rho(v)$ of a vertex v is calculated with respect to the partial solution. Elements of the restricted candidate list RCL are all vertices v whose degree $\rho(v)$ is within a percentage $\alpha \in [0, 1]$ of the maximum degree $\rho_{max} = max\{\rho(v) : v \in CL\}$:

$$RCL = \{v \in CL : \rho(v) \geq \alpha\rho_{max}\}. \tag{3.24}$$

The next vertex v^* to be added to the partial solution is randomly selected from RCL. The vertex v^* is placed in its layer, say layer l, in the position prescribed by the barycenter method, $bc(v^*)$.

The barycenter is probably the most frequently applied method to order vertices in hierarchical graphs. It simply computes the average position of their neighbors and sorts vertices with respect to these numbers. More precisely, the barycenter method estimates the position of vertex v^* in layer l as the average position of its neighbors (adjacent) vertices, $\Lambda(v^*)$. In mathematical terms:

$$bc(v^*) = \frac{\sum_{u \in \Lambda_{l-1}(v^*)} \varphi^{l-1}(u)}{2\,|\Lambda_{l-1}(v^*)|} + \frac{\sum_{u \in \Lambda_{l+1}(v^*)} \varphi^{l+1}(u)}{2\,|\Lambda_{l+1}(v^*)|}. \tag{3.25}$$

Each vertex is placed in the closest feasible position prescribed by its barycenter, $bc(v^*)$, computed as in (3.25) with respect to the adjacent vertices that are already in the partial solution. If the vertex is an original vertex, $v^* \in V$, a *feasible position* in the C-IGDP must satisfy the problem constraint and cannot be less than $\max\{1, \varphi_0(v) - K\}$ or larger than $\min\{\varphi_0(v) + K, m_{L(v)}\}$.

The second constructive method, called C2, considers that we already have a partial solution of the problem given by the original drawing $D = (G, \varphi)$. As in [136], C2 starts with D and iteratively adds one incremental vertex in each iteration. Initially, the candidate list contains the incremental vertices $\hat{V} = IV \setminus V$. As in C1, the greedy function is based on the vertex's degree with respect to its adjacent vertices in the partial solution. The vertex v^* to be included is selected at random from RCL, which contains all the unassigned incremental vertices with a degree higher than or equal to α times the maximum degree (3.24). The method computes the barycenter by calculating equation (3.25) and inserts v^* in the closest feasible position to it.

Figures 3.11 to 3.14 illustrate procedure C2. The incremental graph consists of three layers with four original vertices $0, 1, 2, 3$ and two incremental ones A, B in the first layer, labeled Layer 0. Layer 1 has six vertices, three of them are original, $0, 1, 2$, and the other three are incremental, A, B, C. Finally, the last layer, labeled Layer 2, consists of six vertices, being $0, 1, 2, 3$ the original ones, and A, B the incremental ones. Figure 3.11 also shows all the edges between pairs of vertices. For example, vertex 0 in Layer 1 is connected to vertex 0 and A in Layer 0, and to vertices 0 and B in Layer 2. We consider in this example the position constraint value $K = 1$.

To feed the constructive method C2, all the original vertices are copied in the solution in consecutive positions. Figure 3.12 shows the initial partial solution with the original vertices in the first positions. Note that, positions [4] and [5] are free in Layer 0 and Layer

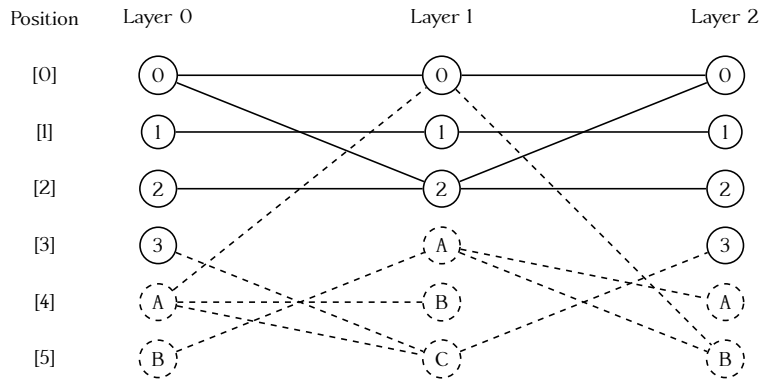

Fig. 3.11 Graph to illustrate the C2 method.

2 and in Layer 1 the free positions are [3], [4] and [5]. Firstly, RCL elements are all the incremental vertices and then at each iteration, one of them is added to the partial solution.

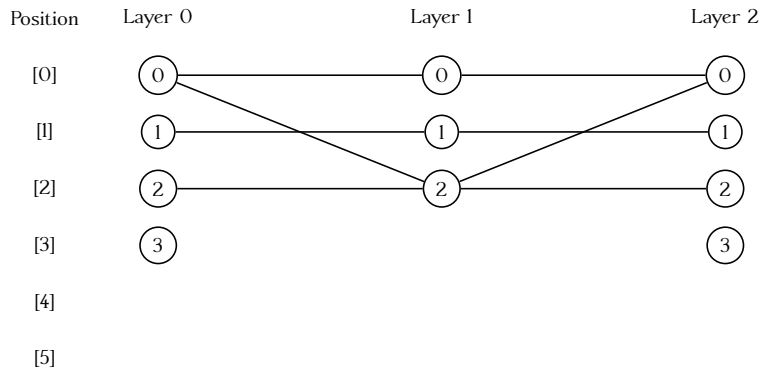

Fig. 3.12 Initial partial solution.

Let us consider the partial solution in one iteration of the construction algorithm C2, shown in Figure 3.13. Four incremental vertices are already in the partial solution, vertex A in Layer 0, vertices B and C in Layer 1 and vertex B in Layer 2. In each layer, all original vertices were moved one position down in order to insert the incremental vertices, A (Layer 0), B (Layer 1) and B (Layer 2) in the first position, called position [0]. These movements are feasible since K is equal to 1, and each vertex can move one position up or down to its original position.

In this iteration the candidate list contains vertices B from Layer 0, A from Layer 1, and A from Layer 2, too. Suppose that after the construction of the RCL the vertex to be inserted in the partial solution is A from Layer 1. If we compute its barycenter with equation (3.25), then $\mathrm{bc}(A) = 0$. Note that to compute this value we only have to

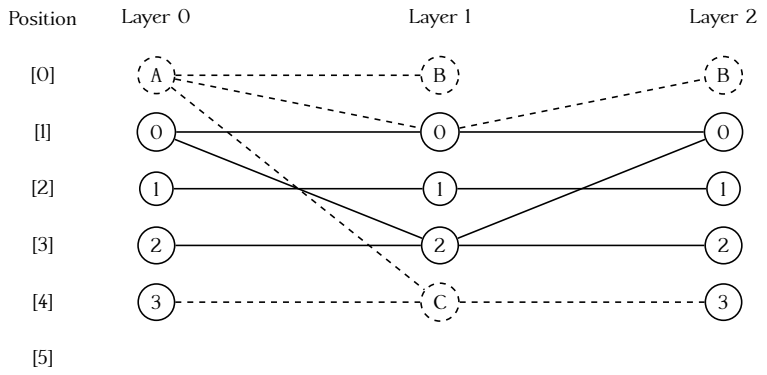

Fig. 3.13 Partial solution in an iteration of C2 construction phase.

consider the vertices adjacent to A that are already in the partial solution. However, in this particular case, A has only one adjacent vertex in the solution, vertex B (Layer 2).

As $bc(A) = 0$, the candidate position to insert A in Layer 1 is $[0]$, but this position is already occupied by vertex B. As the position constraint value is $K = 1$ and vertices 0, 1 and 2 were already shifted to one position down in previous iterations, then vertex A cannot be inserted in positions $[0]$, $[1]$, $[2]$ and $[3]$. We then insert A in the closest feasible free position, which in this case is $[4]$.

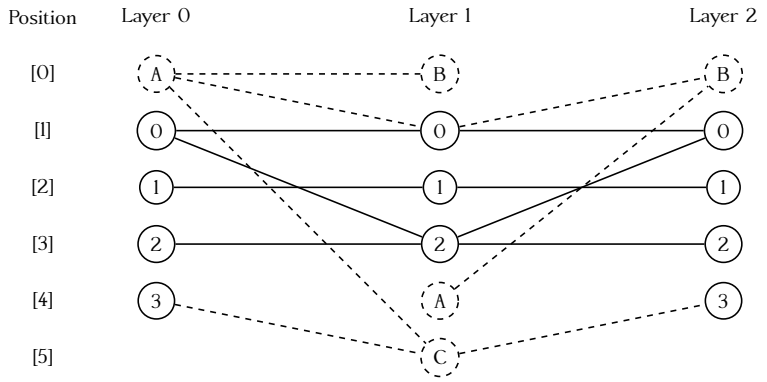

Fig. 3.14 Partial solution after an iteration of C2 construction phase.

In the basic GRASP construction phase, at each iteration, the choice of the next element to be added is determined by ordering all candidate elements, in a candidate list with respect to a myopic greedy function. This function measures the benefit of selecting each element. In the previous construction methods, C1 and C2, this function is based on the vertices' degree. In our last construction method, C3, function $g(v, q)$ computes the number of edge crossings when vertex v is inserted in position q in its layer. To calculate the best insertion for vertex v, the greedy function $g(v)$ computes the minimum of the $g(v, q)$ values for all

```
1  ID ← D;
2  CL ← IV \ V;
3  forall v ∈ CL do
4  |  compute g(v) and τ;
5  forall v ∈ CL do
6  |  if g(v) ≤ τ then
7  |  |  RCL ← RCL ∪ {v};
8  while ID is not complete do
9  |  v* ←select_node_randomly(RCL);
10 |  ID ← add_node_to_solution(v*);
11 |  recompute g and τ;
12 |  rebuild RCL;
13 return ID
```

Fig. 3.15 Constructive phase C3 for the C-IGDP.

q-positions. In mathematical terms, $g(v) = \min_q g(v, q)$. If the vertex v is selected, then it is placed in this best position in which the number of crossings is minimized. Initially, as in the case of C2, the method starts with the original drawing. In subsequent iterations, the candidate list CL is formed with all unselected vertices and the RCL consists of all vertices in CL whose the number of edge crossings is lower than or equal to a threshold τ. That is $RCL = \{v \in CL : g(v) \leq \tau\}$, where

$$\tau = \min_{v \in CL} g(v) + \alpha \left(\max_{v \in CL} g(v) - \min_{v \in CL} g(v) \right). \tag{3.26}$$

The parameter α controls the balance between the diversity and the quality of the solution, and it is empirically tuned (see Section 3.9). Then, a vertex v^* is selected at random among the RCL elements. If the position associated to v^* is not free, then previous vertices are shifted up as in C2 if it is possible. The pseudocode of construction phase C3 is described in Figure 3.15.

3.7.2 Memory construction procedure

GRASP constructions are memory-less methods, since no information is recorded and used from one construction to the next. In other words, it performs an independent sampling in the solution space. Conversely, it is maybe beneficial to save information from the past history thus designing constructions performing a guided selection in the solution space. We consider the inclusion of a frequency memory function $freq(\cdot, \cdot)$ to modify the evaluations of the greedy function with the inclusion of the recorded information too. Specifically,

we record in $freq(v, q)$ the number of times that vertex v was inserted in the solution in position q in the previous iterations. Then, we modify the evaluation of the attractiveness of each non-selected vertex in the current construction to favor different types of solutions, which were not generated in previous iterations. Algorithm C4 performs the same steps as C3 but, instead of using the greedy function $g(v) = \min_q g(v, q)$, it uses:

$$g(v) = \min_q \left(g(v, q) + \beta freq(v, q) \right), \tag{3.27}$$

where β is a critical parameter between $[0, 1]$. The first iteration of C4 produces the same solution as C3 since $freq(v, q) = 0$, for all vertices v and positions q. But then, in subsequent iterations the method favors the selection of those vertices with low $freq(\cdot, \cdot)$ values.

3.7.3 Local Search Procedure

Our local search method explores each layer from 1 to k, one by one, searching for an improving move. We consider swapping the position of two new vertices as the first mechanism in the local search. We called $N_0(ID)$ to this neighborhood where ID is initially the solution obtained with one of the constructive methods described above. In a given layer, the method examines the new vertices, starting from the first one. For that vertex, we consider its possible swapping with all the new vertices in its layer.

The method performs the best feasible move if it improves the objective function (i.e., if it reduces the number of crossings). Then, it resorts to the next new vertex in the layer (following the current order) to try to swap it. Note that, since we only swap new vertices here, all these moves are feasible. When we finish the exploration of a layer, say l, we consider the next one $l + 1$, and apply the same procedure. This local search performs multiple sweeps from the first to the last layer until no further improvement is possible. In short, it implements the so-called best strategy over a swapping move. We call *Swap* this local search phase based on neighborhood $N_0(ID)$, whose pseudo-code is reported in Figure 3.16.

We complement our local search with a second phase, called *Insertion*, based on neighborhood $N_1(ID)$. Specifically, this phase scans the layers from 1 to k, and within each layer it considers the incremental vertices in their current ordering to perform an insertion. Given a new vertex v, the method explores all its feasible insertions in previous position. Note that we have to check here the feasibility of the original vertices. By a feasible move we mean that the position of original vertices is within its limits. If $\varphi(v)$ is the position of vertex v after the move, then $\varphi(v) \in [\max(\varphi_0(v) - K, 1), \min(\varphi_0(v) + K, m_{L(v)})]$. If

```
1  best-cost ← C(ID);
2  ID* ← ID;
3  improvement ← true;
4  while improvement do
5  |  improvement ← false;
6  |  forall layers: t = 1...k do
7  |  |  forall vertices: ν ∈ V^t | ν is a new vertex do
8  |  |  |  best-swap ← −1;
9  |  |  |  forall vertices: ν̄ ∈ V^t | ν̄ is a new vertex do
10 |  |  |  |  if ν̄ ≠ ν then
11 |  |  |  |  |  ID* ← swap(ν, ν̄);
12 |  |  |  |  |  if C(ID*) < best-cost then
13 |  |  |  |  |  |  best-cost ← C(ID*);
14 |  |  |  |  |  |  best-swap ← ν̄;
15 |  |  |  |  |  ID* ← swap(ν̄, ν);
16 |  |  |  |  if best-swap ≠ −1 then
17 |  |  |  |  |  ID* ← swap(ν,best-swap);
18 |  |  |  |  |  improvement ← true;
19 return ID*;
```

Fig. 3.16 Swapping phase in local search for C-IGDP.

the best feasible move in a previous position improves the current solution, we perform it; otherwise we consider the insertions of v in a posterior position, identifying the best feasible one. We perform the best move if it improves the solution. Once v has been examined, we resort to the next incremental vertex (from $\hat{V}$) in this layer.

As in the previous phase, we perform sweeps from layer 1 to p until no further improvement is possible. Our local search finishes when the two phases, *Swap* and *Insertion*, are performed, and returns the local optimum found.

Consider again the example in Section 3.7.1 and assume that after the construction phase shown in Figures 3.11 to 3.14, we obtain the feasible solution in Figure 3.17 with 13 crossings. In the first layer, Layer 0, swapping the vertices A and B does not decrease the number of crossings. The next layer to explore is Layer 1. In this case, exchanging positions of vertices A and C produces an improvement in the solution. The move is done and the the number of crossings is reduced to 11, as shown in Figure 3.18-left. As in the first layer, the possible moves in last layer do not produce an improvement, then the first phase stops since no further swaps can reduce the number of crossings. In the second phase, in Layer 0 and 1 there are no insertion moves for the incremental vertices improving

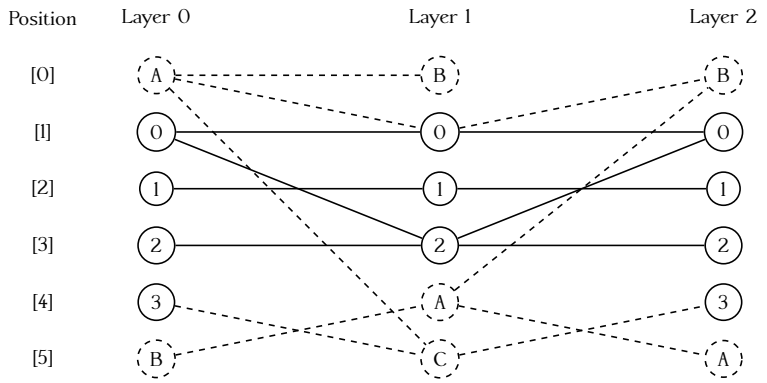

Fig. 3.17 Initial solution.

the quality of the solution. While, in Layer 2, by inserting the new vertex B in position [1] the number of crossings is reduced from 11 to 10 (see Figure 3.18-right).

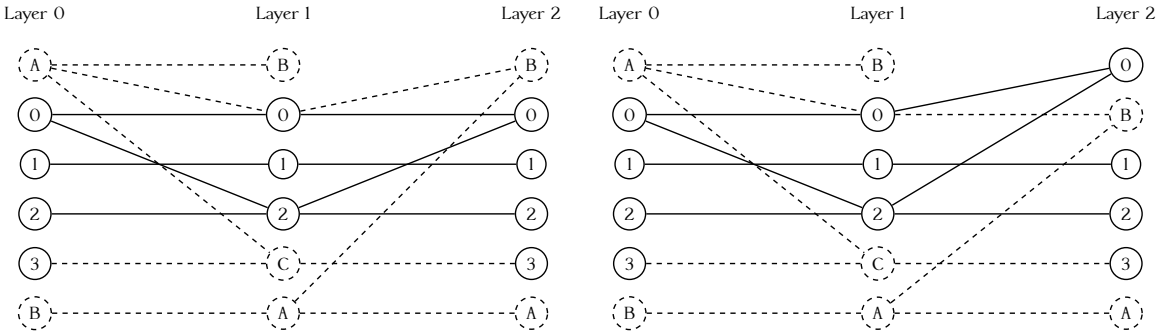

Fig. 3.18 Local search procedure. Left: Swap phase. Right: Insertion phase.

3.7.4 Tabu Search

To complement the memory based construction strategy, we implement a tabu search method, based on the neighborhood $N_1(ID)$. The procedure starts with the initial solution obtained by C4 and operates as follows. It scans the layers from 1 to p, and for each layer, it evaluates all possible insertions of the new vertices and performs the best one, i.e., the one that produces the minimum number of crossings. It is worth mentioning that the tabu search method always performs the best available move, even if it is a non-improving move. The moved vertex is made tabu-active and remains tabu for *tenure* iterations, which means that during this period it cannot be moved. In the next steps, the insertions are only possible with non tabu-active vertices. As it is customary in the tabu search methodology, the tabu status is overridden if the movement leads to a solution that improves upon the

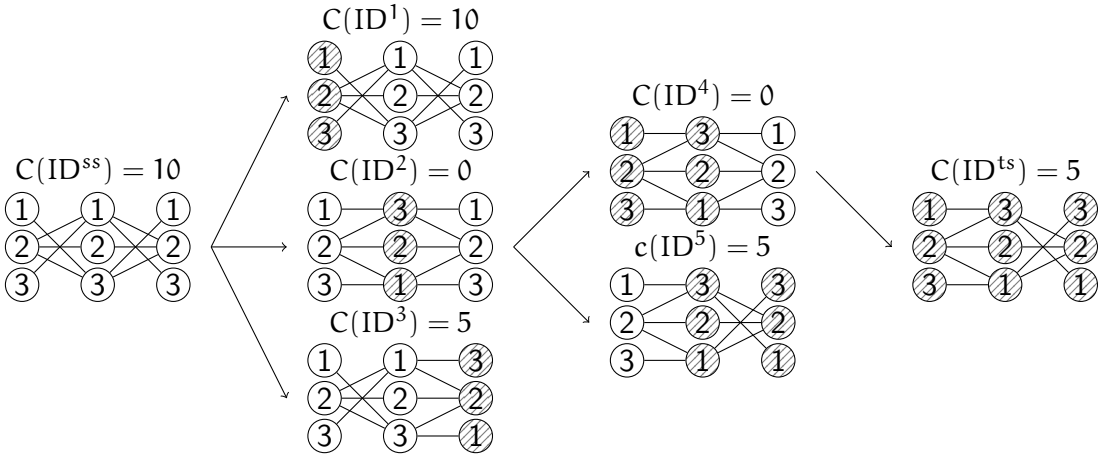

Fig. 3.19 Example of the Path Relinking procedure.

best solution found so far. The tabu search method finishes when a pre-established number of iterations is met, and the best visited solution is returned as its output.

3.8 Path Relinking post-processing

Path Relinking (PR) is an approach suggested in the context of tabu search to combine solutions by creating paths between them [73]. GRASP hybridized with PR was first proposed by Laguna and Martí in [93] as an intensification method, and hybridizations of GRASP with Path Relinking are deeply discussed in [56]. We consider here the variant known as forward PR [136] for the C-IGDP.

Path Relinking is performed between two solutions, ID^{ss} (source solution) and ID^{ts} (target solution) for the C-IGDP. Each single move generated by the PR consists of replacing 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^{ss} to ID^{ts} is $\frac{k(k+1)}{2} - 1$. For example, given the two solutions, ID^{ss} and ID^{ts} in Figure 3.19 with 3 layers and 3 vertices in each layer, the algorithm generates three different solutions exchanging one of the layers of ID^{ss} with the layers of ID^{ts}. Then, the path continues from the intermediate solution ID^2, which is the best solution found, and generates another two solutions, ID^4 and ID^5. Finally, the path is completed from ID^4 to ID^{ts}. The method finds two solutions that improve the current one ID^2 and ID^4, both with number of crossings equal to 0.

The procedure operates on a set of solutions, called *Elite Set* (ES), constructed with the application of the previous method. In order to obtain an elite set as good and diverse as possible, we start by considering as elite the $m = |ES|$ solutions obtained by our previous method. Then, in the subsequent iterations, we analyze whether the solution generates

ID^* qualifies to enter in the elite set or not. In particular, if solution ID^* is better than the best solution in ES, then the worst solution is replaced by ID^*. However, if ID^* is better than the worst solution and it is sufficiently different (with a distance larger than a parameter γ) from the other elite solutions then it is also inserted in ES. In this latter case, it will replace the worst most similar solution. We define the diversity distance or the difference between two solutions as the number of positions that are not occupied by the same vertices divided by the number of vertices in the graph. The PR finishes when the paths between all pairs of elite solutions have been explored.

3.9 Computational Experiments II

This section details our computational experiments to study the performance of the procedures presented above. In particular, we consider the following methods: GRASP1 (C1+Local Search), GRASP2 (C2+Local Search), GRASP3 (C3+Local Search), and TS (C4+Tabu Search). Additionally, when we disclose the best GRASP variant, then we couple it with PR, and called the resulting method as GRASP+PR. In the following subsections, we first outline the experimental framework (see Subsection 3.9.1), and then we analyze the performance of these methods with respect to their parameters and settings in Subsection 3.9.2. This subsection also includes a comparison of the GRASP variants and Tabu Search. Finally, we compare in Subsection 3.9.3 our best heuristics with LocalSolver, and CPLEX, run with the integer linear programing model proposed above. LocalSolver is a well-known commercially available optimization software (localsolver.com). It is a black-box solver that uses metaheuristic methodologies to solve any combinatorial problem. It provides high-quality solutions in short computing times, and it is currently considered an alternative to customized heuristic methods.

3.9.1 Experimental Setup

All the procedures were implemented in C++, compiled with gcc 5.4.0, and the experiments were conducted on an Intel Corei7-4020HQ CPU @2.60Ghz x 8.
The set of instances used is available on-line in the web-page http://www.optsicom.es/igdp. In line with previous graph drawing papers [136], this set consists of 240 instances with four different numbers of layers 2, 6, 13 and 20, and 0.065, 0.175 and 0.3 graph densities. The number of layers p is an input to the graph generator and the number of vertices in each layer is randomly chosen between 5 and 30. For each vertex u in layer t, an edge to a randomly chosen vertex v in layer $t + 1$ is included. In addition, the generator checks

that all vertices in the last layer have a degree of at least one. If a vertex in layer p is found with a zero degree, an edge is added to a randomly chosen vertex in layer $p-1$. The generator then adds enough edges to cover the difference between the current number and the number that results from the desired density. We then applied the barycenter algorithm to obtain a drawing for each graph. Finally, of the 20 instances generated for each combination of number of layers and density, 10 are augmented by incrementing the number of vertices by 20%, and 10 are augmented incrementing by 60%.

We divide the test set of 240 instances in four subsets according to the number of layers, $p = 2, 6, 13$, and 20, with 60 instances in each one. The average number of vertices per layer is approximately 25, and the average number of edges between consecutive layers is approximately 90. To give the reader an idea of the instances dimension, we can say that according to this average computation, an instance with for example 13 layers would have close to $n = 325$ vertices and 1080 edges. To avoid the over training of the methods, we consider a subset with a 10% of them, 24 instances, 6 in each subset, with different sizes and densities, to fine tune the parameters of the algorithms. We chose them to be representative of the entire set. Specifically, their number of vertices ranges from $n = 32$ (corresponding to a 2-layer instance) to $n = 960$ (a 20-layer instance). We call this subset the *training set*, as opposite to the entire set of instances called the *testing set*.

During our experiments, we report the following measures:

- $\bar{C}$: Average number of crossings.
- $\%\ dev$: Average percent deviation with respect to the best solution found in the experiment.
- $Best$: Number of instances for which a procedure is able to match the best-known solution.
- $Score$: It is calculated as $(s(p_r - 1) - r)/(s(p_r - 1))$, where p_r is the number of procedures being compared, s is the number of instances, and r is the number of instances in which the $p_r - 1$ competing procedures obtain a better result. Roughly speaking, the score is inversely proportional to the fraction of instances for which the competing procedures produce better solutions than the procedure being scored. Hence, the best score is 1 (when there is no better procedures, $r = 0$) and the worst is 0 (when all the procedures in each instances are better than the procedure being scored, $r = s(p_r - 1)$)
- $Time$: Time in seconds to execute the method.

- % gap: Average percentage deviation of the objective function value of the new method with respect to the solution value obtained with CPLEX.

An important element in the model is the value of the parameter K, which measures the distance between the original position of the vertices and the feasible ones. It is related to the problem definition since its objective is to keep vertices close to their original positions to preserve the user's mental map. It is clear that $K = 1$ satisfies this condition but the resulting problem is extremely constrained. We therefore consider low values of this parameter and test if a marginal increase would permit to obtain better solutions in terms of number of crossings. In particular, we tested $K = 1, 2$, and 3. Therefore, for each instance in our set we generated three instances, one for each K value in most cases (note that the number of added vertices in each layer has to be larger than K, thus some values of K cannot be considered). Then, the training set has a total of 62 instances, and the testing set a total of 609 instances.

3.9.2 Preliminary Experiments

In this section, we first perform an experimental study to fine tune the algorithmic parameters of our methods. We have designed four constructive algorithms based on a greedy function (see Section 3.7.1). These constructive methods, namely C1, C2, C3 and C4 are parametrized by α, which balances greediness and randomness. In this first experiment, we evaluate the influence of this parameter by considering three different values of α: 0.25, 0.50, 0.75. We also include a variant labeled random, where the method randomly selects an α value in the range $[0, 1]$ for each construction. Table 3.12 shows the corresponding results when generating 500 independent constructions for each instance in the training set. This table shows for each procedure, the average number of crossings ($\bar{C}$), the average percent deviation from the best solution found ($\%\ dev$), the number of best solutions ($Best$), the score statistic ($Score$), and the CPU-time in seconds required to execute the method ($Time$).

α	$\overline{C}$	$\%$ dev	Best	Score	Time
0.25	17711.8	0.63	19	0.49	0.92
0.50	17747.1	0.69	12	0.47	0.96
0.75	17780.1	0.73	14	0.38	0.96
random	**17685.7**	**0.16**	**39**	**0.87**	**0.93**

Table 3.12 Fine-Tune parameter α for the constructive C1 procedure.

Results in Table 3.12 clearly show that the best performance for C1 is achieved by the *random* variant. This variant is able to obtain 39 best solutions out of the 62 instances, which compares favorably with the other variants. Likewise, we analyze the value of α for the constructive methods C2, C3, and C4, obtaining a similar result. Computational effort does not play a role here because all procedures build solutions in a negligible amount of time.

In our second preliminary experiment, we undertake the exploration of the memory construction procedure, C4, which has an additional search parameter, β, to be tuned. This parameter controls the number of times that a vertex has been selected in previous iterations. We test $\beta = 0.25, 0.50, 0.75$, and a random selection. Table 3.13 summarizes the results of this experiment with the same measures described above, and shows that the best solutions on average are obtained with β selected at random in each iteration (*random* option).

β	$\overline{C}$	% dev	Best	Score	Time
0.25	17293.6	0.56	15	0.54	9.94
0.50	17288.1	0.82	9	0.43	8.83
0.75	17286.5	0.88	10	0.37	8.52
random	**17248.1**	**0.30**	**33**	**0.73**	**9.27**

Table 3.13 Fine-Tune parameter β for the constructive C4 procedure.

Table 3.14 compares the four construction procedures with their parameter values as indicated above in the training set instances. In order to run the methods for similar CPU times to perform a fair comparison, the number of constructions performed with each one is different. In particular, 1000 constructions for the C1 and C2 procedures, 100 for C3 and C4. As in the previous tables, the deviation values reported in this table are obtained considering the best solutions found in the experiment.

Procedure	$\overline{C}$	% dev	Best	Score	Time
C1	17645.5	3.79	0	0.204	1.837
C2	17681.9	3.77	0	0.156	1.973
C3	17395.3	1.01	29	0.796	2.076
C4	17387.6	0.27	36	0.860	2.185

Table 3.14 Performance comparison of the construction methods.

Our local search method is based on two different moves, swaps and insertions. Initially, it performs swaps between incremental vertices, and then, in subsequent iterations, it

implements insertion moves. In this preliminary experiment, we evaluate the effectiveness of each type of move (i.e., neighborhood structure) and its contribution to the final solution quality. In order to perform this analysis, we execute a C3 constructive phase coupled with three different local search procedures. We also include the solutions of the constructive method with no local search as a baseline in the comparison (C3). The three methods are: swap-only neighborhood (C3 + S), insertion-only neighborhood (C3 + I), and both (C3 + S + I). These four setups are tested on the whole *training set*, for this experiments we keep track of: the average number of crossings, the mean percentage deviation from the best value, the number of best solution found, the score statistics and the time to best.

The results reported in Table 3.15 show that, as expected, the combination of the two neighborhood strategies achieves the maximum number of bests and best crossing average, with extremely low deviation and very high score statistic. For this reason, in the next experiments, the local search used for the GRASP algorithm consists in S + I as described in Section 3.7.3.

Procedure	$\overline{C}$	% dev	Best	Score	Time
C3	17383.9	7.12	1	0.016	1.698
C3+S	16660.0	1.48	5	0.446	11.713
C3+I	16575.3	0.58	17	0.694	2.299
C3+S+I	**16497.9**	**0.00**	**60**	**0.989**	12.021

Table 3.15 Comparison among different local search setups.

An interesting question when comparing constructive methods is if they really need to produce high-quality solutions, or if their role is to obtain diverse initial solutions from which to apply the local search. To investigate this point, we perform an experiment to compare the value of the constructed solutions when applying C3, with the value of the improved ones after applying the local search method. In particular, we compute the correlation of these two values over 100 solutions for several instances. We obtain that in all the cases the correlation is relatively low. However, results are heterogeneous since in some instances the correlation is close to 0, while in others is close to 0.4. Figure 3.20-left shows the scatter-plot diagram of an instance with very low correlation, and Figure 3.20-right shows this diagram of another instance with a correlation close to 0.4. The x-axis represents the value of the constructed solution, and the y-axis the value of the improved one. A point is plot for each pair of associated values. In both diagrams the points are scattered over the plane and they are not aligned over a straight line. Note, however, that they present different patterns in terms of their correlation.

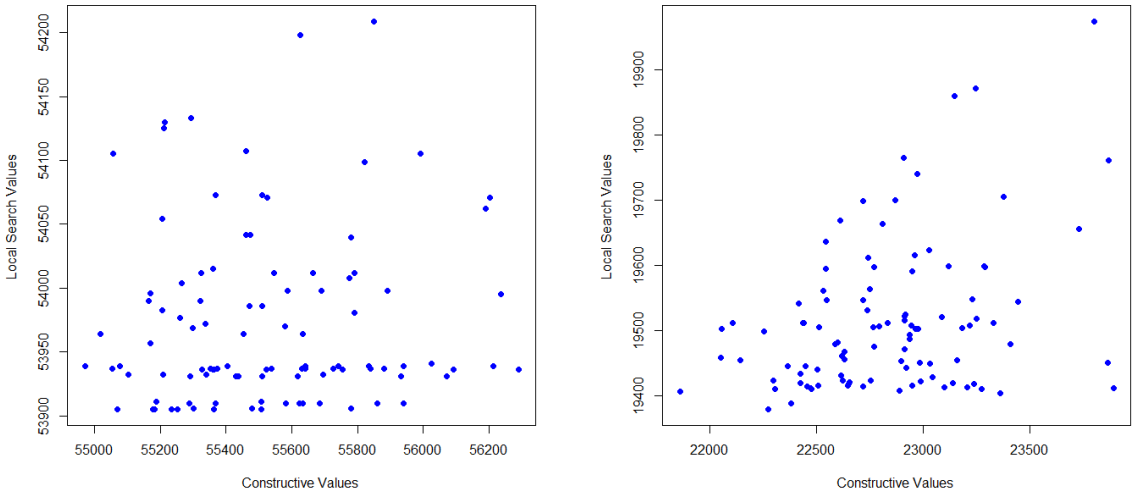

Fig. 3.20 Scatter Plot for two different instances.

Since results in the previous experiment indicate a different pattern across the instances in terms of their correlation, we cannot conclude that we should construct good solutions to obtain good local optima. Additionally, diversity is relatively difficult to directly evaluate on the constructed solutions to test their ability to produce different local optima. Therefore, we compare the quality of the solutions obtained with the complete method, once the local search has been applied. Table 3.16 shows the results of the solutions when these constructive methods are coupled with the local search. We analyze the combination of the GRASP constructive methods `C1`, `C2`, and `C3` with the local search, and `C4` with the tabu search. They are denoted as `GRASP1`, `GRASP2`, `GRASP3`, and `TS` respectively. All of them generate and improve 100 solutions.

Procedure	$\overline{C}$	$\%$ dev	Best	Score	Time
`GRASP1`	17193.7	0.14	307	0.63	7.61
`GRASP2`	17188.3	0.15	348	0.70	10.04
`GRASP3`	17188.3	0.13	**358**	0.72	11.12
`TS`	17183.3	**0.04**	335	0.73	34.36

Table 3.16 Performance comparison of GRASP variants and Tabu Search.

Table 3.16 shows that `GRASP3` is slightly better than `GRASP1` and `GRASP2`, both in average number of crossings ($\bar{C}$) and number of best results (Best). On the other hand, `TS` slightly outperforms `GRASP3` in terms of the average percentage deviation ($\%$ dev).

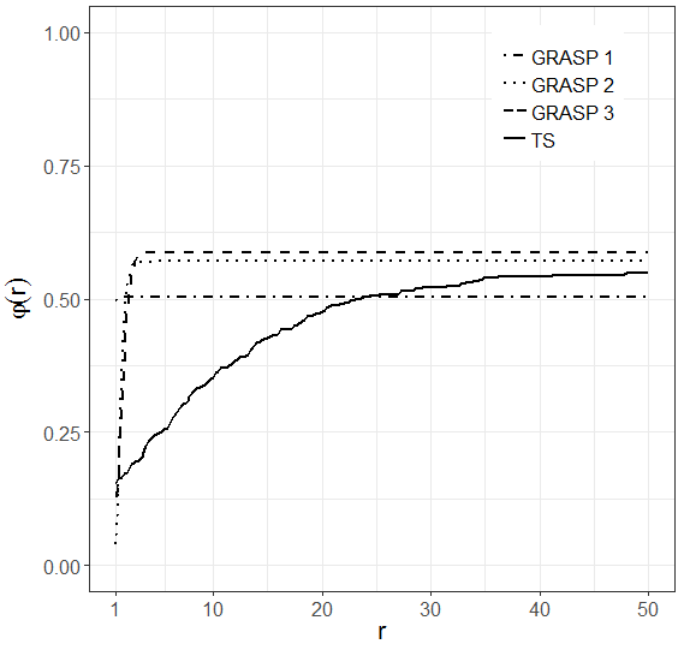

Fig. 3.21 Performance profile for three GRASP implementation and a TS.

However, it exhibits a lower number of best solutions (335, while GRASP3 is able to obtain 358). We therefore cannot conclude that one is better than the other, and select both GRASP3 and TS as our solving methods to be applied in the rest of the experimentation.

We complement the comparative analysis of the findings with a performance profile plot, as described in [39]. This plot consists in the cumulative distribution function for a certain performance metric, which is the ratio of the computing time of each procedure versus the best time among all of them. The performance profile of each heuristic shows the probability $\varphi(r)$ that the ratio is within a factor $r \in \mathbb{R}$ of the best possible ratio. Figure 3.21 confirms how GRASP1 is the fastest of the four heuristics. Indeed, this performance can be observed in $r = 1$, in which GRASP1 shows the highest probability of achieving the best solution in the shortest time. This findings are consistent with the ones collected in Table 3.16, since the performance profile does not select multiple best solutions, but considers as best solution the one reached in the shortest time. At the same time, we can observe how considering higher computational efforts, the other two GRASP implementations are able to outperform GRASP1, with GRASP3 exhibiting the highest probability to obtain the best solution. This switch in terms of performances happens for relatively low values of r, the upper bound on the relative time employed. Given the high number of best solution obtained and the quickly growing performance profile, we select GRASP3 as the

best-performing GRASP implementation. Moreover, the consistency evidenced by TS in Table 3.16, let us also consider TS as base methodology for our final experimentation.

Note that regarding the comparison between memory-less methods (GRASP in our case) and memory-based methods (Tabu Search), as we mentioned above, there is no clear winner. We cannot say that one systematically outperforms the other since each metric provides a different ranking. These results are in line with previous studies ([107], [26]), which also indicate that it is more a problem specific or implementation question, than a methodological one.

Our last preliminary experiment has the goal of setting the parameters of the Path Relinking post-processing. In particular, we set the value of the elite set size $|\mathrm{ES}| = 3$, and the distance parameter $\gamma = 0.2$). We do not reproduce the table of this experiment, since as in the previous ones, our selection is based on a trade-of between quality and computing time.

3.9.3 Final Experiments

In this last section, we undertake to compare GRASP3 and TS methods when solving the C-IGDP, as well as to compare them with other previous methods: CPLEX and LocalSolver. Additionally, we evaluate the contribution of the Path Relinking post-processing to the final quality of the solution. Figure 3.22 shows the typical search profile in which the current solution value is represented at different times in the search process. Specifically, this figure shows the time in seconds on the $\mathrm{x}-$axis, and the objective function value on the $\mathrm{y}-$axis of an instance with 444 vertices (a 20% of them are incremental) and 2228 edges.

Figure 3.22 shows the value of the solution constructed with C3 at 2 seconds. Then, when we move to the right-hand-side of the diagram, from 2 seconds to almost 60 seconds, we can see how the local search in GRASP3 is able to improve the solution from a value close to 24,000 to a value of 23,800. In the final stage of the profile, after 60 seconds, we can easily identify the PR application since the search profile shows a significant improvement in the value, which ends close to 23,650. Although this diagram only shows the performance for a single instance with 20 layers, we have empirically found that this is a typical performance for different instances.

Figure 3.22 shows the contribution of PR to the final solution of our complete method GRASP3+PR. We now complement this analysis by comparing GRASP3 with and without PR. Specifically, Figure 3.23 shows the search profile of GRASP3 over 150 seconds, and also the profile of a method consisting of applying first GRASP3 for 100 seconds, and then PR for the remaining 50 seconds. This figure clearly shows that it is worth investing the final search time on PR instead of continuously applying GRASP3 for the entire search. Note

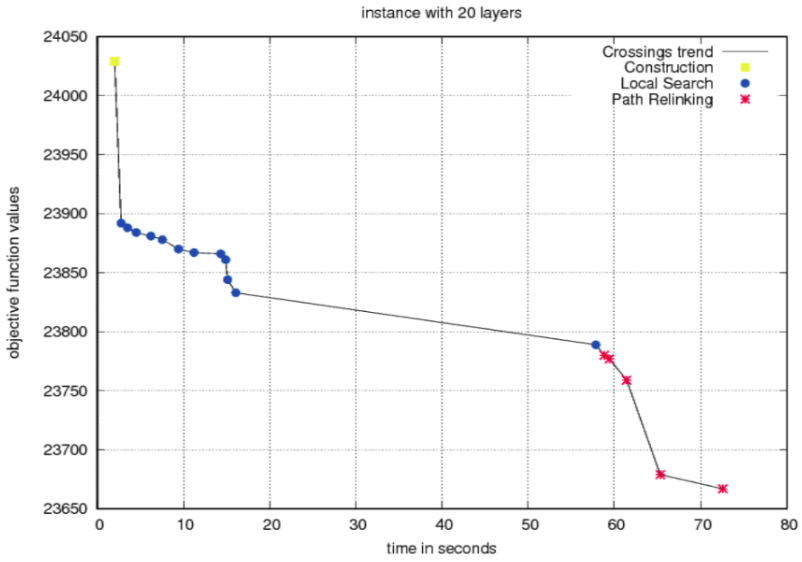

Fig. 3.22 GRASP3+PR Search Profile.

that at a certain time of the search GRASP3 is not able to further improve the solution and the profile stagnates on a certain value. At that point (around 100 seconds), PR is able to further improve the current solution. We can conclude that GRASP3+PR obtains high quality solutions, better than GRASP3 over a relatively long-term horizon.

We now compare GRASP3, TS, GRASP3+PR, and TS+PR with CPLEX and LocalSolver over the entire set of 609 instances. Table 3.17 shows the associated results classified by size. In this experiment, we run our heuristics for 100 iterations, which corresponds to moderate running times (from 0.5 to 50 seconds depending on the instance size). Since CPLEX performs an implicit enumeration from the mathematical model proposed in Section 3.6, it requires larger running times, so we configure it to run for a maximum of $1,800$ seconds. To give LocalSolver the opportunity to reach high-quality solutions, it is run with a time limit of 20 seconds on the instances with 2 layers, and $60, 150$ and 300 seconds on those with $6, 13$ and 20 layers respectively. As in previous experiments, this table shows, for each procedure, the average of number of crossings, the average percentage deviation from the best solution found, the number of best solutions, and the CPU-time in seconds. In addition, we also show the average percentage deviation between the heuristic solution value and the CPLEX best solution value ($\%gap$). Note that CPLEX is able to obtain the optimal solution in 572 out of the 609 instances.

Table 3.17 shows that, as expected, GRASP3+PR consistently obtains better results than GRASP3, and similarly TS+PR improves upon TS. Note that in the PR variants, only a small fraction of the total time is employed by PR, since it has the role of a post-processing

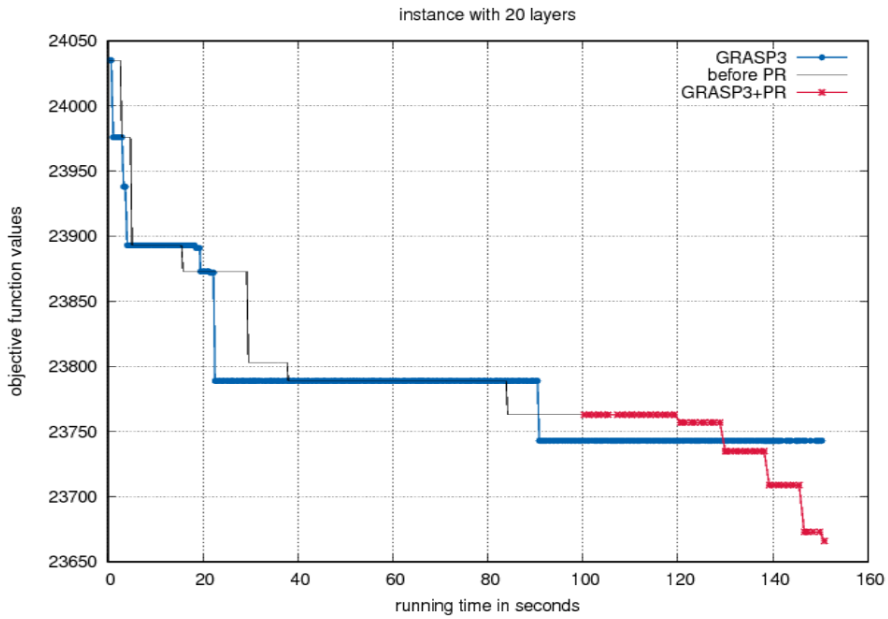

Fig. 3.23 Search Profile.

method. It is worth mentioning that CPLEX is able to obtain the exact solutions for the small instances in similar time than our heuristic methods. However, the situation changes when we move to the large instances, where heuristics are able to obtain solutions of similar quality than CPLEX in lower running times. In particular, we can see that in the instances with 20 layers, GRASP3+PR and TS+PR have an average gap value of -0.12 and -0.19 respectively, which indicates that the heuristic outperforms CPLEX on average. Note that this table also indicates that CPLEX obtains the optimal solution in 116 instances with 20 layers, while our heuristics only obtain a fraction of these optimal solutions. However, this is achieved by CPLEX at a large computational cost, and additionally, for the remaining instances in this set where CPLEX is unable to obtain the optima, GRASP3+PR and TS+PR obtain better solutions, which cause the average gap to take negative numbers. Note that our heuristics only employ a fraction of the running time required by CPLEX, which in many cases employs the total time permitted of $1,800$ seconds, as evidenced by the long average running times shown in this table. Table 3.17 also shows that LocalSolver presents a poor performance since in spite of running it for longer CPU times than the competing heuristics, it exhibits the largest deviations with respect to the best known solutions and optimal values.

We now summarize the results in this experiment in a different way. In particular, Table 3.18 shows the average value of the statistics considered aggregating the instances by their K-value. We can observe more differences between the heuristics gap and deviations with

Procedures	$\overline{C}$	% gap	% dev	Bests	Opt	Time
2 Layers ($32 \leq n \leq 96$), 171 instances						
CPLEX	2408.50	-	0.00	171	171	0.77
GRASP3	2409.19	0.14	0.14	161	161	1.11
GRASP3+PR	2408.92	0.14	0.14	164	164	1.18
TS	2408.58	0.00	0.00	163	163	1.03
TS+PR	2408.51	0.00	0.00	170	170	2.73
LocalSolver	2785.47	17.01	17.01	12	12	20.11
6 Layers ($48 \leq n \leq 288$), 159 instances						
CPLEX	9995.70	-	0.01	157	157	56.24
GRASP3	9997.43	0.13	0.14	81	81	5.53
GRASP3+PR	9994.32	0.08	0.08	100	98	5.39
TS	9999.73	0.20	0.20	63	63	5.48
TS+PR	9994.19	0.05	0.06	93	93	16.70
LocalSolver	11024.89	16.59	16.6	0	0	62.43
13 Layers ($104 \leq n \leq 611$), 141 instances						
CPLEX	23469.05	-	0.21	129	128	273.77
GRASP3	23319.41	0.13	0.33	28	27	15.22
GRASP3+PR	23305.22	0.02	0.22	49	44	15.34
TS	23334.38	0.24	0.44	26	26	15.64
TS+PR	23301.16	-0.07	0.14	38	31	47.42
LocalSolver	25530.24	15.95	16.16	0	0	162.06
20 Layers ($120 \leq n \leq 960$), 138 instances						
CPLEX	37918.20	-	0.38	118	116	383.73
GRASP3	37522.42	0.03	0.39	21	20	25.77
GRASP3+PR	37495.44	-0.12	0.24	36	29	28.39
TS	37540.99	0.11	0.47	22	22	26.72
TS+PR	37486.65	-0.19	0.17	39	26	86.26
LocalSolver	40954.07	14.30	14.68	0	0	328.77

Table 3.17 Comparison on entire benchmark set according to instance size

$K = 3$ than with $K = 1$, with this being due to the fact that CPLEX is able to solve almost every instance with $K = 1$. In line with the analysis in Table 3.17, we can conclude that our heuristics are performing well when comparing them with CPLEX and LocalSolver.

We complement our comparison with 10 additional very large instances with 50 layers and $1,000$ vertices. These instances are too large to be solved with CPLEX, therefore we limit this comparison to GRASP3+PR and LocalSolver. To set up a benchmark for future comparison, we report in Table 3.19 with $K = 1$, Table 3.20 with $K = 2$, and Table 3.21

Procedures	$\overline{C}$	% gap	% dev	Bests	Opt	Time
			K = 1, 240 instances			
CPLEX	17196.63	-	0.03	238	238	49.85
GRASP3	17164.45	0.03	0.06	132	131	5.89
GRASP3+PR	17160.88	0.00	0.04	157	156	6.32
TS	17162.75	0.02	0.05	149	149	6.27
TS+PR	17159.60	0.00	0.03	159	158	23.92
LocalSolver	18205.27	9.45	9.49	6	6	141.80
			K = 2, 210 instances			
CPLEX	17323.09	-	0.15	195	194	184.94
GRASP3	17183.44	0.06	0.20	93	93	11.01
GRASP3+PR	17172.75	-0.02	0.13	115	109	11.78
TS	17190.76	0.11	0.26	75	75	11.39
TS+PR	17170.58	-0.06	0.08	104	95	36.29
LocalSolver	19164.93	17.11	17.26	4	4	134.46
			K = 3, 159 instances			
CPLEX	17471.57	-	0.27	142	140	313.39
GRASP3	17230.84	0.30	0.56	66	65	19.17
GRASP3+PR	17210.94	0.15	0.41	77	70	19.80
TS	17254.76	0.33	0.58	50	50	19.15
TS+PR	17203.96	-0.08	0.17	77	67	52.53
LocalSolver	19413.82	24.57	24.85	2	2	121.49

Table 3.18 Comparison on entire benchmark set according to K value

with $K = 3$, the individual results of both methods on each instance. They run for 100 iterations for GRASP3+PR and a time limit of 900 seconds for the LocalSolver on each instance. Results in these tables clearly show the superiority of our procedure with respect to Local Solver in both solution quality and speed. Additionally, when comparing the solution value for different values of K, we can conclude that only a marginal improvement in the number of crossing is achieved when K increases, and therefore low values of K are recommended for the sake of stability in the sequence of drawings.

We perform now the so-called time to target plot, see [1], for this large instance with 20 layers and 512 vertices. This diagram shows the ability of a heuristic to match a target value (optimal value in our case). In particular, we run our heuristic GRASP3 for $n = 100$ trials and record the time to reach that target in each run. We sort the time values in increasing order: $t_1, \ldots, t_n$. The time to target plot, showed in Figure 3.24, depicts the cumulative probability $p_i = (i - 1/2)/n$ for each time value t_i for $i = 1, \ldots, n$. The plots

	GRASP3+PR		LocalSolver	
Instance	Crossings	Time	Crossings	Time
L50N1000.0	53048	49.92	59220	1019.69
L50N1000.1	47051	42.85	52037	1003.95
L50N1000.2	72018	64.73	78800	1058.08
L50N1000.3	101428	93.84	110649	1127.93
L50N1000.4	55112	51.73	61279	1016.93
L50N1000.5	89599	76.68	97372	1093.13
L50N1000.6	58829	54.32	65527	1023.40
L50N1000.7	79236	75.52	87359	1069.29
L50N1000.8	58147	55.45	65320	1028.31
L50N1000.9	84496	79.64	92960	1112.17

Table 3.19 Best values on very large instances, with $K = 1$.

	GRASP3+PR		LocalSolver	
Instance	Crossings	Time	Crossings	Time
L50N1000.0	52010	77.02	62547	1020.65
L50N1000.1	46152	66.63	54934	1004.17
L50N1000.2	70711	99.65	83344	1058.18
L50N1000.3	99767	142.35	108847	1128.66
L50N1000.4	54020	79.18	63984	1016.26
L50N1000.5	88247	115.74	100745	1093.07
L50N1000.6	57639	84.00	67388	1023.72
L50N1000.7	77716	111.58	91685	1070.21
L50N1000.8	57160	83.56	68692	1028.18
L50N1000.9	82910	123.70	96252	1113.10

Table 3.20 Best values on very large instances, with $K = 2$.

in this figure show the expected exponential runtime distribution for GRASP3. Therefore, linear speed is expected if the algorithm is implemented in parallel.

To finish our experimentation we consider the example shown in Figure 3.9 and obtain the incremental drawing with our new GRASP3+PR. Figure 3.25 shows the output of our method with $K = 4$. It is easy to check that the vertices in this figure are close to their original position in Figure 3.9. For example, vertices 3 and 12, which were in positions 1 and 17 respectively, are now in positions 2 and 21. The number of edge crossings of this graph is 4961, which compares favorably with the 6963 crossings in the initial drawing. Note however that the solution in Figure 3.10 has 4647 crossings. This is expected since we are solving a more constrained model. We believe that the marginal increase in the number of crossings of our model is completely justified by the stability obtained in the

	GRASP3+PR		LocalSolver	
Instance	Crossings	Time	Crossings	Time
L50N1000.0	51413	99.90	63430	1019.81
L50N1000.1	45361	83.13	57458	1004.67
L50N1000.2	69940	132.82	84770	1058.01
L50N1000.3	98704	187.58	120328	1128.18
L50N1000.4	53298	106.53	65782	1016.16
L50N1000.5	87324	155.52	103617	1093.30
L50N1000.6	56783	111.33	70991	1024.39
L50N1000.7	76828	152.64	94363	1072.14
L50N1000.8	56527	106.78	70587	1027.96
L50N1000.9	81838	160.89	91291	1112.31

Table 3.21 Best values on very large instances, with $K = 3$.

incremental drawing. In other words, the experimentation shows that our proposal provides a good trade-off between crossing reduction and drawing stability.

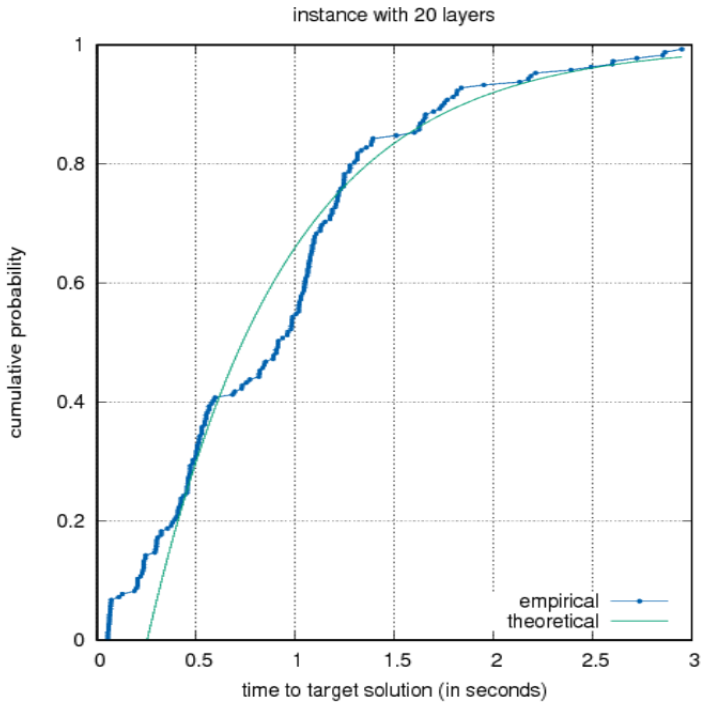

Fig. 3.24 Time to target plot.

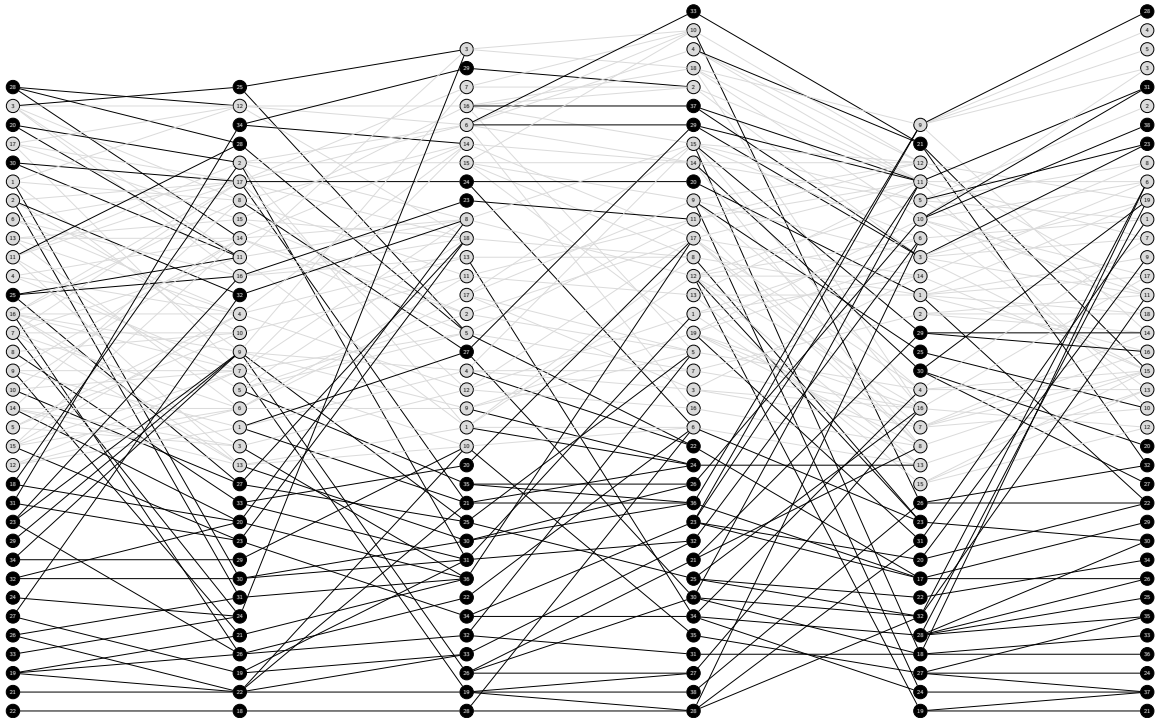

Fig. 3.25 Example optimized with new method.

Chapter 4

Handling Dynamic Informations in Network Optimization

In Chapter 3, studying the C-IGDP, we introduced the idea of optimization on a dynamic graph. Networks of this type are naturally subject to changes: either in their topology, i.e. their adjacencies, or in other distinctive parameters, such as arc-costs, node-weights, and so on.

Being able to properly tackle dynamic problems would not only bring theoretical advancements, but would also mean a step further towards a complete integration of optimized solutions in non-deterministic real-word scenarios.

While solving a problem on dynamic graphs, there are several approaches that can be considered. As in the case of the C-IGDP, at times it could be valuable to ensure continuity in the solutions obtained after repeated changes. In this case we modeled such requirements in terms of mathematical constraints. In others, the use of past informations can bring useful insights, able to dramatically speed up the algorithm, or simply ensure a more reliable solution.

In this chapter we thoroughly discuss the use of a simheuristic, specifically designed to address the Vehicle Routing Problem with Stochastic Demands, and compare its philosophy with the one founding re-optimization approaches. In particular, re-optimization refers to the re-use of past optimized solutions to quickly solve the problem after that small changes happened in the instance. To give an example of such strategies we consider the case of Shortest Path Problems (SPPs) on dynamic networks.

4.1 The Vehicle Routing Problem with Stochastic Demands

As in the case of other notable stochastic combinatorial optimization problems, a Stochastic Vehicle Routing Problem (SVRP) emerges whenever a random component is included in the mathematical model of a classic VRP formulation. The most-common examples of such random components are mainly related to the stochasticity of customer availability, customer demands, and travel times.

Given the critical importance of understanding the influence of uncertainty over decisions, which is a crucial matter in logistics and operations management, solving SVRPs is vitally important to both the simulation-optimization community and industry [68]. One of the initial contributions to the VRP with stochastic demands can be found in [12], where the author tries to: *(i)* determine a fixed routing sequence covering all potential customers; and then *(ii)* update the solution according to two different criteria once the information regarding customer demands becomes available.

The aim of the algorithm here described, and presented in [59], is to generate a 'reliable' or 'stable' solution for the VRPSD, i.e., a set of routes that can achieve a good performance not only in terms of average cost but also by being able to support moderate levels of variability in the values of the random customer demands.

The VRPSD is formally stated on a graph $G = (V, E)$, where the set of nodes includes both the depot, usually denoted by 0, and the location of customers, $V \setminus \{0\}$. A fleet of vehicles with restricted capacity is available at the depot, which is the starting and ending point for each trip. Moreover, a non-negative cost function $c : E \rightarrow \mathbb{R}^+$ associates a cost (either based on distance or on travel time) to each edge in E. For each $i \in V \setminus \{0\}$, a random variable x_i indicates the demand of goods that customer i requires. The goal of the problem is to find a set of routes with minimal total expected cost while serving all customers and satisfying the capacity constraints.

4.2 Simheuristics: Bringing Together Optimization and Simulation

The complexity of intractable combinatorial optimization problems, such as the VRP, heavily limits the size of the instances that can be solved with exact methods. Nevertheless, driven by the attractiveness of optimal solutions, several efforts have been made in the literature to design exact algorithms for routing problems. Gendreau et al. [67] provide optimal

solutions to the VRP with stochastic customers and demands by means of an integer L-shaped algorithm. In [96], a branch-and-cut method is described for solving the VRP with stochastic travel times. Other examples of exact algorithms that have been investigated are column generation [147] and branch-and-price [31]. At the same time, in order to solve larger instances, a parallel stream of research has focused on heuristic and metaheuristic algorithms. Thus, [69] propose a Tabu Search algorithm for the VRP with stochastic customers and demands. For the VRP with stochastic demands, the work of [103] describes a particle swarm optimization method, while [14] compare the results achieved by five different well-known metaheuristics. The common aspect shared by both exact and heuristic methods is the use of a dynamic approach, in which some of the stochastic values are only revealed once vehicles reach customers, at which point solutions can be dynamically re-optimized or updated with recourse actions.

The simheuristic framework is one of the most successful examples of hybrid simulation-optimization approaches. In this paradigm, the typical randomness of real-world optimization scenarios is effortlessly accounted for in the solution strategy, whether it has to be included in a stochastic objective function or a constraint.

One essential idea in the simheuristic concept is the strong relation between the strategy devised for solving a classical deterministic problem and the simulation of the random variables. These two different perspectives to the problem are handled respectively by the heuristic (or metaheuristic) core and by a stochastic simulation (in any of its forms). Thus, given a stochastic optimization problem, a first step of a simheuristic approach consists of the transformation of the stochastic variables into their deterministic counterparts. This allows the approach to efficiently obtain a deterministic solution by means of a heuristic algorithm. Subsequently, the quality of the constructed solution is evaluated in a stochastic environment to determine its real value in scenarios with uncertainty. This two-step process establishes a feedback system between the deterministic optimization algorithm and the stochastic simulation. This feedback system is not only able to more accurately evaluate solution quality, but also to better guide the exploration of the search space.

The scientific literature has recently seen several successful applications of simheuristic algorithms to a broad variety of stochastic combinatorial optimization problems. In [82], a hybrid simheuristic is proposed to solve a variant of the capacitated VRP. This variant combines vehicle routing and packing constraints, and so it is known as the two-dimensional VRP with stochastic travel times. The multi-start randomized simheuristic, presented in [88] for the single-period and stochastic Inventory Routing Problem with stock-outs, shows how personalized refill policies can be crucial to enable significant cost reductions in comparison to what can be achieved by other algorithms employing standard refill policies.

[77] tackle the Arc Routing Problem (ARP) with stochastic demands by embedding Monte Carlo simulation into a classical strategy for the capacitated ARP. Without being exhaustive in our review, other notable efforts in the literature can be found in the area of Permutation Flow Shop Problems ([87]; [53]; [78]), and facility location problems [33].

4.3 Integrating a Biased Randomized GRASP with Monte Carlo Simulations

This section begins by discussing how a basic GRASP framework can be enhanced by making use of biased-randomization concepts (BR-GRASP). Subsequently, the BR-GRASP is hybridized with Monte Carlo simulation to properly address the stochastic component of the VRPSD.

GRASP is a well-known metaheuristic framework for hard combinatorial optimization problems. As discussed in Section 2.2, a classical GRASP is made up of two main components (Algorithm 1): a constructive phase and a local search. In the former, the element to be added to the solution under construction is selected uniformly at random among the elements of a *restricted candidate list* (RCL). The RCL collects all the best insertion candidates according to some greedy score function. It includes all the elements whose score is better than a user-specified percentage of the best score. This adaptive thresholding process is guided by a parameter, α, which tunes the greediness of the constructive phase. Our solution method relies on a recent extension of the classical GRASP paradigm: GRASP with biased randomization or BR-GRASP (Algorithm 2).

Input: $\alpha \in [0,1]$
$s \leftarrow \emptyset$
initialize candidate set: CL $\leftarrow$ E
order CL according to $c(\cdot)$
while *solution* s *is not complete* **do**
  $c_{min} \leftarrow \min_{x \in CL}\{c(x)\}$
  $c_{max} \leftarrow \max_{x \in CL}\{c(x)\}$
  $thr \leftarrow c_{min} + \alpha(c_{max} - c_{min})$
  $RCLsize \leftarrow |\{x \in CL: c(x) \leq thr\}|$
  $pos \leftarrow$ UnifRand$(1, 2, \ldots, RCLsize)$
  $s \leftarrow s \cup \{CL[pos]\}$
  $CL \leftarrow CL \setminus \{CL[pos]\}$
  Reorder CL
return s

Algorithm 1: Construction phase with RCL

Input: Distribution D; β
$s \leftarrow \emptyset$
initialize candidate set: CL $\leftarrow$ E
order CL according to $c(\cdot)$
while *solution* s *is not complete* **do**
  Randomly select $pos \in \{1, \ldots, |CL|\}$ according to distribution $D(\beta)$
  $s \leftarrow s \cup \{CL[pos]\}$
  $CL \leftarrow CL \setminus \{CL[pos]\}$
  Reorder CL
return s

Algorithm 2: Construction phase with Biased Randomization

The aim of the BR-GRASP is to guide the construction process towards more favorable regions of the search space, with the goal of achieving diversification without excluding any potentially good candidate elements. More specifically, while in the classical GRASP the RCL implementation is obtained by the two-step process described above (thresholding plus uniform selection), this new paradigm guides the constructive phase using a skewed (non-uniform) probability distribution and without using a RCL.

Different skewed probability distributions can be considered within this extension. As suggested in [80], we use a geometric distribution with parameter β. The result of this process is a constructive phase that is able to balance greedy bias and search diversification. A depiction of the differences among the selection processes of the classical GRASP and the BR-GRASP can be found in Figure 4.1.

The idea of using biased sampling in a heuristic algorithm was originally described in [19], and since then biased-randomization techniques have been successfully used to solve different VRP variants, including the two-dimensional loading VRP ([40]; [41]) or the multi-depot VRP [89].

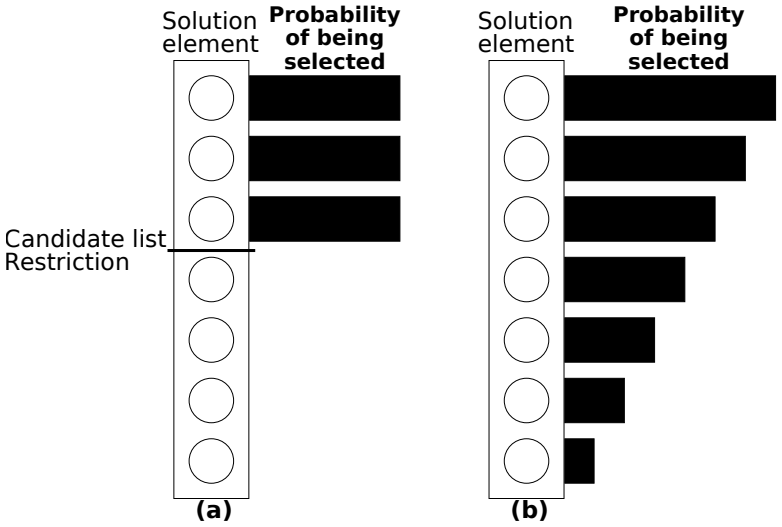

Fig. 4.1 Differences in the selection processes of the classic GRASP *(a)* and the BR-GRASP *(b)*.

Metaheuristic frameworks were originally conceived for tackling deterministic problems. Accordingly, the classic GRASP is not designed to consider uncertainty within the input data. From a technical perspective, this represents a major drawback when dealing with real-life scenarios, for which the use of a deterministic model could represent an oversimplification which leads to suboptimal solutions [100].

With the aim of allowing for the inherent uncertainty within logistics and transportation problems, our algorithm embraces the paradigm of simheuristics [79]. The resulting framework, the SimGRASP, incorporates Monte Carlo simulation at two different stages of the BR-GRASP approach. Incorporating simulation within an optimization algorithm is not on its own enough to guarantee a reliable solution. The performance of the stochastic solution obtained by the simheuristic is tightly linked to the quality of the deterministic optimization algorithm, that is, both components are complementary to the quality of the final solution. In this work we use BR-GRASP as the deterministic optimization component. The operation flow of our simulation-optimization technique is summarized in Figure 4.2.

As in the deterministic case, the algorithm begins with a biased-randomized constructive phase. The output is a first feasible solution s. This first phase is then followed by a local search, with the aim of improving s to achieve a deterministic local optimal solution, s^*, according to a two-opt exchange neighborhood structure $\mathrm{N}(\mathrm{x})$.

The deterministic local optimum s^* is then evaluated in a stochastic environment, which represents the first of two applications of Monte Carlo simulation, in which a limited number of simulation replications are used. Within each single run of the simulation, all customers' demands are randomly generated accordingly to the corresponding probability distributions. The number of runs performed in this first simulation stage is denoted $\mathrm{nIter}_{\mathrm{short}}$. This use

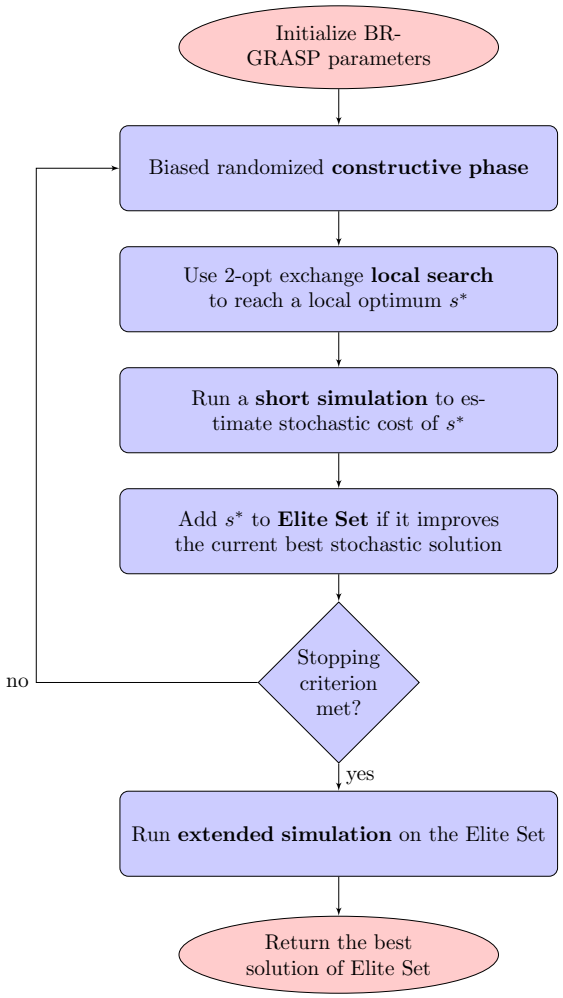

Fig. 4.2 Flowchart of our SimGRASP algorithm.

of a limited number of simulations runs provides a mechanism for controlling the trade-off between run times and accuracy. Subsequently, the initial solution s^* is set as the current best solution, and the 'elite set' (ES) is initialized with s^* as its only element.

Thereafter, as in the classical GRASP paradigm, the procedure is repeated in a multi-start fashion until the stopping criterion is satisfied. In each iteration, new deterministic solutions are generated and consequently improved by means of the local search, hence obtaining new deterministic local optima s^{**}. Each of the locally optimal solutions is evaluated in $\mathrm{nIter}_{\mathrm{short}}$ simulation runs. If in this stochastic analysis it is found that the performance of s^{**} is better than those obtained by the current best solution, then s^{**} is added to ES and the current best solution is accordingly replaced.

Once the multi-start process has finalized, all the solutions belonging to ES are evaluated through a second simulation stage, this time using a higher number of simulation runs,

$nIter_{long}$, for each evaluation of a candidate solution. This second simulation stage is only applied to a reduced set of 'promising' solutions, which limits the necessary computing time. The aim of this second simulation stage is to thoroughly assess, with a higher degree of accuracy, the stochastic quality of the solution generated during the heuristic search process.

4.4 Algorithmic Performances

In this section, a series of computational experiments are described and their results analyzed. These numerical experiments contribute to verify and validate the proposed simulation-optimization methodology.

4.4.1 Experimental Settings and Benchmarks

An assessment of the performance of our SimGRASP was obtained by comparison with the multi-start simheuristic discussed in [86]. In this paper, the authors use a multi-start algorithm that is later combined with simulation to estimate the expected distribution costs. Both the multi-start algorithm and our SimGRASP make use of a geometric probability distribution to benefit from biased-randomization techniques [80]. In addition, the multi-start algorithm (SimMultiStart) uses a splitting strategy, which tries to improve each solution by dividing it into smaller problems. The SimMultiStart approach considers the concept of safety stocks to prevent running out of goods due to the occurrence of higher-than-expected demands. Finally, the local search employed in the SimMultiStart algorithm implements a cache-memory to store past routes, while the local search in our BR-GRASP incorporates a 2-opt exchange operator. In the interest of a fair analysis, the simulation paradigm considered for the two algorithms follows the same construction mechanism. The demands d_i used in the deterministic benchmark set are replaced by their stochastic counterpart, D_i, according to a Log-Normal probability distribution, with $E[D_i] = d_i$. Also, $Var[D_i] = k \cdot d_i$, where $k > 0$ is a design parameter. The experiment results reported here were obtained with a variance level of $k = 0.25$. Consequently, the Log-Normal distribution can be described through the location (μ_i) and scale (σ_i) parameters as follows:

$$\mu_i = \ln(E[D_i]) - \frac{1}{2} \cdot \ln\left(1 + \frac{Var[D_i]}{E[D_i]^2}\right)$$
$$\sigma_i = \left|\sqrt{\ln\left(1 + \frac{Var[D_i]}{E[D_i]^2}\right)}\right|$$

In order to obtain a reliable evaluation, the method presented here was applied to the well-established benchmark instances of types A and B that were proposed in [4] for the capacitated VRP. The SimGRASP was implemented as a Java application and run directly on Eclipse using a personal computer with an Intel i7 Quad core, 2.67 GHz clock, and 6 GB RAM. After a quick fine-tuning of the BR-GRASP parameters, the following values were set: for the threshold parameter in Algorithm 1, $\alpha = 0.3$; for the Geometric distribution parameter in Algorithm 2, $\beta = 0.5$. The algorithm was executed with a time limit of 10 seconds.

4.4.2 Analysis of Results

Table 4.1 summarizes the results generated in the testing phase. The comparison of the solutions obtained by: *(i)* the coupling of a multi-start heuristic and simulation; and *(ii)* the SimGRASP approach shows how this new technique outperforms the multi-start one in each instance. The measured gap, indeed, is negative for all instances in the benchmark set, with an average value of -9.81% over all instances. These good results are mainly due to the optimization paradigm used in the implementation of our BR-SimGRASP, mainly due to the incorporation of an enhanced local search operator.

Furthermore, the results show that the performance of the stochastic solutions are enhanced by the biased-randomized construction with respect to the classical GRASP paradigm. Another example of the effect of the quality of the solution generated in the construction phase and stochastic value of the final solution can be found in the radar chart presented in Figure 4.3. This plot illustrates the solution values obtained, for instances A-n33-k5 and B-n31-k5, by three different algorithmic approaches: SimMultiStart, SimGRASP, and SimGRASP with BR. Thus, values closer to the center of the triangle represent a lower-cost value. Figure 4.3 shows how the biased-randomized SimGRASP achieves the best results among the three approaches.

Table 4.1 Performance of BR-GRASP and SimGRASP for the VRP.

Instance	SimMultiStart (1)	SimGRASP (2)	% Gap (1)-(2)
A-n32-k5	993.20	890.95	-10.30
A-n33-k5	815.40	750.63	-7.94
A-n33-k6	912.60	837.63	-8.21
A-n37-k5	795.00	734.44	-7.62
A-n38-k5	885.10	824.37	-6.86
A-n39-k6	1010.60	926.11	-8.36
A-n45-k6	1184.30	1091.36	-7.85
A-n45-k7	1502.00	1336.08	-11.05
A-n55-k9	1408.40	1258.72	-10.63
A-n60-k9	1795.70	1579.79	-12.02
A-n61-k9	1330.60	1224.44	-7.98
A-n63-k9	2203.70	1897.29	-13.90
A-n65-k9	1555.30	1437.84	-7.55
A-n80-k10	2328.40	2177.10	-6.50
B-n31-k5	855.70	757.60	-11.46
B-n35-k5	1255.50	1098.00	-12.54
B-n39-k5	695.90	621.34	-10.71
B-n41-k6	1103.20	1005.62	-8.84
B-n45-k5	904.60	828.04	-8.46
B-n50-k7	945.80	859.48	-9.13
B-n52-k7	944.40	848.71	-10.13
B-n56-k7	920.00	845.37	-8.11
B-n57-k9	2199.70	1885.69	-14.27
B-n64-k9	1179.60	1064.53	-9.75
B-n67-k10	1404.50	1247.67	-11.17
B-n68-k9	1754.70	1515.88	-13.61
Average			***-9.81***

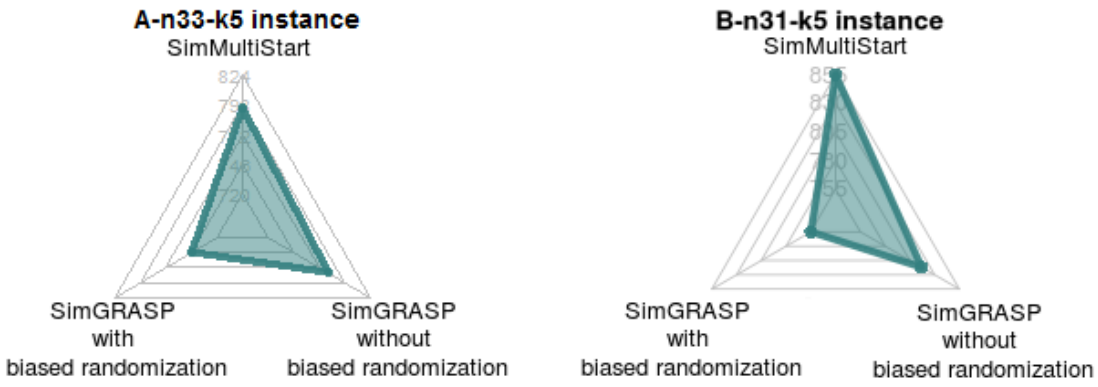

Fig. 4.3 Comparison of a MS simheuristic, SimGRASP with BR, and SimGRASP without BR.

4.5 The Shortest Path Problem: Classical Approaches

Shortest Path Problems can be classified in three different sub-categories: shortest path point-to-point (P2P), shortest path tree (SPTP), and all pairs shortest paths (APSPP). In the following we give the mathematical formulations of the classical P2P and SPTP, and list the most notable solution strategies devised in the past decades.

Let $G = (V, A)$ be a directed weighted graph, where:

- $V = \{1, 2, \ldots, n\}$ 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;
- $w\colon A \to \mathbb{R}^+$ is a function that assigns a non-negative cost w_{ij} to each arc $(i,j) \in A$.

Furthermore, for each $i = 1, \ldots, n$, 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.

The P2P Shortest Path Problem consists in finding a shortest path $P^* = (v_1, v_2, \ldots, v_h)$ 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:

$$\text{(P2P)} \qquad z = \min \sum_{(i,j)\in A} w_{ij} x_{ij}$$

subject to:

$$\text{(P2P-1)} \qquad \sum_{j\in BS(i)} x_{ji} - \sum_{j\in FS(i)} x_{ij} = b_i, \qquad \forall i \in V$$

$$\text{(P2P-2)} \qquad x_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A,$$

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

On the other hand, given an origin node $r \in V$, the aim of the SPTP is to find a shortest path from r to all the other nodes of V. The SPTP admits the following linear programming formulation:

$$\text{(SPTP)} \qquad \min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

subject to:

$$\sum_{(j,i)\in BS(i)} x_{ji} - \sum_{(i,j)\in FS(i)} x_{ij} = b_i, \quad \forall\, i \in V,$$

$$x_{ij} \geq 0, \qquad \forall\, (i,j) \in A,$$

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 $i \neq r$.

A first successful attempt to solve the SPP was originally proposed in [61, 62], although the most famous algorithm to solve P2P and SPTP is a labeling method proposed by Dijkstra [38].

In the case of P2P problem, *bidirectional versions* of Dijkstra's algorithm were proposed in [32, 42, 120]. 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. [83] 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.

4.6 Reoptimization

Nowadays, there is a rising need of well performing algorithms, able to handle the massive amount of available information. One of the suitable approaches to tackle this complexity is to reuse information already computed, in order to reduce the computational time needed to obtain an optimal solution. 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 source node has been changed;
- some nodes have been added or removed;
- some arcs have been added or removed;

- some arcs weight have been increased or decreased.

This problem can be addressed as a shortest path reoptimization problem [52, 55], which consists in solving a sequence of shortest path problems, where the k^{th} problem marginally differs from the $(\mathrm{k}-1)^{\mathrm{th}}$ one.

In the first case, we say that there was a root change from the $(\mathrm{k}-1)^{\mathrm{th}}$ 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.

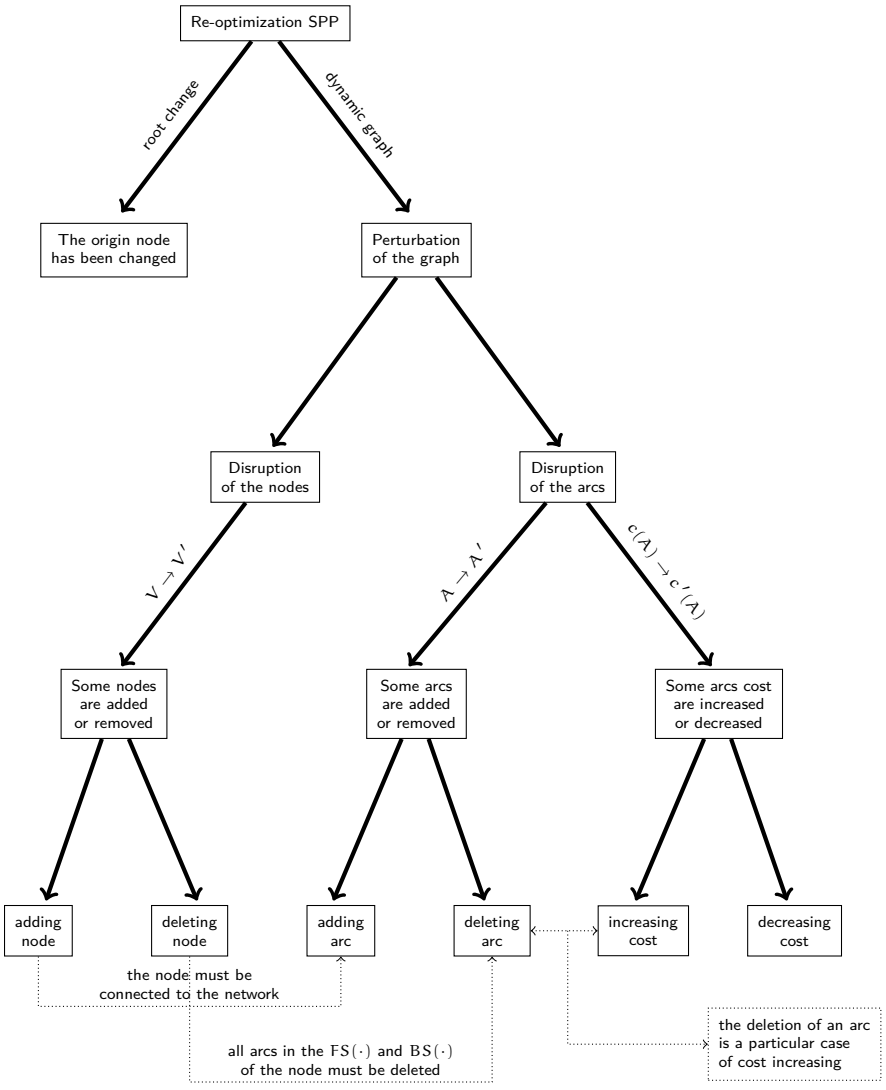

Fig. 4.4 Reoptimization problems hierarchy.

In Figure 4.4, we propose a diagram of the interplays among all possible cases of shortest path reoptimization.

4.6.1 Root change

The purpose of this paragraph is to briefly review the efforts devoted to the root change for the SPTP problem, showing how 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 [64], 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. Let s be a node of V, $s \neq r$, and T_s be a SPTP rooted at node s. Moreover, for all $u, v \in V$, denote with $d(u, v)$ their distance.

The following proposition shows 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 2. $T_r(s) \subseteq T_s$ *and* $d(s, j) = d(r, j) - d(r, s)$*, for any* $j \in T_r(s)$.

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. Indeed, the paths contained in the subtree of T_r rooted in s still remain optimal shortest paths from s to its descendants.

As formerly stated, beyond the theoretical insight given by Proposition 2, 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 [35, 36]. 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 weight among the arcs, $w_{max} = \max_{(i,j)\in A} w_{ij}$. Gallo [64] shows how theoretical properties of the shortest path trees make possible to reduce w_{max} without changing the sets of feasible and optimal solutions.

Deriving a cost reduction in a similar fashion, in 1982 Gallo and Pallottino [65] devised an algorithm which outperforms both the one proposed in [64] 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. Denoting as $p(v)$ the precedessor of v in the shortest path,

the nodes in NQ are those whose distance from the root node is equal to the one of their predecessor, i.e. $d(r,v)=d(r,p(v))$. This property ensures that all the $v \in NQ$ can be inserted straightaway in NP without further comparisons, thus speeding up the execution of the algorithm. In [65], it has been noted how in any reoptimization problem instance 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. [60]: 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 in [60] consists in the adaptation of the dual simplex method to compute the shortest paths from the new root s, obtaining an algorithm that 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 [64].

The experimental evaluations proposed in these works show how Gallo [64] is slightly better performing than Dijkstra's from scratch technique, meanwhile Gallo and Pallottino [65] further improves the previous results. Although, among all, the algorithm proposed by Florian et al. [60] appears to outperform the other competitive methods.

4.6.2 Arc Cost Change

Given a shortest paths tree, T_r, the problem of arc cost change consists in recomputing the optimal tree T_r when a new weight is assigned to one or more arcs. As extensively discussed in Gallo [64], the results of Theorem 1 can be used to derive algorithms that reoptimize the solution of a SPP after a change of the cost of a single arc.

Theorem 1. *Let T_r be a shortest paths tree, w'_{uv} be the new cost of the arc $(u,v) \in A$, and $S(u) = \{ v \in V \mid d(r,v) \leq d(r,u) \}$. Denoting with T'_r the new shortest paths tree, the following properties hold:*

(i) if $w'_{uv} < w_{uv}$ and $(u,v) \in T_r$, then

$$T_r(v) \subseteq T'_r(v),$$
$$j \in T_r(v) \Rightarrow d'(r,j) = d(r,j) - (w_{uv} - w'_{uv}),$$
$$j \in S(u) \Rightarrow d'(r,j) = d(r,j);$$

(ii) if $w'_{uv} < w_{uv}$ and $(u,v) \notin T_r$, then

$$T_r(v) \subseteq T'_r(v),$$
$$j \in S(u) \Rightarrow d'(r,j) = d(r,j);$$

(iii) if $w'_{uv} > w_{uv}$ *and* $(u,v) \in T_r$, *then*

$$j \notin S(u) \Rightarrow d'(r,j) = d(r,j);$$

(iv) if $w'_{uv} > w_{uv}$ *and* $(u,v) \notin T_r$, *then*

$$T_r(v) = T'_r(v),$$
$$d'(r,j) = d(r,j) \quad \forall j \in V.$$

In other words, a decrease in the cost of the arc (u,v) (cases (i) and (ii)) implies that the sub-tree $T_r(v)$ remains part of the optimal solution and the optimal distances of its nodes accordingly decreased where necessary. Case (iii) tackles the cost increase when (u,v) is part of the solution, stating that the optimal distances are preserved for each $j \notin S(u)$. Finally, in case (iv) the entire solution remains optimal.

Pallottino and Scutellà [123] assume that a shortest paths tree T_r has been determined and address the problem of computing the shortest paths tree when new costs are given to a subset K of the arcs of G, either lower or higher than the old ones. The reoptimization framework used is based on the determination of a suitable decomposition of the arc set K into disjoint subsets, and performs subsequent phases, where each phase reoptimizes with respect to the change of the arc costs of one subset of such decomposition. Pallottino and Scutellà [123] also compute the time complexity of the algorithm as a function of both the input size and the overall cost perturbations, without proposing an implementation.

One of the most important works in shortest path reoptimization in case of arc cost changes is Ramalingam and Reps [130], that describes the so called *DynamicSWSF-FP* algorithm. At any time, it is defined

$$rhs(v) = \min_{x \in BS(v)} \left\{ \hat{d}(r,x) + w'_{xv} \right\},$$

where $\hat{d}(r,x)$ is the distance of x from root r in the current intermediate tree $\hat{T}$. In the proposed algorithm, each node is processed differently according to whether $rhs(v)$ is greater than (*underconsistent node*), equal to (*consistent node*), or less than (*overconsistent node*) $\hat{d}(r,v)$. The algorithm processes inconsistent nodes, opportunely stored in a heap H, in a nondecreasing order of key values, where $key(v) = \min \left\{ \hat{d}(r,v), rhs(v) \right\}$. Let

q be the inconsistent node currently selected. If it is underconsistent, then the algorithm sets $\hat{d}(r, q) = +\infty$; otherwise, it sets $\hat{d}(r, q) = key(q)$, its forward star is relaxed, and q is removed from the heap. The algorithm terminates when the set of inconsistent nodes is empty.

Buriol et al. [23] empirically showed how the algorithm devised by Ramalingam and Reps outperforms an optimization from scratch by means of Dijkstra's algorithm. Furthermore, they proposed a new technique to improve computational times of the approach described above. This technique is called *Reduced-Heap*, because it reduces the number of nodes of Q (inconsistent nodes) to be inserted in the heap H.

While the technique proposed in [23] is able to handle the cost update of a single arc, Chan and Yang [28] discuss a comparison between several algorithms devised to handle multiple arc cost updates (batch updates). The first algorithm studied is a dynamic version of the classic Dijkstra's algorithm. Given the original shortest paths tree T_r, after the arc cost changes occur, all the updated arcs are removed from T_r, and the set $\bar{N}$, the set of nodes that are not reachable anymore from the root r, is computed. The nodes that have an arc connecting to a node in $V \setminus \bar{N}$ are added in an heap H and a Dijkstra-like algorithm is used to update labels.

The main accomplishment made by Chan and Yang [28] is the *MFP* algorithm. It is an extension of algorithm by Ramalingam and Reps [130] that can handle path optimization in fully dynamic networks. The original algorithm has been improved to avoid unnecessary rhs value recomputations and also simplifying computation whenever possible.

Other notable approaches able to reoptimize shortest paths in case of batch updates, or more generally solve SPPs on dynamic networks, are D'Andrea et al. [45], Nannicini et al. [118], and Tretyakov et al. [150].

4.7 Comparing Simheuristics and Reoptimization

Studying both simheuristics and reoptimization approaches one may be interested in which approach dominates the other. However, as often happens in Operations Research, it is not possible to identify any kind of dominance. Moreover, both methods propose a trade-off between computational expenses and solution flexibility, and the only question that can be asked is which one is more suitable for a given dynamic optimization problem.

The common thread of the reoptimization techniques discussed is their ability to generate a deterministic solution and to adjust it in real-time as uncertainties materialize. Such a strategy has clear advantages whenever the optimization scenario accounts for enough time

resources to solve the problem online, or equivalently in the case of tractable problems, as the SPP ([52], [55]).

However, for problems that are characterized by their hard complexity or whenever a reduced amount of computing resources are available to complete a reactive strategy, then allowing for stochasticity in the initial solution process is beneficial. In this last case, simheuristics can achieve a reliable outcome, without assuming that recourse actions are available to correct solutions during search process.

Chapter 5

Topology Optimization: a Hardly Constrained Design Problem

The Min-Max GDP problem studied in Chapter 3 belongs to that family of combinatorial optimization problems whose hardness lies in the solution evaluation rather than their feasibility. On the contrary, this chapter is devoted to the study of a design problem characterized by the complete opposite paradigm.

Indeed, in this study we focus on the optimization of stress-constrained concrete structures. In particular, this problem consists in the minimization of the volume (or mass) of the structure, while satisfying hard stress constraints related to material properties of concrete elements.

5.1 The Origins of Topology Optimization: the Compliance Problem

One of the most notable contributions in the field of shape optimization and structural design is due to Bendsøe and Kikuchi [8], whose work paved the way for modern topology optimization as we know it today. Indeed, in their study the authors considered an optimization paradigm to not only explore different shapes, but also to include changes in the connectivity of the structure, defining a distribution method based on the use of an artificial composite material with microscopic voids.

Since then topology optimization went a long way, not solely for the studies proposed in scientific literature, but to a great extent because of important advances in the field of additive manufacturing. See [9] for an extensive review on the subject.

As can be seen in the literature, the vast majority of topology optimization studies address the problem of compliance minimization. The use of compliance, as inverse of the structural stiffness, was the first attempt to to control the response of the structure and study occasional failures.

To formally model the problem, a Finite Element (FE) analysis is employed to compute the physical relations that rule the optimization scenario. Moreover, in order to describe the material domain, classical approaches couple a FE analysis with the *density-based model*, as done for example in [139].

Let M be a $n_x \times n_y$ mesh, such model defines decision variables x_e for each element e of the mesh, representing the density of e in the current design, thus requiring $0 \leq x_e \leq 1$ $\forall e$.

On the other hand, intermediate densities do not have a clear physical interpretation, so it is customary to use a penalization technique to obtain a solid-void solution. The most widespread penalization technique in the literature is the *SIMP* method (Solid Isotropic Material with Penalization) [7, 157, 2]. This method is based on a heuristic relation between x_e and the elemental Young's modulus E_e, given by:

$$E_e = E_e(x_e) = x_e^p E_0 \quad x_e \in (0, 1]\,, \tag{5.1}$$

where E_0 is the Young's modulus of the solid material and p is the penalization power (usually $p \geq 3$). Moreover, in order to avoid numerical singularities, a modified *SIMP* approach uses the adjusted relation:

$$E_e = E_e(x_e) = E_{min} + x_e^p\,(E_0 - E_{min}) \quad x_e \in (0, 1]\,, \tag{5.2}$$

with E_{min} being the non-zero elastic modulus of a fictitious void-like material.

Accordingly, the general compliance optimization problem reads as follows:

$$\min \sum_{e=1}^{N} (x_e^p)\, \mathbf{u}_e^T \mathbf{k}_e \mathbf{u}_e \tag{5.3}$$

s.t.

$$\mathbf{K}\mathbf{u} = \mathbf{f} \tag{5.4}$$

$$\frac{V(\mathbf{x})}{V_0} = \phi \tag{5.5}$$

$$\mathbf{x} \in [0, 1]^N\,, \tag{5.6}$$

where $\mathbf{x}$ is the vector of design variables representing the density of the elements of the mesh, x_e is the elemental density, and N indicates the number of elements. u_e and k_0 are, respectively, the element displacement vector and stiffness matrix, while, $\mathbf{K}$, $\mathbf{u}$ and $\mathbf{f}$ are the global stiffness matrix, the global displacement and force vectors, respectively. Finally, volume constraints are enforced by means of (5.5), where $V(\mathbf{x})$ is the volume of the designed solution, V_0 is the total volume of the physical domain, and ϕ is a prescribed volume fraction.

Problems described by means of (5.3) are considered relatively easy to solve in the landscape of topology optimization, since there exists several methods able to address them effectively, such as Optimality Criteria methods [139], Sequential Linear Programming, or the Method of Moving Asymptotes [146].

5.2 The Stress Constrained Problem: Models and Challenges

Even if compliance minimization problems gathered notable efforts in the past decades, as argued in [44], structural layout may be different for maximum stiffness (minimum compliance) and strength (e.g. with stress measurements) as soon as the material is characterized by a different behavior in tension and compression.

This difference between tensile and compression strengths is especially notable in the case of concrete-like materials, that therefore require a stress-constrained optimization model, rather than the solution of a classical compliance problem.

In the present study we address the problem of designing lightweight concrete elements under the restrictions of hard stress constraints. Moreover, as argued in the following, we will study a new form used in the imposition of the stress constraints and compare it with the classical Von Mises stress criterion.

Our problem of interest can be described by means of the following model:

$$\min \sum_{e=1}^{N} x_e \tag{5.7}$$

$$\text{s.t.}$$

$$\mathbf{K}\mathbf{u} = \mathbf{f} \tag{5.8}$$

$$\boldsymbol{\sigma}_- \leq \boldsymbol{\sigma} \leq \boldsymbol{\sigma}_+ \tag{5.9}$$

$$\mathbf{x} \in [0,1]^N \tag{5.10}$$

where the decision variables x_e and the physical quantities appearing in constraint (5.8) are as in (5.3). Concerning the stress constraint, $\boldsymbol{\sigma}$ is the stress vector, and $\boldsymbol{\sigma}_-$ and $\boldsymbol{\sigma}_+$, are, respectively, the vectors whose components are all equal to the maximum stresses allowed in compression (-) and traction (+).

It should be noted that the objective function in (5.7) aggregates the contributions of all the elements in the mesh, without taking into account their size. This means that the objective values of two different mesh representations of a same beam would not be directly comparable. For this reason, rather than (5.7), is more appropriate to rescale the objective function, ruling out the influence of the refinement of mesh. To do this, it suffices to consider an average of the densities x_e, thus giving

$$\min \frac{\sum_{e=1}^{N} x_e}{N} \tag{5.11}$$

$$\text{s.t.}$$

$$\mathbf{K}\mathbf{u} = \mathbf{f} \tag{5.12}$$

$$\boldsymbol{\sigma}_- \leq \boldsymbol{\sigma} \leq \boldsymbol{\sigma}_+ \tag{5.13}$$

$$\mathbf{x} \in [0,1]^N \tag{5.14}$$

In the following we will refer to the objective function in (5.11) as *average density*.

The main feature of our mathematical formulation consists in a new expression of the stress constraints. In literature, generally, the Von Mises criterion (σ_{vms}) is used to measure stress levels in the structure,

$$\sigma_{vms} = \sqrt{\sigma_1^2 - \sigma_1\sigma_2 + \sigma_2^2 + 3\tau_{12}}, \tag{5.15}$$

where σ_1 and σ_2 are respectively the $x-$ and $y-$ components of the stress tensor, and τ_{12} is the shear stress. Then, the associated constraint is expressed as follows,

$$\sigma_{vms} \leq \sigma_{lim}, \tag{5.16}$$

enforced for each node of the structure, where σ_{lim} is the maximum value allowed to the Von Mises stress.

This measure of the stresses appears to be not highly suitable for concrete design purposes, being more representative of an isotropic metal body, rather than a concrete structure. In Section 5.4.2, an experimental assessment of this claim is set out through an example of comparative testing.

Our new form for the stress constraints is based on principal stresses, decoupling stress limits, and effectively allowing the consideration of different material behavior in traction and compression, as concrete-based elements require. Consequently, the constraint and the material distribution law are modified and embedded in an iterative algorithm as follows.

During the generic iteration, a finite element analysis is performed and the stress tensor $\boldsymbol{\sigma}$ of the current structure is computed. At this point, rather than computing the Von Mises stress, we express the tensor $\boldsymbol{\sigma}$ in terms of principal stresses. After such computation, two separate bounds are forced on, respectively, the maximum and the minimum values of the principal stresses. In this way it is possible to control asymmetrically the stresses both in case of compression (bound on the minimum value) and traction (bound on the maximum value). Considering a 2D problem, we denote with σ_x e σ_y the two principal stresses in the x and y principal directions. The new constraints can be represented with the help of Figure 5.1, where for each element in the structure: with respect to the x direction, a value of $\sigma_x = 0$ indicates that the element is not stressed, a value of $\sigma_x < 0$ indicates that the element examined is compressed, while $\sigma_x > 0$ shows that the element is in traction. The same applies to σ_y.

$$\overline{\underbrace{\sigma_- \qquad\qquad 0}_{\text{Compression}} \underbrace{\qquad \sigma_+}_{\text{Traction}}}$$

Fig. 5.1 Representation of the stress constraints with asymmetrical traction (σ_+) and compression (σ_-) bounds.

Hence, for each integration point, and for each direction, we write the following inequalities:

$$\begin{cases} \sigma_- \leq \sigma_x \leq \sigma_+ \\ \sigma_- \leq \sigma_y \leq \sigma_+. \end{cases} \tag{5.17}$$

Moreover, in order to properly handle these constraints throughout the optimization process, we define the Risk Factors RF_1 and RF_2 for each principal stress direction:

$$\begin{cases} RF_1 = \max\left(\dfrac{\sigma_x}{\sigma_+}, \dfrac{\sigma_x}{\sigma_-}\right) \\ RF_2 = \max\left(\dfrac{\sigma_y}{\sigma_+}, \dfrac{\sigma_y}{\sigma_-}\right). \end{cases} \tag{5.18}$$

Mathematically speaking, these variables measure how much the values σ_x e σ_y computed are within the allowed interval $[\sigma_-, \sigma_+]$. Consequently, whenever $RF_i > 1$ it means that the

stress is outside the allowed interval, and the current solution is not feasible. Furthermore, as an added value, we observe how specific stress requirements can be taken into account with the use of the *risk factor* approach. While σ_+ and σ_- are strictly related to material properties, in some optimization scenarios can be indeed desirable to limit the stresses in a confidence interval which is indeed more conservative than the actual material limits. If this is the case, it can be defined a percentage of confidence (p_σ) of the material limits σ_+ and σ_-. Then, the *risk factor* allows a straightforward adaptation of our method substituting $RF_i \leq 1$ with the following constraint

$$RF_i \leq \frac{p_\sigma}{100}, \quad i = 1, 2. \tag{5.19}$$

5.3 An Iterative Heuristic Method

The optimization strategy on which our algorithm relies on is based on the technique originally proposed in [15], with the name of *Proportional Topology Optimization* (PTO). The algorithm described in [15] is a simple heuristic technique that finds a sub-optimal solution without the use of any gradient information; it follows the iterative two-step procedure outlined in Figure 5.2, integrating a FE analysis with a heuristic optimization phase.

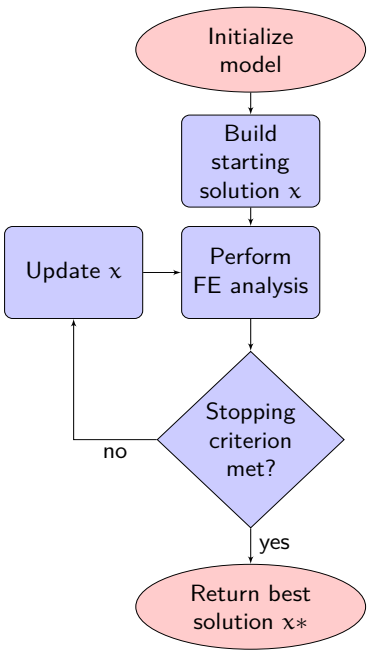

Fig. 5.2 General outline of the iterative topology optimization algorithm.

The PTO, starting from a first solution, iterates two distinct processes: the analysis of the Von Mises stresses and the update of the current solution. The stress analysis, and the study of the corresponding material constraints, are expressed by means of (5.15) and (5.16); while the solution is updated in a way such that overly-stressed elements are filled with more material. This proportional distribution strategy, carried over using the Von Mises stress criterion, is what gives the name to the algorithm presented in [15].

The algorithm that we describe and test in the present chapter, called Principal Stress Topology Optimizer (PS-TOpt), is obtained including in the PTO framework the asymmetric stress constraints, and making use of the Risk-Factor approach to determine the distribution of material in the beam.

After the initialization phase, a starting solution is built filling each element of the FE mesh with a constant amount of material, $\rho_{st} \in [0, 1]$. Often, in scientific literature, ρ_{st} has been set to a value of 0.5, see [15, 84] for example. In our analysis, we choose a higher starting value, namely $\rho_{st} = 0.9$. The reason for this choice is two-fold: firstly, a value of ρ_{st} close to 1 would almost certainly guarantee that the beam is feasible from the start, if the loads are not too heavy for the concrete class, so any possible stopping iteration of the algorithm would almost always yield a feasible –even if reasonably sub-optimal– solution. Secondly, we value the choice of a high ρ_{st} as a more natural optimization paradigm, in which the weight of the beam is decreased during the execution of the algorithm, meaning that the optimization procedure iteratively lighten the structure, yielding a sequence of improving beams.

After the construction of the first solution, a FE analysis is carried out to compute the stresses and the main loop begins, being executed for the whole duration of the algorithm. Each iteration starts with a solution update, and ends with a feasibility check, performed computing stresses and the subsequent Risk Factors. This two-phased iteration is based on a feedback process between structural optimization and stress computation, in which at each step, the stresses in the structure let the algorithm adjust the amount of material distributed in the physical body, whose feasibility is subsequently tested by means of the Risk Factor paradigm. More specifically, if the computations reveal that the structure is over-stressed, i.e., that the extremal measures of the stress among all elements of the mesh exceed the stress limits ($RF_i > 1$), then some material is added to the structure, otherwise, a lighter structure can be designed and some material is subtracted.

To do so, after each FE analysis, as in [15], the algorithm determines a target amount of material (`xTarget`) to be distributed in the beam,

$$\texttt{xTarget} = \begin{cases} \sum_{i}^{N} x_i + 0.001 \cdot N, & \text{if the structure is over-stressed} \\ \sum_{i}^{N} x_i - 0.001 \cdot N, & \text{otherwise,} \end{cases} \tag{5.20}$$

and, having set each x_i of the solution vector to zero, `xTarget` is distributed proportionally among the elements of the mesh. Such distribution is carried out in such a way that elements with higher stress value are filled with more material. This strategy is translated in the PS-TOpt algorithm in terms of principal stresses and Risk Factors. As stated above, with our new formulation of the stress constraints, the structure is considered to be over-stressed once there are elements such that at least one of their associated risk factors, RF_1 and RF_2, is greater than one. According to this new modeling paradigm, the material distribution for each element of the mesh has to be modified according the following update proportional factor:

$$pr_e = \frac{RF_1^q + RF_2^q}{\sum (RF_1^q + RF_2^q)}. \tag{5.21}$$

where q is the *proportion exponent*, that throughout the experimental phase has been set to a value of 3, as done in [15].

As reported in scientific literature, *checkerboard pattern* can arise as pathological behavior in topology optimization [141, 158, 140]. This phenomenon consists in the appearing of alternating areas with void and solid spaces in the discrete domain, disrupting consistency and continuity in the optimized beam design. In order to mitigate its occurrence, one of the most common approach consists in the usage of a *density filter*, which rescales the density of each element of the mesh with respect to the densities of its neighbors. The procedure adopted in this work is nothing but a weighted local average [21] given by the following equation:

$$\bar{x}_i = \frac{\sum_j w_{ij} x_j}{\sum_j w_{ij}} \tag{5.22}$$

where

$$w_{ij} = \begin{cases} \frac{r_{min} - r_{ij}}{r_0} & \text{when} \quad r_{ij} < r_{min} \\ 0 & \text{otherwise.} \end{cases} \tag{5.23}$$

Where $\bar{x}_i$ is the filtered density of the i^{th}-element and x_j is the density of the j^{th}-element before the filtering operation, r_{ij} is the Euclidean distance between the centers of elements i and j, and r_{min} is the filter radius.

Given the filtering process, it has to be noted that the actual amount of total material distributed among the elements, `xDist`, can be less than the `xTarget` computed. For this reason the remaining material, `xRemaining`, obtained as the difference between `xTarget` and `xDist`, is then iteratively distributed among the elements of the mesh, with the same proportional strategy and up to a certain threshold value.

As last step of each iteration, the stopping criteria is checked. If the criterion returns true, the simulation terminates, otherwise, the algorithm continues to optimize the topology of the structure. The stopping criteria let the algorithm stop if and only if both the two following occurrences take place: (i) a minimum number of iterations has been performed, (ii) the maximum of the risk factor RF_i is close to 1 within a certain value (in the present study the tolerance is set equal to 0.001). The first condition prevents premature convergence to poor quality solutions, and the second condition is related to the heuristic observation that the most light-weight structure is most likely obtained once that is no longer possible to subtract any amount of material due to the stress limitation, i.e. when the stresses cannot be pushed any further. As the algorithm terminates, the output solution is the lightest feasible design solution encountered during the search process. For the sake of clarity, a scheme of the described optimization strategy is reported in Algorithm 3.

As final methodological remark, we implemented an isoparametric bi-linear quadrilateral element for the FE analysis, adopting a Gauss-Legendre quadrature technique [76] for the computation of the elemental stiffness matrix. This FE scheme can handle any kind of rectangular geometry (beam-like objects). Element connectivity, nodes coordinates, boundary and load conditions are automatically defined through subroutine units.

5.4 Computational Results

This section is devoted to the experimental testing, executed to investigate the material properties of the optimized designs obtained.

Our analysis can be divided in three different tests: (i) the optimization of concrete beams belonging to different resistance classes, (ii) a comparative analysis of our solution with a beam designed using Von Mises stress constraints, and (iii) a final experiment that investigates the inclusion of a project-specific displacement constraint.

In the first experiment we take on the optimization of concrete beams belonging to the following classes: C20/25, C30/37, C40/50, and C60/75. For each of such concretes, we

Algorithm: PS-TOpt
Set up FE;
Set up filtering matrix W;
Initialize first solution x_{st};
Perform FE and stress analysis;
$It = 0$;
$x^* = x_{st}$; // Initialize best solution
while ($|\max RF - 1| \geq 0.001$ OR $It <$ MinIt) **do**
  if $\max RF > 1$ **then**
    xTarget$= \sum_i^N x_i + 0.001 \cdot N$;
  else
    xTarget$= \sum_i^N x_i - 0.001 \cdot N$;
  xRemaining $=$ xTarget;
  set $x_e = 0 \quad \forall e$;
  Compute pr_e;
  while (*xRemaining* > 0.001) **do**
    xDist(e) $=$ xRemaining$\cdot pr_e \quad \forall e$;
    $x_e = x_e + W \cdot$xDist(e);
    check on the bounds imposed on x and cut-off if necessary;
    xRemaining $=$ xTarget-$\sum_i^N x_i$;
  perform FE analysis;
  compute Risk Factors;
  if x *is feasible and improves* x^* **then**
    $x^* = x$;
  It++;
return best solution x^*;

Algorithm 3: Pseudo-code for the proposed PS-TOpt algorithm.

consider two distinct load scenarios, a first one in which the force is equal to a reference value fixed for all classes (11110 N), and a second load scenario that scales the forces according to the elasticity of the beam, assigning higher loads to concretes with higher strength. The aim of this first set of experiments, with its two load scenarios, consists in the investigation of the versatility of our method over an assorted set of material parameters.

The second set of experiments is focused on the comparison between our Risk-Factor approach and the classical Von Mises constraints. Its goal is to investigate the necessity of a method that is able to handle compression and tensile stresses of different magnitudes, effectively decoupling material properties, rather than grouping together stress requirements in a single expression as done in Von Mises-like approaches.

Lastly, we conclude our experimental analysis showing how our algorithm allows the inclusion of project-specific constraints. As an example we consider an additional restriction regarding the maximum displacement found in the optimized beam.

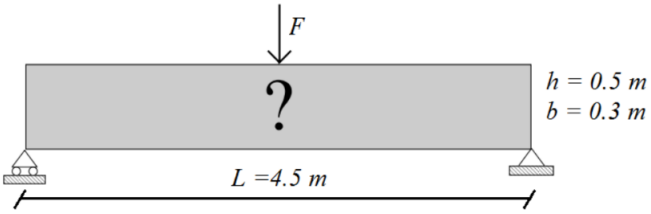

Fig. 5.3 Representation of the case study considered in the experimental phase.

The physical dimensions of the beam are summed up in Figure 5.3, while the mesh used has 900x200 elements. Given the symmetry of the physical problem, the loads are applied following the scheme depicted in Figure 5.4.

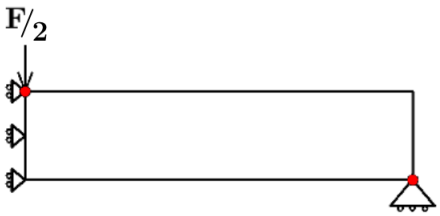

Fig. 5.4 Representation of the load scheme considered in the experimental phase.

All the computational experiments are conducted on a computer with a 2.6 GHz Intel Core i7 processor with 8 GB of RAM, and an Ubuntu 16.04 operating system.

5.4.1 Resistance Class Analysis

This first tranche of experiments is carried out to asses the computational performances of the algorithm over a diverse set of material parameters. In order to test the flexibility of our method, we divide this analysis in two different steps: a *fixed load analysis*, and a *proportional load analysis*. The goal of the former step, in which the same load (11110 N) is considered for each concrete class, is to test the capability of the algorithm to recognize distinct concrete elasticities, and to design different optimized solutions accordingly. In the latter step, the loads are scaled according to material parameters, and the aim consists in verifying the performances of the heuristic while handling a variety of load scenarios, ranging from 10555 N to 24444 N.

The material parameters considered in the *fixed load analysis* are summed up in Table 5.1, which lists, for each concrete class, the elastic modulus (E), compression (σ_-) and traction (σ_+) stress limits, and the load applied (F).

The results obtained are reported in Table 5.2, and displayed in Figures 5.5-5.8. More specifically, Table 5.2 lists, for each concrete class, the average density of the optimized

Class	E (Mpa)	σ_- (Mpa)	σ_+ (Mpa)	F (N)
C20/25	29960	-20	1.9	11110
C30/37	32840	-30	2.9	11110
C40/50	35220	-40	3.5	11110
C60/75	39100	-60	4.4	11110

Table 5.1 Material parameters considered in the fixed load experiment.

solution, the maximum stresses in compression and traction measured in the beam, respectively $\overline{\sigma}_-$ and $\overline{\sigma}_+$, and the maximum displacement detected, $\max\ U$. On the other hand, Figures 5.5-5.8 report the outlines of the optimized beams, and two plots, depicting the evolution of the average density (*D-plot*) and the maximum of the Risk Factor (*RF-plot*) through the execution of the algorithm.

Observing the *RF-plots* of Figures 5.5-5.8, we can see how the maximum Risk Factor increases when the amount of material used in the beam decreases, as expected, meaning that as the structure becomes lighter, the load imposed causes higher stresses. Moreover, in each one of the *RF-plots*, we find that below a certain average filling value, instability spikes occur in the Risk Factor, which locally passes the limit value of 1. Taking as example the *RF-plot* in Figure 5.5, we can see how such spikes can be observed starting from iteration 620, when the average filling drops below 30%. These instability spikes indicates that the material flexibility has been considerably exploited, and that the design of a sensibly lighter structure can be an unfeasible task. This algorithmic behavior means that the technique itself, by means of the Risk Factor, is able to detect whether the execution is getting close to the elastic limits of the material. This reasoning can serve as a retrospective justification of the stopping rule used, meaning that it is reasonable to stop the algorithm once that the execution gets closer to feasibility limit values.

Moreover, observing the shapes presented in Figures 5.5-5.8, and the average filling values in Table 5.2, we can conclude that the heuristic is able to not only design different beam outlines for distinct concrete classes, but also successfully supply lighter designs when the concrete class so permits, properly exploiting material elasticity.

Class	Avg filling %	$\overline{\sigma}_-$ (Mpa)	$\overline{\sigma}_+$ (Mpa)	$\max\ U$ (mm)
C20/25	25.6	-10.755	1.649	-0.5415
C30/37	14.9	-12.921	2.755	-0.9539
C40/50	14.0	-13.784	3.421	-0.9719
C60/75	13.8	-15.641	4.376	-1.0088

Table 5.2 Solution properties obtained in the fixed load experiment.

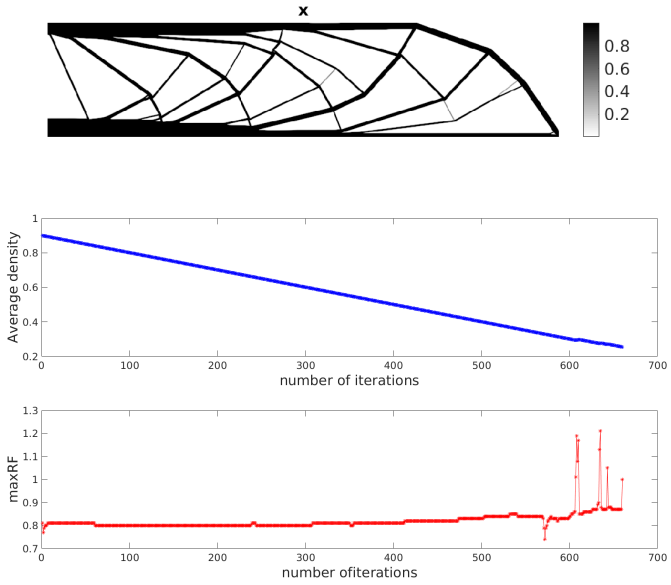

Fig. 5.5 Fixed load analysis: Class 20/25.

The second step of the first experimental phase is the *proportional load analysis*. In these tests the load applied is scaled according to the material properties of the concrete class, with the aim of investigating the algorithmic capabilities of handling different loads and elasticities. For each class, the value of the force F is obtained starting from the classic Navier's formula for the stress computation in a beam under simple bending, (5.24).

$$\sigma_{max} = \frac{M_x}{I_x}\frac{h}{2} \tag{5.24}$$

Considering a simply supported beam with a rectangular cross section under a concentrated load, the moment at the mid-span is $M_x = F \cdot L/4$ (where L is the length of the beam), hence equation (5.24) becomes:

$$\sigma_{max} = \frac{3FL}{2bh^2} \Rightarrow F = \frac{2\sigma_{max} bh^2}{3L} \tag{5.25}$$

In order to leave some room for the optimization algorithm to subtract material, a suitable value for the load F is obtained, for each concrete class, setting an appropriate amount for the stress factor in (5.25). More specifically, for each class, in (5.25) we impose $\sigma_{max} = \sigma_+/2$, thus obtaining the load values reported in Table 5.3.

The results obtained in the *proportional load analysis*, Table 5.4, Figures 5.9-5.12, show how the algorithm is able to optimize beams loaded with a diverse set of forces, suitably adapting solution design to material properties. Furthermore, comparing the *RF-plots* of Figures 5.10-5.12 with the ones presented in 5.6-5.8, we can see how the increase in the

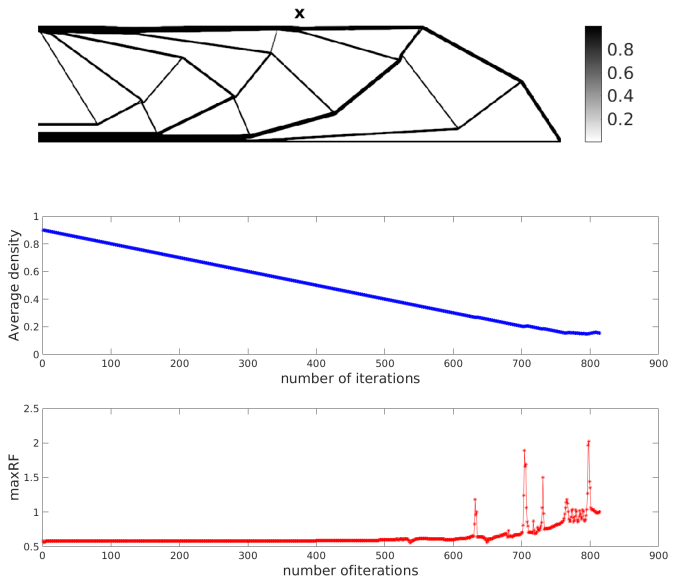

Fig. 5.6 Fixed load analysis: Class 30/37.

Class	E (Mpa)	σ_- (Mpa)	σ_+ (Mpa)	F (N)
C20/25	29960	-20	1.9	10555
C30/37	32840	-30	2.9	16111
C40/50	35220	-40	3.5	19444
C60/75	39100	-60	4.4	24444

Table 5.3 Material parameters considered in the proportional load experiment.

loads is rightfully reflected upon the Risk Factor value, that in the case of Figures 5.10-5.12 immediately starts closer to 1. As an extreme example of the algorithmic capabilities of handling high loads, we observe how in Figure 5.12, throughout the whole execution, the maximum Risk Factor is equal to the limit value 1, meaning that the load of 24444 N intensely stresses the beam. As concluding remark, we observe how this circumstance is handled by the heuristic, that in the optimized design properly fills up 55.4% of the total physical domain.

Class	Avg filling %	$\overline{\sigma}_-$ (Mpa)	$\overline{\sigma}_+$ (Mpa)	max U (mm)
C20/25	23.1	-10.255	1.826	-0.5659
C30/37	23.5	-17.220	2.859	-0.8460
C40/50	25.6	-20.000	3.497	-0.9441
C60/75	55.4	-20.000	4.398	-0.6581

Table 5.4 Solution properties obtained in the proportional load experiment.

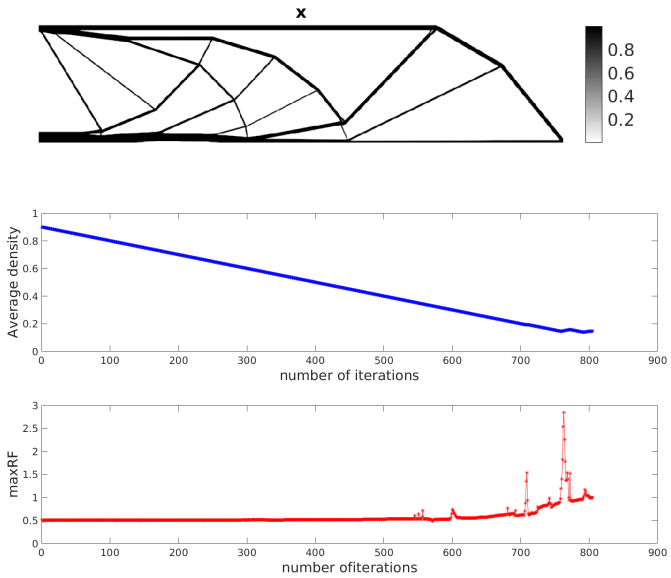

Fig. 5.7 Fixed load analysis: Class 40/50.

5.4.2 Comparison with Von Mises' Constraints

In the second part of our computational experiments, we take on the comparison of our Risk Factor approach with the classical Von Mises' stress constraint. Our goal is to show how, while working with strongly anisotropic materials such as concrete, it is highly advisable to decouple compression and traction stresses, rather than express structural constraints by means of a merged estimate such as the Von Mises criterion.

The experiment is structured as follows: to properly carry out a comparison, we start from a set of reference material parameters and a load used in the first phase, namely the values considered for class C20/25 in the *fixed load experiment*, and select three different representative limits for the maximum Von Mises Stress to be used. Then, we run the algorithm one time for each of such limits, and as in [15], distribute material in the beam according to the measured Von Mises Stresses in the structure, rather than using our Risk Factor approach. The aim of this comparison is to show how a Von Mises constraint is not suited to properly describe tensions and prevent failure in concrete based structures.

The reference principal stress limits and the load considered for class C20/25, as reported in Table 5.1, are, respectively: $\sigma_- = 20$ Mpa, $\sigma_+ = 1.9$ Mpa, and $F = 11110$ N. These material parameters led to the reference optimized beam in Figure 5.5.

The three representative limit values for the Von Mises stress, σ'_{vms}, σ''_{vms}, and σ'''_{vms} , are obtained as follows.

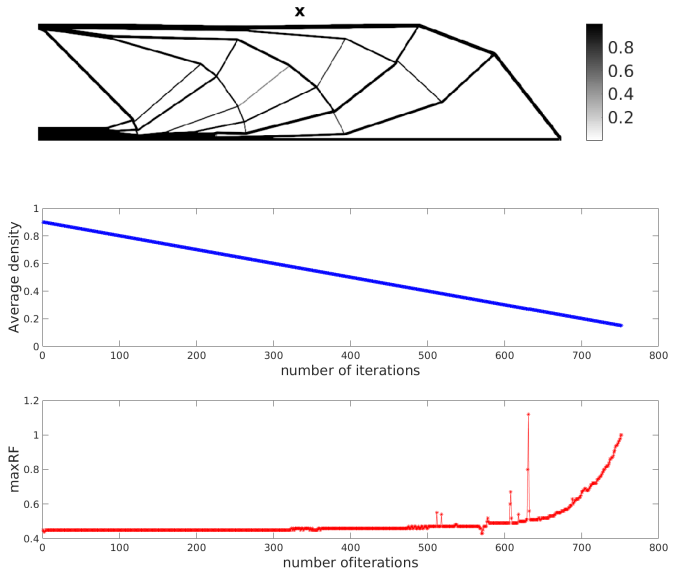

Fig. 5.8 Fixed load analysis: Class 60/75.

The value for σ'_{vms} is set to the absolute value of the maximum compression stress allowed for class C20/25,

$$\sigma'_{vms} = 20 \text{ Mpa}. \tag{5.26}$$

σ''_{vms} is obtained considering the formula:

$$\sigma_{vms} = \sqrt{\sigma_1^2 - \sigma_1\sigma_2 + \sigma_2^2 + 3\tau_{12}} \tag{5.27}$$

and assigning to σ_1, σ_2, and τ_{12} their maximum absolute values found in the reference beam (Ω). These three maximum stresses are, respectively,

$$\max_{\Omega} |\sigma_1| = 6.2842 \text{ Mpa};$$
$$\max_{\Omega} |\sigma_2| = 8.0070 \text{ Mpa};$$
$$\max_{\Omega} |\tau_{12}| = 3.9267 \text{ Mpa}.$$

Substituting these values in (5.27), we obtain $\sigma''_{vms} = 10.01$ Mpa.

As last representative value for the Von Mises limit, we choose the maximum Von Mises stress measured in the reference beam, thus

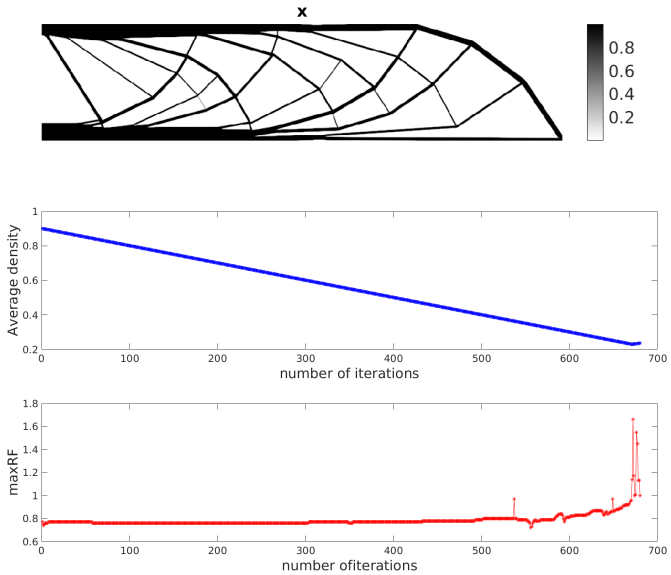

Fig. 5.9 Proportional load analysis: Class 20/25.

$$\max_{\Omega} \sigma_{vms} = \sigma'''_{vms} = 9.75 \text{ Mpa}. \tag{5.28}$$

The results obtained in this experiments are outlined in Table 5.5, while Figures 5.13-5.15 depict the optimized beam with its Von Mises stress distribution.

Analyzing the outputs of the Von Mises optimization, we can see how σ'_{vms} is not a proper choice for the limit value, since the resulting beam appears to be excessively thin, severely violating the traction limit of the material (10.60 Mpa $>>$ 1.9 Mpa), presenting, at the same time, an unreasonable maximum displacement (-14.04 mm). On the other hand, while cases σ''_{vms} and σ'''_{vms} distribute a comparable amount of material with respect to results found in Table 5.2 for class C20/25, once again we can see how the Von Mises constraint alone is not able to ensure the respect of the specific traction requirement of concrete-based materials, being both 2.94 Mpa and 2.14 Mpa greater than the required limit of 1.9 Mpa.

C20/25, $\sigma_- = -20$ Mpa, $\sigma_+ = 1.9$ Mpa, F = 11110 N

Case	Avg filling %	$\overline{\sigma}_-$ (Mpa)	$\overline{\sigma}_+$ (Mpa)	max U (mm)
σ'_{vms}	3.7	-20.00	**10.60**	-14.04
σ''_{vms}	18.2	-11.27	**2.94**	-0.75
σ'''_{vms}	25.6	-10.90	**2.14**	-0.66

Table 5.5 Solution properties obtained in the VMS comparison.

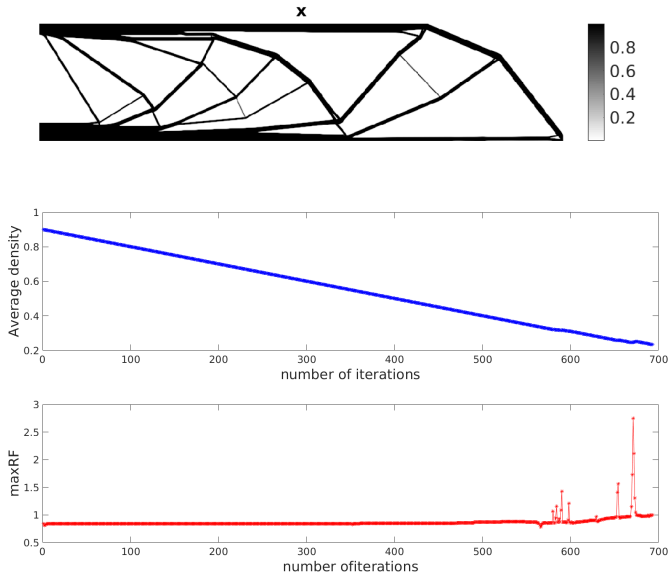

Fig. 5.10 Proportional load analysis: Class 30/37.

5.4.3 An Example of Project Constraint: Maximum Displacement

As final step of our computational experiments, we show how our algorithm is able to easily include other project-specific constraints. As an example of such limitations, we take into account a restriction over the maximum displacement of the beam. Considering the results obtained in the *fixed load analysis* for class C40/50 (Table 5.2), we can see how the optimized beam present a maximum displacement of -0.9719 mm. Let's assume that our project requires a maximum displacement not greater than 0.1% of our beam height. With respect to the dimensions depicted in Figure 5.3, this yields a maximum displacement whose absolute value is not greater than 0.5 mm. The technique described in Section 5.3 admits a straightforward adaptation to take into account this new constraint. More specifically, we let the algorithm distribute material not only when the principal stresses do not respect traction or compression limits, but also whenever the maximum displacement allowed is exceeded. Furthermore, the stopping rule is modified accordingly, letting the algorithm stop whenever, after the minimum number of iteration required, either the Risk Factor or the maximum displacement, are within a predefined threshold from their limit values.

The inclusion of this new constraint in our algorithm yielded a new beam outline, depicted in Figure 5.16, that is able to respect the project-specific constraint, strategically adding a small amount of material in the beam, as can be observed in Table 5.6.

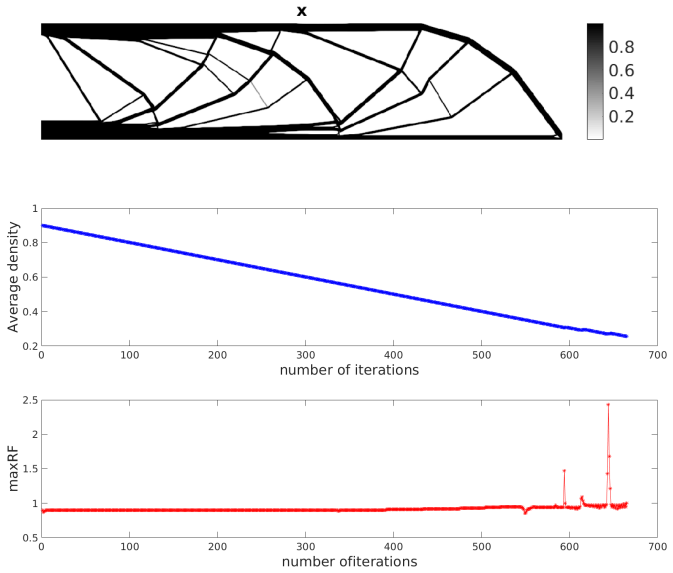

Fig. 5.11 Proportional load analysis: Class 40/50.

C40/50, $\sigma_- = -40$ Mpa, $\sigma_+ = 3.5$ Mpa, F = 11110 N, $|U| \leq 0.5$ mm

Case	Avg filling %	$\overline{\sigma}_-$ (Mpa)	$\overline{\sigma}_+$ (Mpa)	max U (mm)
w/o	25.6	-20.000	3.497	**-0.9441**
w/	27.9	-12.53	1.93	-0.4996

Table 5.6 Comparison of solution properties obtained in the experiments with (w/) and without (w/o) displacement constraint.

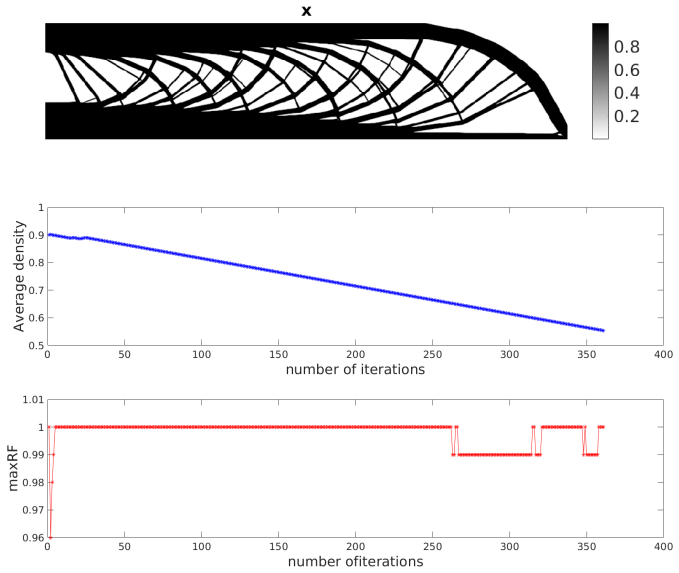

Fig. 5.12 Proportional load analysis: Class 60/75.

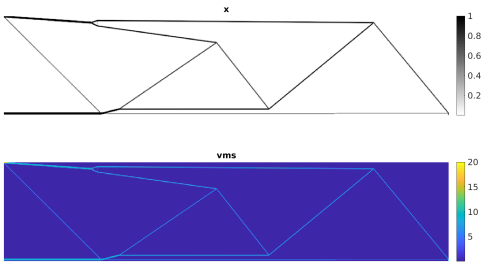

Fig. 5.13 VMS comparison: case σ'_{vms}

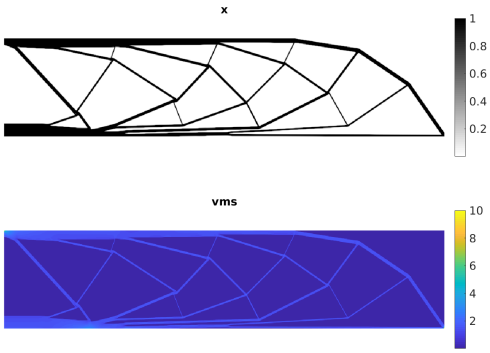

Fig. 5.14 VMS comparison: case σ''_{vms}

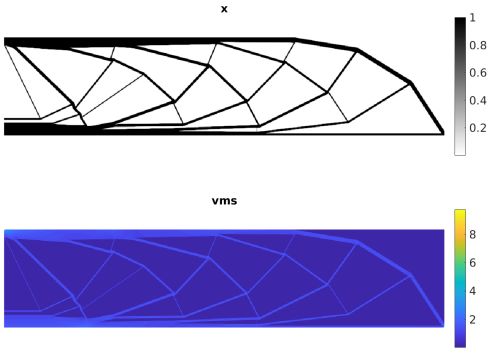

Fig. 5.15 VMS comparison: case σ'''_{vms}

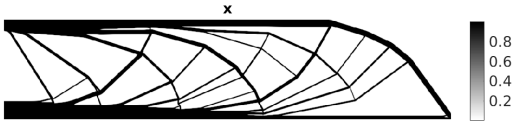

Fig. 5.16 Beam optimized with project-specific displacement constraint.

Chapter 6

Conclusions and Future Perspectives

In this thesis we investigated how metaheuristics can effectively address two key problems of data processing, such as two different instances of data extraction problems and Graph Drawing. Moreover, we investigated some solution opportunities regarding problems arising in engineering or management science, such as Topology Optimization or the Vehicle Routing Problem with Stochastic Demands.

All the algorithms here presented were tested competitively and evidenced soundness and efficiency, with performances able to outperform other state-of-the-art techniques or commercial solvers.

In the following paragraphs we briefly sum-up individual conclusions and future investigation opportunities that can be pursued for each one of the problems here discussed.

6.1 Minimum Cost SAT

Section 2.2 describes a GRASP metaheuristic to solve Minimum Cost SAT instances arising in the context of Supervised Learning in Boolean Fields. The algorithm shows excellent performances in the solution of instances of large size, both in terms of objective function value and computational expenses. Moreover, the probabilistic stopping rule studied provides a notable reduction in terms of number of iterations without jeopardizing the quality of the solution produced.

Aside from the solution of Minimum Cost SAT instances, the future research directions that we will consider are twofold. On one hand the goal is to test the capability of our GRASP in a learning framework, comparing its accuracy with other notable learning techniques such as Random Forests. At the same time an independent research stream can lead to the refinement of the dynamic estimate of the probability, extending its application range to several multistart heuristic algorithms.

6.2 Maximum Cut-Clique

In Section 2.5, we investigate a hybrid meta-heuristic based on the integration of a GRASP framework and a Phased Local Search. The method evidenced good performances in comparison with other two state-of-the-art heuristic approaches based on an ILS technique.

Further investigations will include the solution of specific case study instances arising from real world networks, and a study of the contributions that a probabilistic stopping rule –as in Section 2.3– can bring to the performances of our multistart metaheuristic.

6.3 Min-Max GDP

In Section 3.1 we target a hard graph drawing variant recently proposed: minimizing the maximum number of edge crossings in hierarchical graphs. Our tabu search method implements two memory structures, short term and long term, for an efficient exploration based on the search history. Our experimentation shows that they are indeed very effective compared with the randomized design of a previous heuristic. Special mention deserves the use of an auxiliary evaluation function to guide the search in such a flat landscape as the one that characterizes the min-max crossing problem. The comparison with two previous heuristics, whose moves are based only on the objective function, permits to conclude the remarkable performance of our tabu search method. The experimentation also reveals that the mathematical formulation is only able to solve instances with low density and very small size, and, as expected, requires long running times.

The highly attractive results provided by the δ evaluation function suggest an investigation to address other notable min-max problems with a similar approach. It is of great interest, indeed, a study on how the coefficients of δ can be set in relation to the topological characteristics of the graph, and dynamically adjusted according to informations gathered in the search process.

6.4 Constrained Incremental GDP

As described in Section 3.7, we have developed a heuristic procedure based on the GRASP methodology to provide high quality solutions to the problem of minimizing straight-line crossings in hierarchical graphs with an additional constraint. This problem is known as incremental graph drawing and the additional constraint models the stability on a sequence of drawings (the so-called user's mental map) when some vertices and edges are added by means of a parameter K. Our method is coupled with a Path Relinking post-processing

to obtain improved solutions in the long term. We also tested a tabu search procedure to evaluate the contribution of memory structures in comparison with semi-random designs. Exhaustive experimentation first discloses the best configuration of our methods and then performs an empirical comparison with the existing ones, namely the general purpose solvers `CPLEX` and `LocalSolver`. Our `GRASP` and `TS` implementations were shown to be competitive in a set of problem instances for which the optimal solutions are known, and clearly outperform `LocalSolver`. Finally, as revealed on large instances, the larger the parameter K the lower the number of crossings. However, this improvement is just marginal and therefore low values of K (close to 1) are recommended to obtain good stable graphs.

Given the importance of both aesthetic criteria considered in Chapter 3, in further investigations we will focus on the Constrained Incremental problem considering a bi-objective function, in the attempt of reducing both the total sum of crossings, and the maximum crossing among edges, as done in the case of the Min-Max GDP.

6.5 Vehicle Routing Problem with Stochastic Demands

The work presented in Section 4.2 is motivated by the benefits of accounting for stochasticity when solving hard combinatorial optimization problems. In our view, successfully incorporating stochasticity within deterministic optimization approaches depends on a careful integration of well-performing optimization tools in industrial settings. More specifically, we have proposed the use of an approach that combines simulation and metaheuristic optimization to solve one of the most difficult and widely studied stochastic optimization problems: the Vehicle Routing Problem with Stochastic Demands.

The proposed simheuristic is based on a recent extension of the classical GRASP framework, the GRASP with biased randomization. This extension has been shown to be a successful meeting point between performance and simplicity of implementation. The algorithm has been tested over a set of stochastic instances obtained from an established benchmark set for the deterministic version of the problem. The results show that our method is able to attain high-quality solutions over all instances in reasonably low computing times. In particular, our results support the idea that high-quality deterministic local optimal solutions are complementary to the quality of the robust solutions that are derived from the proposed hybridization of BR-GRASP and Monte Carlo Simulation.

Future extensions will follow two different research paths: a first stream of investigation can be based on improving implementation features that enhance the performance of the heuristic core of our algorithm. Path relinking, for example, is a feature that has been successfully embedded in multiple GRASP frameworks ([94]; [131]; [54]). On the other

hand, to improve the quality of the solutions generated specifically for the vehicle routing problem, some problem-specific intensification-diversification strategies can be devised, such as the granular neighborhood structure as applied in [148].

6.6 Topology Optimization of Stress-Constrained Structures

Chapter 5 studies the problem of designing lightweight concrete structures in the case of hard stress constraints. An appropriate handling of this optimization scenario is of great interest because of significant breakthroughs that additive manufacturing brought in the field of structural engineering; moreover, given these technological developments, an early inclusion of optimization in the design process could lead to structures with unconventional and high-performing designs.

More specifically, Section 5.2 introduces a new form for the stress constraint that proves to be more suitable than the classical Von Mises Criterion to handle concrete-like materials. Furthermore, the heuristic algorithm described in Section 5.3 is able to handle a diverse set of concrete classes and load scenarios, properly exploiting material elasticity and designing lightweight concrete beams accordingly.

Future researches will both address other part of the beam production problem, such as constrained shortest paths for 3D-printability, as well as consider more composed materials, including a mathematical modeling of steel reinforcing bars in the concrete structure.

References

[1] R. M. Aiex, M. G. C. Resende, and C. C. Ribeiro. Ttt plots: a perl program to create time-to-target plots. *Optimization Letters*, 1(4):355–366, Sep 2007.

[2] E. Andreassen, A. Clausen, M. Schevenels, B. S. Lazarov, and O. Sigmund. Efficient topology optimization in matlab using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, 2011.

[3] 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.

[4] P. Augerat, J. Belenguer, E. Benavent, A. Corberán, D. Naddef, and G. Rinaldi. Computational results with a branch-and-cut code for the capacitated vehicle routing problem. Technical Report R.495, IASI-CNR, 1998.

[5] O. Bastert and C. Matuszewski. *Layered Drawings of Digraphs*, pages 87–120. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[6] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1999. ISBN 0133016153.

[7] M. P. Bendsøe. Optimal shape design as a material distribution problem. *Structural and multidisciplinary optimization*, 1(4):193–202, 1989.

[8] M. P. Bendsøe and N. Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computer methods in applied mechanics and engineering*, 71(2):197–224, 1988.

[9] M. P. Bendsøe and O. Sigmund. *Topology optimization: theory, methods, and applications*. Springer Science & Business Media, 2013.

[10] P. Bertolazzi, G. Felici, and E. Weitschek. Learning to classify species with barcodes. *BMC bioinformatics*, 10(14):1, 2009.

[11] 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.

[12] D. J. Bertsimas. A vehicle routing problem with stochastic demand. *Operations Research*, 40(3):574–585, 1992.

[13] S. Bhatt and F. T. Leighton. A framework for solving vlsi graph layout problems. *Journal of Computer and System Sciences*, 28:300–343, 1984.

[14] L. Bianchi, M. Birattari, M. Chiarandini, M. Manfrin, M. Mastrolilli, L. Paquete, O. Rossi-Doria, and T. Schiavinotto. Hybrid metaheuristics for the vehicle routing problem with stochastic demands. *Journal of Mathematical Modelling and Algorithms*, 5(1):91–110, 2006.

[15] E. Biyikli and A. C. To. Proportional topology optimization: A new non-sensitivity method for solving stress constrained and minimum compliance problems and its implementation in matlab. *PloS one*, 10(12):e0145041, 2015.

[16] K.-F. Böhringer and F. N. Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, pages 43–51, New York, NY, USA, 1990. ACM.

[17] 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.

[18] J. Branke. *Dynamic Graph Drawing*, pages 228–246. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[19] J. L. Bresina. Heuristic-biased stochastic sampling. In *AAAI/IAAI, Vol. 1*, pages 271–278, 1996.

[20] S. Bridgeman and R. Tamassia. A user study in similarity measures for graph drawing. *J. Graph Algorithms Appl.*, 6(3):225–254, 2002.

[21] T. E. Bruns and D. A. Tortorelli. Topology optimization of non-linear elastic structures and compliant mechanisms. *Computer Methods in Applied Mechanics and Engineering*, 190(26):3443–3459, 2001.

[22] M. Burch, C. Müller, G. Reina, H. Schmauder, M. Greis, and D. Weiskopf. Visualizing Dynamic Call Graphs. In M. Goesele, T. Grosch, H. Theisel, K. Toennies, and B. Preim, editors, *Vision, Modeling and Visualization*. The Eurographics Association, 2012.

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

[24] S. Butenko and W. E. Wilhelm. Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research*, 173(1):1–17, 2006.

[25] M.-J. Carpano. Automatic display of hierarchized graphs for computer-aided decision analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(11):705–715, 1980.

[26] R. Carrasco, A. Pham, M. Gallego, F. Gortázar, R. Martí, and A. Duarte. Tabu search for the max-mean dispersion problem. *Knowledge based systems*, 85:256–264.

[27] L. D. Cecco, M. Giannoccaro, E. Marchesi, P. Bossi, F. Favales, L. Locati, L. Licitra, S. Pilotti, and S. Canevari. Integrative mirna-gene expression analysis enables refinement of associated biology and prediction of response to cetuximab in head and neck squamous cell cancer. *Genes*, 8(1):35, 2017.

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

[29] J. Chen and I. T. Chau. The hierarchical dependence diagram: improving design for reuse in object-oriented software development. In *Proceedings of 1996 Australian Software Engineering Conference*, pages 155–166, Jul 1996.

[30] M. Chimani, C. Gutwenger, P. Mutzel, M. Spönemann, and H. Wong. Crossing minimization and layouts of directed hypergraphs with port constraints. *Lecture Notes in Computer Science 6502 LNCS*, pages 141–152, 2011.

[31] C. H. Christiansen and J. Lysgaard. A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research Letters*, 35(6): 773–781, 2007.

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

[33] J. de Armas, A. A. Juan, J. M. Marquès, and J. P. Pedroso. Solving the deterministic and stochastic uncapacitated facility location problem: from a heuristic to a simheuristic. *Journal of the Operational Research Society*, 68(10):1161–1176, 2017.

[34] L. De Moura and N. Bjørner. Z3: An efficient smt solver. pages 337–340. Springer, 2008.

[35] 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.

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

[37] S. Diehl and C. Görg. Graphs, they are changing. In *10th International Symposium on Graph Drawing GD 2002*, page 23–30. Springer, 2002.

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

[39] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[40] O. Dominguez, A. A. Juan, and J. Faulin. A biased-randomized algorithm for the two-dimensional vehicle routing problem with and without item rotations. *International Transactions in Operational Research*, 21(3):375–398, 2014.

[41] O. Dominguez, A. A. Juan, B. Barrios, J. Faulin, and A. Agustin. Using biased randomization for solving the two-dimensional loading vehicle routing problem with heterogeneous fleet. *Annals of Operations Research*, 236(2):383–404, 2016.

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

[43] P. Duysinx and M. P. Bendsøe. Topology optimization of continuum structures with local stress constraints. *International journal for numerical methods in engineering*, 43(8):1453–1478, 1998.

[44] P. Duysinx, L. Van Miegroet, E. Lemaire, O. Brüls, and M. Bruyneel. Topology and generalized shape optimization: Why stress constraints are so important? *International Journal for Simulation and Multidisciplinary Design Optimization*, 2(4): 253–258, 2008.

[45] 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.

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

[47] L. F. Escudero, J. F. Monge, and D. R. Morales. On the time-consistent stochastic dominance risk averse measure for tactical supply chain planning under uncertainty. *Computers & Operations Research*, 100:270–286, 2018.

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

[49] 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.

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

[51] M. Fernández-Ropero, R. Pérez-Castillo, and M. Piattini. Graph-based business process model refactoring. In *SIMPDA*, pages 16–30, 2013.

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

[53] D. Ferone, P. Festa, A. Gruler, and A. A. Juan. Combining simulation with a grasp metaheuristic for solving the permutation flow-shop problem with stochastic processing times. In T. M. K. Roeder et al., editor, *Proceedings of the 2016 Winter Simulation Conference*, pages 2205–2215, Piscataway, New Jersey, 2016a. IEEE.

[54] D. Ferone, P. Festa, and M. G. Resende. Hybridizations of grasp with path relinking for the far from most string problem. *International Transactions in Operational Research*, 23(3):481–506, 2016c.

[55] D. Ferone, P. Festa, A. Napoletano, and T. Pastore. Shortest paths on dynamic graphs: a survey. *Pesquisa Operacional*, 37(3):487–508, 2017.

[56] P. Festa and M. Resende. *Hybridizations of GRASP with Path-Relinking*, volume 434, pages 135–155. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[57] 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.

[58] 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.

[59] P. Festa, D. Ferone, T. Pastore, A. A. Juan, and C. Bayliss. Integrating biased-randomized grasp with monte carlo simulation for solving the vehicle routing problem with stochastic demands. In *accepted paper at the 2018 Winter Simulation Conference*, 2018.

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

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

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

[63] Z. Fu and S. Malik. Solving the minimum-cost satisfiability problem using sat based branch-and-bound search. pages 852–859, 2006.

[64] G. Gallo. Reoptimization procedures in shortest path problem. *Rivista di matematica per le scienze economiche e sociali*, 3(1):3–13, 1980.

[65] 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.

[66] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *SOFTWARE - PRACTICE AND EXPERIENCE*, 30(11): 1203–1233, 2000.

[67] M. Gendreau, G. Laporte, and R. Séguin. An exact algorithm for the vehicle routing problem with stochastic demands and customers. *Transportation science*, 29(2): 143–155, 1995.

[68] M. Gendreau, G. Laporte, and R. Séguin. Stochastic vehicle routing. *European Journal of Operational Research*, 88(1):3–12, 1996.

[69] M. Gendreau, G. Laporte, and R. Séguin. A tabu search heuristic for the vehicle routing problem with stochastic demands and customers. *Operations Research*, 44(3): 469–477, 1996.

[70] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Science*, 8:156–166, 1977.

[71] F. Glover. Tabu search: Part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[72] F. Glover and M. Laguna. *Tabu Search*, pages 70–150. Blackwell Scientific Publications, Oxford, 1993.

[73] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997. ISBN 079239965X.

[74] F. Glover, M. Laguna, E. Taillard, and D. de Werra. Tabu search. *Annals of Operations Research*, 41, 1993.

[75] B. L. Golden, S. Raghavan, and E. A. Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43. Springer Science & Business Media, New York, 2008.

[76] G. H. Golub and J. H. Welsch. Calculation of gauss quadrature rules. *Mathematics of computation*, 23(106):221–230, 1969.

[77] S. Gonzalez-Martin, A. A. Juan, D. Riera, M. G. Elizondo, and J. J. Ramos. A simheuristic algorithm for solving the arc routing problem with stochastic demands. *Journal of Simulation*, 12(1):53–66, 2018.

[78] E. M. Gonzalez-Neira, D. Ferone, S. Hatami, and A. A. Juan. A biased-randomized simheuristic for the distributed assembly permutation flowshop problem with stochastic processing times. *Simulation Modelling Practice and Theory*, 79:23–36, 2017.

[79] A. Grasas, A. A. Juan, and H. R. Lourenço. Simils: a simulation-based extension of the iterated local search metaheuristic for stochastic combinatorial optimization. *Journal of Simulation*, 10(1):69–77, 2016.

[80] A. Grasas, A. A. Juan, J. Faulin, J. de Armas, and H. Ramalhinho. Biased randomization of heuristics using skewed probability distributions: a survey and some applications. *Computers & Industrial Engineering*, 110:216–228, 2017.

[81] T. Gschwind, J. Pinggera, S. Zugal, H. A. Reijers, and B. Weber. A linear time layout algorithm for business process models. *J. Vis. Lang. Comput.*, 25(2):117–132, Apr. 2014. ISSN 1045-926X.

[82] D. Guimarans, O. Dominguez, A. A. Juan, and E. Martinez. A multi-start simheuristic for the stochastic two-dimensional vehicle routing problem. In T. M. K. Roeder et al., editor, *Proceedings of the 2016 Winter Simulation Conference*, pages 2326–2334, Piscataway, New Jersey, 2016. IEEE.

[83] 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.

[84] E. Holmberg, B. Torstenfelt, and A. Klarbring. Stress constrained topology optimization. *Structural and Multidisciplinary Optimization*, 48(1):33–47, 2013.

[85] C. Hu, Y. Li, X. Cheng, and Z. Liu. A virtual dataspaces model for large-scale materials scientific data access. *Future Generation Computer Systems*, 54:456 – 468, 2016.

[86] A. A. Juan, J. Faulin, J. Jorba, J. Caceres, and J. M. Marquès. Using parallel & distributed computing for real-time solving of vehicle routing problems with stochastic demands. *Annals of Operations Research*, 207(1):43–65, 2013.

[87] A. A. Juan, B. B. Barrios, E. Vallada, D. Riera, and J. Jorba. A simheuristic algorithm for solving the permutation flow shop problem with stochastic processing times. *Simulation Modelling Practice and Theory*, 46:101–117, 2014a.

[88] A. A. Juan, S. E. Grasman, J. Caceres-Cruz, and T. Bektaș. A simheuristic algorithm for the single-period stochastic inventory-routing problem with stock-outs. *Simulation Modelling Practice and Theory*, 46:40–52, 2014b.

[89] A. A. Juan, I. Pascual, D. Guimarans, and B. Barrios. Combining biased randomization with iterated local search for solving the multidepot vehicle routing problem. *International Transactions in Operational Research*, 22(4):647–667, 2015.

[90] M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to the multi-layer crossing minimization problem. In *International Symposium on Graph Drawing*, pages 13–24. Springer, 1997.

[91] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *Journal of Graph Algorithms and Applications*, 1: Paper 1, 25 p., 1997. URL http://eudml.org/doc/48063.

[92] H.-P. Kriegel, P. Kröger, M. Renz, and T. Schmidt. Hierarchical graph embedding for efficient query processing in very large traffic networks. In B. Ludäscher and N. Mamoulis, editors, *Scientific and Statistical Database Management*, pages 150–167. Springer Berlin Heidelberg, 2008.

[93] M. Laguna and R. Martí. Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

[94] 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.

[95] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research*, 59(3):345–358, 1992.

[96] G. Laporte, F. Louveaux, and H. Mercure. The vehicle routing problem with stochastic travel times. *Transportation science*, 26(3):161–170, 1992.

[97] C. Le, J. Norato, T. Bruns, C. Ha, and D. Tortorelli. Stress-based topology optimization for continua. *Structural and Multidisciplinary Optimization*, 41(4):605–620, 2010.

[98] N. W. Lemons, B. Hu, and W. S. Hlavacek. Hierarchical graphs for rule-based modeling of biochemical systems. *BMC Bioinformatics*, 12(1):45, Feb 2011.

[99] M. Lozano, A. Duarte, F. Gortázar, and R. Martí. Variable neighborhood search with ejection chains for the antibandwidth problem. *Journal of Heuristics*, 18(6):919–938, 2012.

[100] T. W. Lucas, W. D. Kelton, P. J. Sanchez, S. M. Sanchez, and B. L. Anderson. Changing the paradigm: Simulation, now a method of first resort. *Naval Research Logistics (NRL)*, 62(4):293–303, 2015.

[101] 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.

[102] V. M. Manquinho, P. F. Flores, J. P. M. Silva, and A. L. Oliveira. Prime implicant computation using satisfiability algorithms. pages 232–239. IEEE, 1997.

[103] Y. Marinakis, G.-R. Iordanidou, and M. Marinaki. Particle swarm optimization for the vehicle routing problem with stochastic demands. *Applied Soft Computing*, 13(4): 1693–1704, 2013.

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

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

[106] R. Martí, A. Martínez-Gavara, J. Sánchez-Oro, and A. Duarte. Tabu search for the dynamic bipartite drawing problem. *Computers & Operations Research*, 91:1–12, 2018. ISSN 0305-0548.

[107] A. Martínez-Gavara, D. Landa-Silva, V. Campos, and R. Martí. Randomized heuristics for the capacitated clustering problem. *Information Sciences*, 417:154–168.

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

[109] 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.

[110] 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.

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

[112] 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.

[113] R. Martí. A tabu search algorithm for the bipartite drawing problem. *European Journal of Operational Research*, 106:558–569, 1998.

[114] R. Martí, V. Campos, A. Hoff, and J. Peiró. Heuristics for the min-max arc crossing problem in graphs. *Expert Systems with Applications*, 2018.

[115] R. Mateescu, R. Dechter, and R. Marinescu. And/or multi-valued decision diagrams (aomdds) for graphical models. *J. Artif. Int. Res.*, 33(1):465–519, Dec. 2008. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=1622698.1622711.

[116] C. Matuszewski, R. Schönfeld, and P. Molitor. Using sifting for k-layer straightline crossing minimization. In J. Kratochvíyl, editor, *Graph Drawing*, pages 217–224, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-46648-2.

[117] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. pages 530–535. ACM, 2001.

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

[119] A. Napoletano, A. Martínez-Gavara, P. Festa, T. Pastore, and R. Martí. Heuristics for the constrained incremental graph drawing problem. *European Journal of Operational Research*, 2018.

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

[121] S. C. North. Incremental layout in dynadag. In *In Proceedings of the 4th Symposium on Graph Drawing (GD*, pages 409–418. Springer-Verlag, 1996.

[122] B. Oselio, A. Kulesza, and A. O. Hero. Multi-layer graph analytics for social networks. In *2013 5th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 284–287, Dec 2013.

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

[124] J. París, F. Navarrina, I. Colominas, and M. Casteleiro. Topology optimization of continuum structures with local and global stress constraints. *Structural and Multidisciplinary Optimization*, 39(4):419–437, 2009.

[125] J. Pereira, E. Fancello, and C. Barcellos. Topology optimization of continuum structures with material failure constraints. *Structural and Multidisciplinary Optimization*, 26(1-2):50–66, 2004.

[126] B. Pinaud, P. Kuntz, and R. Lehn. *Dynamic Graph Drawing with a Hybridized Genetic Algorithm*, pages 365–375. Springer London, London, 2004.

[127] M. Pipponzi and F. Somenzi. An Iterative Approach to the Binate Covering Problem. *Proceedings of the European Conference on Design Automation*, pages 208–211, 1990.

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

[129] T. Raeder and N. V. Chawla. Market basket analysis with networks. *Social Network Analysis and Mining*, 1(2):97–113, Apr 2011. ISSN 1869-5469. doi: 10.1007/s13278-010-0003-7. URL https://doi.org/10.1007/s13278-010-0003-7.

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

[131] M. G. Resendel and C. C. Ribeiro. Grasp with path-relinking: Recent advances and applications. In *Metaheuristics: Progress as Real Problem Solvers*, pages 29–63. Springer, 2005.

[132] 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.

[133] E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. An improved simulated annealing algorithm for bandwidth minimization. *European Journal of Operation Research*, 185(3):1319–1335, 2008.

[134] G. Rozvany. *Topology optimization in structural mechanics*, volume 374. Springer, 2014.

[135] R. Ruiz and T. Stützle. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143–1159, 2008.

[136] J. Sánchez-Oro, A. Martínez-Gavara, M. Laguna, A. Duarte, and A. Martí. Variable neighborhood scatter search for the incremental graph drawing problem. *Computational Optimization and Applications*, 68:775–797, 2017.

[137] F. Scholz. Maximum likelihood estimation. *Encyclopedia of statistical sciences*, 1985.

[138] M. Servit and J. Zamazal. Heuristic approach to binate covering problem. pages 123–129. IEEE, 1992.

[139] O. Sigmund. A 99 line topology optimization code written in matlab. *Structural and multidisciplinary optimization*, 21(2):120–127, 2001.

[140] O. Sigmund. Morphology-based black and white filters for topology optimization. *Structural and Multidisciplinary Optimization*, 33(4):401–424, 2007.

[141] O. Sigmund and J. Petersson. Numerical instabilities in topology optimization: a survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Structural and Multidisciplinary Optimization*, 16(1):68–75, 1998.

[142] N. Sorensson and N. Een. Minisat v1. 13-a sat solver with conflict-clause minimization. *SAT*, 2005:53, 2005.

[143] M. F. Stallmann. A heuristic for bottleneck crossing minimization and its performance on general crossing minimization: Hypothesis and experimental study. *ACM Journal of Experimental Algorithms*, 17(1):1–30, 2012.

[144] M. Stolpe and K. Svanberg. On the trajectories of the epsilon-relaxation approach for stress-constrained truss topology optimization. *Structural and multidisciplinary optimization*, 21(2):140–151, 2001.

[145] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man, Cybern.*, 11:109–125, 1981.

[146] K. Svanberg. The method of moving asymptotes—a new method for structural optimization. *International journal for numerical methods in engineering*, 24(2): 359–373, 1987.

[147] D. Taş, M. Gendreau, N. Dellaert, T. Van Woensel, and A. De Kok. Vehicle routing with soft time windows and stochastic travel times: A column generation and branch-and-price solution approach. *European Journal of Operational Research*, 236(3): 789–799, 2014.

[148] P. Toth and D. Vigo. The granular tabu search and its application to the vehicle-routing problem. *Informs Journal on computing*, 15(4):333–346, 2003.

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

[150] K. Tretyakov, A. Armas-Cervantes, L. García-Bañuelos, J. Vilo, and M. Dumas. Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1785–1794. ACM, 2011.

[151] K. Truemper. *Design of logic-based intelligent systems*. John Wiley & Sons, 2004.

[152] J. Vanhatalo, H. Völzer, F. Leymann, and S. Moser. Automatic workflow graph refactoring and completion. In A. Bouguettaya, I. Krueger, and T. Margaria, editors, *Service-Oriented Computing – ICSOC 2008*, pages 100–115. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-89652-4.

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

[154] E. Weitschek, G. Felici, and P. Bertolazzi. Mala: a microarray clustering and classification software. pages 201–205. IEEE, 2012.

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

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

[157] M. Zhou and G. Rozvany. The coc algorithm, part ii: topological, geometrical and generalized shape optimization. *Computer Methods in Applied Mechanics and Engineering*, 89(1-3):309–336, 1991.

[158] M. Zhou, Y. K. Shyy, and H. L. Thomas. Checkerboard and minimum member size control in topology optimization. *Structural and Multidisciplinary Optimization*, 21 (2):152–158, 2001.