A BRIEF INTRODUCTION TO EXACT, APPROXIMATION, AND HEURISTIC ALGORITHMS FOR SOLVING HARD COMBINATORIAL OPTIMIZATION PROBLEMS

P. FESTA

Abstract. This paper presents a short (and not exhaustive) introduction to the most used exact, approximation, and metaheuristic algorithms for solving hard combinatorial optimization problems.

After introducing the basics of exact approaches such as Branch & Bound and Dynamic Programming, we focus on the basics of the most studied approximation techniques and of the most applied algorithms for finding good suboptimal solutions, including genetic algorithms, simulated annealing, tabu search, variable neighborhood search, greedy randomized adaptive search procedures (GRASP), path relinking, and scatter search.

1. Introduction

Any combinatorial optimization problem involves a finite number of alternatives: given a ground set $\mathcal{E} = \{1, \ldots, n\}$ and an objective function $f : 2^{\mathcal{E}} \mapsto \mathbb{R}$, the set of feasible solutions $\mathcal{X} \subseteq 2^{\mathcal{E}}$ is finite. In case of minimization (resp. maximization), one searches for an optimal solution $x^* \in \mathcal{X}$ such that $f(x^*) \leq f(x)$ (resp. $f(x^*) \geq f(x)$), $\forall\, x \in \mathcal{X}$. To illustrate one among the most famous examples of combinatorial optimization problems, let us consider the Traveling Salesman Problem (TSP), defined on an undirected edge-weighted graph $G = (V, E)$. In this case, the ground set $\mathcal{E}$ is the set of edges connecting nodes in V to be visited, $\mathcal{X}$ is formed by all edge subsets that determine a Hamiltonian cycle, and the objective function value $f(x)$ to be minimized is the sum of the costs of all edges in a solution $x \in \mathcal{X}$. As a further example, let us consider the Shortest Path Problem (SPP), defined on an directed arc-weighted graph $G = (V, A)$. In this case, the ground set $\mathcal{E}$ is the set of arcs connecting nodes in V, $\mathcal{X}$ is formed by all simple paths connecting two nodes $s, d \in V$, $s \neq d$, and the objective function value $f(x)$ to be minimized is the sum of the costs of all arcs in a solution $x \in \mathcal{X}$. As a further and last example, let us consider the Knapsack Problem (KP), where one is given a knapsack with finite capacity W and a set $O = \{o_1, \ldots, o_n\}$ of n objects. Each object $o_j \in O$, $j = 1, \ldots, n$, is associated with a weight $w_j \leq W$ and a profit p_j. Without loss of generality, one can assume that $W \in \mathbb{Z}^+ \cup \{0\}$, that for all $j = 1, \ldots, n$, w_j, $p_j \in \mathbb{Z}^+ \cup \{0\}$, and that obviously $\displaystyle\sum_{j=1}^{n} w_j > W$. In the case of KP, the ground set $\mathcal{E} = \{0, 1\}$, $\mathcal{X}$ is formed by all object subsets x such that the sum of the weights of the elements in x does not exceed W, and the objective function value $f(x)$ to be maximized is the sum of the profits of all objects in a solution $x \in \mathcal{X}$.

Key words and phrases. Hard combinatorial optimization, Exact solution, Metaheuristics, Approximate solutions.

Besides their theoretical relevance [46], combinatorial optimization problems have practical impact, given their applicability to real–world scenarios [47]. In fact, they arise in several and heterogenous domains, among many others we recall routing, scheduling, production planning, decision making process, location problems, transportation (air, rail, trucking, shipping), energy (electrical power, petroleum, natural gas), and telecommunications (design, location).

In principle, since the feasible solutions set $\mathcal{X}$ is finite, any combinatorial optimization problem could be exactly solved by an algorithm that simply enumerates all elements in $\mathcal{X}$ and outputs one among those elements corresponding to the best objective function value. Unfortunately, since the number of feasible solutions $|\mathcal{X}|$ usually grows exponentially with the size of the instance to be solved, such a naive approach is not efficient and surely not applicable to solve real–world applications of practical interest. Several combinatorial optimization problems can be solved in polynomial time (problems in P), such as for example SPP, when the costs associated with the arcs of the graph are nonnegative [10]. But many of them are hard or computationally intractable (NP-complete problems) in the sense that until now no exact polynomial time algorithm has been designed yet and if such exact approach would be ever proposed for any of these intractable problems, then it will result that P = NP, i.e. that all optimization problems are in P and therefore solvable in polynomial time with respect to the instance size [22]. Among the intractable problems, it has to be noticed that not all of them are at the same level of computational difficulty: some of them are intractable in the strong sense (strongly NP-complete problems). TSP, for example, is a strongly NP-complete problem, that can not be even approximated in polynomial time. KP instead is NP-complete not in the strong sense and can be approximated in polynomial time.

In the operations research community, an approximation method finds a suboptimal solution providing an approximation-guarantee on the quality of the solution found. An algorithm $\mathcal{A}lg$ for a minimization problem is said to be an approximation algorithm if the worst solution returned by $\mathcal{A}lg$ is not greater than ϵ times the best solution, for $\epsilon > 1$. In this case, ϵ is called the approximation guarantee of the proposed algorithm. Such approximation algorithms are important for intractable combinatorial optimization problems, since they are in a sense the best that can be done to give a guarantee of quality for the returned solution. A vast literature on approximation algorithms has been developed in the last decade, and good starting points are the books [12], [55], and [57]. Nevertheless, sometimes it is preferable to approach and solve the problem heuristically, especially in the presence of large scale problem instances. A heuristic approach finds a suboptimal solution whose quality can be only experimentally verified and therefore in general it is meaningful to apply it for solving optimization problems that are hard even to approximate.

This paper gives an overview of the most used exact, approximation, and metaheuristic algorithms for solving hard combinatorial optimization problems. After introducing the basics of exact approaches such as Branch & Bound and Dynamic Programming in Section 2 and of approximation strategies in Section 3, in Section 4 we focus on the basics of the most studied algorithms for finding good suboptimal solutions, including genetic algorithms, simulated annealing, tabu search, variable neighborhood search, greedy randomized adaptive search procedures (GRASP), path relinking, and scatter search.

Concluding remarks are given in the last section.

2. Exact approaches

Classical approaches to exactly solve a combinatorial optimization problem are Branch & Bound (whose study began already earlier 1966 [40]) and Dynamic Programming (whose study began already in 1957 [6]). They are divide-and-conquer methods, since both solve a problem by combining the solutions to its subproblems. The main difference between them resides in the way they divide a problem into subproblems. A Branch & Bound algorithm partitions the problem into independent subproblems, solves the subproblems and outputs as optimal solution to the original problem the best feasible solution found along the search. In contrast, a Dynamic Programming framework is applicable to a smaller set of optimization problems and more specifically those problems that can be divided into subproblems not independent, that is, into subproblems that share subsubproblems. In this context, a dynamic programming approach does not repeatedly solve common subsubproblems: each of them is solved just once and its optimal solution saved in a suitable table.

Referring to a general maximization problem, the subsequent two subsections are devoted to the description of Branch & Bound and Dynamic Programming, respectively.

2.1. **Branch & Bound.** Let Pr_0 be the combinatorial optimization problem to be solved, $\mathcal{X}(Pr_0)$ the set of its feasible solutions, and f^0_{opt} the optimal objective function value, i.e.

$$f^0_{opt} = \max \{f(x) \mid x \in \mathcal{X}(Pr_0)\}.$$

A Branch & Bound algorithm for Pr_0 partitions Pr_0 into a finite number of subproblems $Pr_1, \ldots, Pr_n$, such that any feasible solution for Pr_0 is a feasible solution of at least a subproblem Pr_i, $i = 1, \ldots, n$. The partition of Pr_0 is accomplished by partitioning $\mathcal{X}(Pr_0)$ into n subsets $\mathcal{X}(Pr_1), \ldots, \mathcal{X}(Pr_n)$, such that

$$\bigcup_{i=1}^{n} \mathcal{X}(Pr_i) = \mathcal{X}(Pr_0), \qquad \bigcap_{i=1}^{n} \mathcal{X}(Pr_i) = \emptyset.$$

Denoting by f^i_{opt} the optimal objective function value for the subproblem Pr_i, $i = 1, \ldots, n$, it clearly results that

$$f^0_{opt} = \max_{i=1,\ldots,n} f^i_{opt}.$$

Therefore, starting from an initial feasible solution x^0 for Pr_0 (eventually, empty) and that at any step of the method represents the better feasible solution found so far (*incumbent* solution), Pr_0 is solved by solving the subproblems $Pr_1, \ldots, Pr_n$.

For each subproblem Pr_i approached, one of the following scenarios must be revealed:

(1) Pr_i is impossible, i.e. $\mathcal{X}(Pr_i) = \emptyset$;
(2) there exists an optimal solution x^i, but it is useless to computed it, since it can be proved that it is not better than the incumbent solution, i.e.

$$f(x^i) = f^i_{opt} \leq f(x^0) \qquad (\textit{bounding criterion}); \tag{1}$$

(3) there exists an optimal solution x^i that must be computed and the incumbent solution x^0 eventually updated.

The advantage of any Branch & Bound algorithm is to take advantage of the property of each subproblem Pr_i, $i = 1, \ldots, n$, to be "easier" to be solved, since its feasible solution set $\mathcal{X}(Pr_i)$ contains a smaller number of elements. Nevertheless, each Pr_i, $i = 1, \ldots, n$, is a subproblem of Pr_0 with which it shares properties and characteristics. This implies that to approach Pr_i the same logic is applied as to deal with Pr_0, that is a partition of its feasible set $\mathcal{X}(Pr_i)$ has to be individuated into a finite number of subsets, and so on.

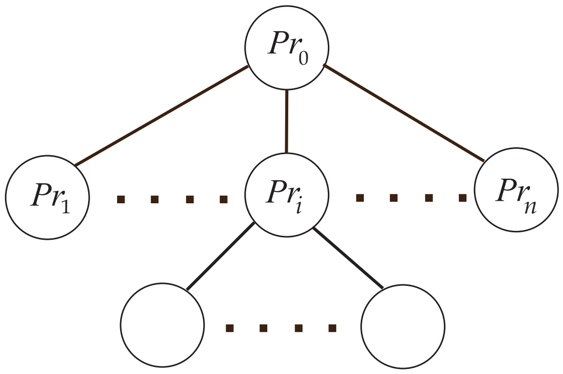

FIGURE 1. Branching tree: dynamical partition of problem Pr_0 into a finite number of subproblems $Pr_1, \ldots, Pr_n$ and subsequent subsubproblems.

This way to proceed is usually *dynamically* represented through a *decision tree*, called *branching tree*, as shown in Figure 1. The root node of the tree corresponds to the original problem Pr_0 and the current leaves correspond to the subproblems still to be solved.

A Branch & Bound algorithm is efficient when a high percentage of generated subproblems is solved without being further partitioned, because for each of them the bounding criterion can be verified. To this end, for each subproblem Pr_i generated, an upper bound $U(Pr_i)$ (lower bound in case of minimization problem) on the optimal objective function value f^i_{opt} is computed such that

$$f^i_{opt} \leq U(Pr_i).$$

Once known $U(Pr_i)$ for each subproblem Pr_i still to be solved, the bounding criterion described above in (1) can be rewritten as follows:

$$U(Pr_i) \leq f(x^0). \tag{2}$$

Such an upper bound $U(Pr_i)$ can be computed as objective function value corresponding to an optimal solution x^i_r to a problem R_i which is a *relaxed* version of problem Pr_i. Given a problem Pr_i, one of its relaxed problem is obtained from the mathematical formulation of Pr_i by eliminating at least one set of constraints and/or replacing at least one set of constraints with a weaker set of constraints.

Let R_i and $\mathcal{X}(R_i)$ be a relaxation of Pr_i and the set of its feasible solutions, respectively. Moreover, let x_r^i and f_r^i be an optimal solution for R_i and the corresponding objective function value, respectively. The following properties hold:

i. $\mathcal{X}(Pr_i) \subseteq \mathcal{X}(R_i)$;
ii. $\mathcal{X}(R_i) = \emptyset \Longrightarrow \mathcal{X}(Pr_i) = \emptyset$;
iii. $f_{opt}^i = \infty \Longrightarrow f_r^i = \infty$;
iv. $\mathcal{X}(R_i) \neq \emptyset$, $\mathcal{X}(Pr_i) \neq \emptyset \Longrightarrow f_{opt}^i \leq f_r^i$ (Upper Bound $U(Pr_i)$);
v. $x_r^i \in \mathbb{Z}^n \Longrightarrow x_r^i \in \mathcal{X}(Pr_i)$ and

$$f_r^i = f_{opt}^i,$$

i.e., x_r^i is an optimal solution for Pr_i.

The choice of the relaxing operations to be applied to the mathematical formulation of problem Pr_i in order to obtain problem R_i depends on the specific characteristics and properties of Pr_i (and therefore of the original problem Pr_0). This choice must take into account that the resulting relaxed problem R_i should be "easier" than Pr_i and that its optimal objective function value f_r^i has to be as much tight as possible to the optimal objective function value f_{opt}^i of problem Pr_i in order to augment as much as possible the number of times the bounding criterion is verified.

Among the most used relaxation strategies deserve to be mentioned the *linear programming relaxation*, the *Lagrangian relaxation*, the *semidefinite* and *surrogate relaxations*, and the *relaxation by elimination*. Since the end of the 60', a wide literature has been published describing details and properties of the different relaxation methods. The interested reader can refer to [40, 46, 42, 35, 44, 52, 53].

2.2. **Dynamic programming.** Dynamic Programming is another exact algorithmic framework for combinatorial optimization problems. It is based on the idea of defining an optimal solution of a problem Pr_0 as combination of the optimal solutions of a properly chosen set of subproblems of Pr_0, which are not necessarily independent. In more detail, let Pr_i and Pr_j be two subproblems of Pr_0. They are not independent when they share subsubproblems, i.e.

$$Pr_i = \{Pr_{i1}, \ldots, Pr_{il}\}, \qquad Pr_j = \{Pr_{j1}, \ldots, Pr_{jk}\}$$

and it results that

$$Pr_i = \{Pr_{i1}, \ldots, Pr_{il}\} \cap Pr_j = \{Pr_{j1}, \ldots, Pr_{jk}\} \neq \emptyset.$$

It is evident that Dynamic Programming can be applied to a smaller set of optimization problems and more specifically those problems Pr_0 that exhibit the following two properties:

(1) Pr_0 has an *optimal substructure*;
(2) Pr_0 can be divided into not independent subproblems.

Property (1) guarantees that an optimal solution for Pr_0 can be constructed from optimal solutions to its subproblems. Moreover, any Dynamic Programming algorithm takes practical advantage of property (2). In fact, it does not repeatedly solve common subsubproblems: each of them is solved just once and its optimal solution saved in a suitable table.

The design of a Dynamic Programming algorithm to exactly solve a problem Pr_0 typically goes through the following steps:

♢ verify that Pr_0 exhibits optimal substructure;

- ♢ characterize the structure of problem Pr_0 as a problem divisible into a finite number of subproblems not necessarily independent;
- ♢ recursively define the optimal objective function value of Pr_0 as a proper combination of the optimal function values of its subproblems;
- ♢ compute the optimal objective function value of Pr_0 in a bottom-up fashion, starting from the obvious and immediate solution of the elementary atomic subsubproblems, i.e. those subsubproblems that can not be further divided;
- ♢ construct an optimal solution for Pr_0 from computed information.

Dynamic Programming has been applied to a wide range of problems and two among the most interesting books about this powerful algorithmic framework are [8, 54].

3. Approximation strategies

Approximation methods find a suboptimal solution providing an approximation-guarantee on the quality of the solution found. They are applied for approaching intractable combinatorial optimization problems, since they are in a sense the best that can be done to give a guarantee of quality for the returned solution.

Let $\mathcal{A}$ be an approximation algorithm for a problem Pr_0. For each instance I of Pr_0, let $f^0_{opt}(I)$ be the optimal objective function value corresponding to an optimal solution x^0_I and let f^0_{appr} be the objective function value corresponding to a solution x^0_{appr} found by $\mathcal{A}$.

The *worst case approximation ratio of* $\mathcal{A}$ (or *worst case absolute error of* $\mathcal{A}$) is the smallest $r(\mathcal{A}) \in \mathbb{R}$ such that

$$R_{\mathcal{A}}(I, x^0_{appr}) = \max\left\{\frac{f^0_{opt}}{f^0_{appr}}, \frac{f^0_{appr}}{f^0_{opt}}\right\} \leq r(\mathcal{A}), \quad \text{for each instance } I \text{ of } Pr_0.$$

The *worst case relative error of* $\mathcal{A}$ is the smallest $\epsilon(\mathcal{A}) \in \mathbb{R}$ such that

$$E_{\mathcal{A}}(I, x^0_{appr}) = \frac{|f^0_{opt} - f^0_{appr}|}{\max\{f^0_{appr}, f^0_{appr}\}} \leq \epsilon(\mathcal{A}), \quad \text{for each instance } I \text{ of } Pr_0.$$

Worst case absolute error and worst case relative error are strongly related each other. In fact, it can be easily seen that

- $E_{\mathcal{A}}(I, x^0_{appr}) = 1 - \frac{1}{R_{\mathcal{A}}(I, x^0_{appr})}$;
- $r(\mathcal{A}) = 1 \Longrightarrow x^0_{appr}$ is optimal;
  $r(\mathcal{A}) \gg 1 \Longrightarrow x^0_{appr}$ is not a “good” solution;
- $\epsilon(\mathcal{A}) = 0 \Longrightarrow x^0_{appr}$ is optimal;
  $\epsilon(\mathcal{A}) \to 1 \Longrightarrow x^0_{appr}$ is not a “good” solution.

Several methods usually applied to find (in polynomial time) optimal solutions for computationally tractable problems often are applied to find a approximation-guaranteed suboptimal solutions for intractable problems. They include:

- greedy algorithms;
- sequential algorithms;
- local search algorithms;
- linear programming and relaxation based algorithms;
- dynamic programming algorithms;
- random algorithms.

A vast literature on approximation algorithms has been developed in the last decade, and good starting points are the books [12], [55], and [57].

4. Heuristic/metaheuristic approaches

Due to the computational complexity of hard combinatorial problems, especially in the presence of large scale problem instances, in the last decades there has been an extensive research effort devoted to the development of heuristic algorithms.

There is no general framework behind the design of a good heuristic method that for any problem is guaranteed able to find good quality solution. The effectiveness of a heuristic method depends upon its ability to adapt to a particular realization, avoid entrapment at local optima, and exploit the basic structure of the problem. Building on these notions, various heuristic search techniques have been developed that have demonstrably improved our ability to obtain good solutions to difficult combinatorial optimization problems. One of the most promising of such techniques are usually called *metaheuristics* and include, but are not restricted to, simulated annealing [36], tabu search [23, 24, 27], evolutionary algorithms like genetic algorithms [30], ant colony optimization [11], scatter search [29, 38, 39], path-relinking [25, 26, 27, 28], iterated local search [5, 41], variable neighborhood search [31], and GRASP (Greedy Randomized Adaptive Search Procedures) [13, 14].

Metaheuristics are a class of methods commonly applied to suboptimally solve computationally intractable combinatorial optimization problems. The term metaheuristic derives from the composition of two Greek words: *meta* and *heuriskein*. The suffix 'meta' means 'beyond', 'in an upper level', while 'heuriskein' means 'to find'. In fact, metaheuristics are a family of algorithms that try to combine basic heuristic methods in higher level frameworks aimed at efficiently exploring the set of feasible solution of a given combinatorial problem. In [56] the following definition has been given:

> "A metaheuristic is an *iterative master process* that guides and modifies the operations of *subordinate heuristics* to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method."

Osman and Laporte [45] in their metaheuristics bibliography define a metaheuristic approach as follows:

> "A metaheuristic is formally defined as an *iterative generation process* which guides a *subordinate heuristic* by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions. "

In the reminder of this section, referring to a general minimization combinatorial optimization problem as defined in Section 1, the basics of the most applied metaheuristics will be described.

4.1. **GRASP.** A GRASP metaheuristic [13, 14, 20] is a multi-start or iterative method, where each iteration is usually made up of two phases: a construction phase and a local search phase. The solution constructed during the construction phase is not necessarily feasible, and if this is the case, then a repairing procedure is invoked to restore its feasibility. The local search phase starts at the constructed solution and applies iterative improvement until a locally optimal solution is found. Repeated applications of the construction procedure yields diverse starting solutions for the local search and the best overall solution is kept as the result. Extensive survey of the literature about GRASP and the several ways it can be hybridized with other metaheuristic schemes can be found in [18, 16, 17, 19, 21].

```
algorithm GRASP($f(\cdot)$, $g(\cdot)$, MaxIterations, Seed)
1     $x_{best}$:=$\emptyset$; $f(x_{best})$:=$+\infty$;
2     for $k = 1, 2, \ldots,$MaxIterations$\rightarrow$
3         $x$:=ConstructGreedyRandomizedSolution(Seed, $g(\cdot)$);
4         if ($x$ not feasible) then
5             $x$:=repair($x$);
6         endif
7         $x$:=LocalSearch($x$, $f(\cdot)$);
8         if ($f(x) < f(x_{best})$) then
9             $x_{best}$:=$x$;
10        endif
11    endfor;
12    return($x_{best}$);
end GRASP
```

FIGURE 2. Pseudo-code of a GRASP for a minimization problem.

The pseudo-code in Figure 2 illustrates the main blocks of a GRASP procedure for minimization, in which `MaxIterations` iterations (loop in lines 2–11) are performed and `Seed` is used as the initial seed for the pseudorandom number generator.

The construction phase performed in line 3 builds a solution x. If x is not feasible, a repair procedure is invoked in line 5 to obtain feasibility. Once a feasible solution x is obtained, its neighborhood is investigated by the local search in line 7 until a local minimum is found. The best overall solution is kept and output as the result in line 12.

The basic GRASP construction phase is similar to the semi-greedy heuristic proposed independently by [33]. Figure 3 reports the pseudo-code of the basic GRASP construction procedure. Starting from an empty solution (line 1), in the construction phase, a complete solution is iteratively constructed (loop in lines 3–8), one element at a time. At each construction iteration, the choice of the next element to be added is determined by ordering (line 2) all candidate elements (i.e. those that can be added to the solution) in a candidate list C with respect to a greedy function $g : C \mapsto \mathbb{R}$. This function measures the (myopic) benefit of selecting each element. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, but not necessarily the top candidate. The list of best candidates is called the *restricted candidate list* (RCL) and is computed in line 4. In other words, the RCL is made up of elements $i \in C$ with the best (i.e., the smallest) incremental costs $g(i)$. There are two main mechanisms to build this list: a *cardinality-based* (CB) and a *value-based* (VB) mechanism. In the CB case, the RCL is made up

```
procedure ConstructGreedyRandomizedSolution(Seed, $g(\cdot)$)
1   $x$:=$\emptyset$;
2   Sort the candidate elements $i$ according to their incremental
    costs $g(i)$;
3   while ($x$ is not a complete solution)$\rightarrow$
4       RCL:=MakeRCL();
5       $v$:=SelectIndex(RCL, Seed);
6       $x := x \cup \{v\}$;
7       Resort remaining candidate elements $j$ according to their
        incremental costs $g(j)$;
8   endwhile;
9   return($x$);
end ConstructGreedyRandomizedSolution
```

FIGURE 3. Basic GRASP construction phase pseudo-code.

of the k elements with the best incremental costs, where k is a parameter. In the VB case, the RCL is associated with a parameter $\alpha \in [0,1]$ and a threshold value $\mu = g_{min} + \alpha(g_{max} - g_{min})$, where g_{min} and g_{max} are the smallest and the largest incremental costs, respectively, i.e.

$$g_{min} = \min_{i \in C} g(i), \qquad g_{max} = \max_{i \in C} g(i). \tag{3}$$

Then, all candidate elements i whose incremental cost $g(i)$ is no greater than the threshold value are inserted into the RCL, i.e. $g(i) \in [g_{min}, \mu]$. Note that, the case $\alpha = 0$ corresponds to a pure greedy algorithm, while $\alpha = 1$ is equivalent to a random construction.

```
procedure LocalSearch($x$, $f(\cdot)$)
1   Let $N(x)$ be the neighborhood of $x$;
2   $H$:=$\{y \in N(x) \mid f(y) < f(x)\}$;
3   while ($|H| > 0$)$\rightarrow$
4       $x$:=Select($H$);
5       $H$:=$\{y \in N(x) \mid f(y) < f(x)\}$;
6   endwhile
7   return($x$);
end LocalSearch
```

FIGURE 4. Pseudo-code of a generic local search procedure.

Solutions generated by a GRASP construction are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm iteratively replaces the current solution by a better solution in the neighborhood of the current solution. It terminates when no better solution is found in the neighborhood. The *neighborhood structure* N for a problem relates a solution s of the problem to a subset of solutions $N(s)$. A solution s is said to be *locally optimal* if in $N(s)$ there is no better solution in terms of objective function value. Figure 4 illustrates the pseudo-code of a generic local search procedure for a minimization problem.

It is difficult to formally analyze the quality of solution values found by using the GRASP methodology. However, there is an intuitive justification that views GRASP as a repetitive sampling technique [49].

GRASP can be implemented either sequentially or in parallel, where only a single global variable is required to store the best solution found over all processors. Moreover, few parameters need to be set and tuned, and therefore development can focus on implementing efficient data structures to assure quick GRASP iterations.

4.2. **Path-relinking.** Path-relinking has been proposed in 1996 by Glover [25] as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search [26, 27, 28].

Starting from one or more elite solutions, path-relinking generates paths in the solution space leading towards other guiding elite solutions with the aim of searching better solutions. This task is accomplished by selecting moves that introduce attributes contained in the guiding solutions. At each iteration, all moves that incorporate attributes of the guiding solution are analyzed and the move that best improves (or least deteriorates) the initial solution is chosen.

Path-relinking is applied to a pair of good quality solutions $\mathbf{x}$, called the *initial solution*, and $\mathbf{y}$, called the *guiding solution*. $\mathbf{x}$ and $\mathbf{y}$ are elements of a set $\mathcal{E}$ of elite solutions that has usually a fixed size that does not exceed `MaxElite`. Given the pair $\mathbf{x}, \mathbf{y}$, their common elements are kept constant, and the space of solutions spanned by these elements is searched. The procedure starts by computing the symmetric difference $\Delta(\mathbf{x}, \mathbf{y})$ between the two solutions, i.e. the set of moves needed to reach $\mathbf{y}$ (target solution) from $\mathbf{x}$ (initial solution). A path of solutions is generated linking $\mathbf{x}$ and $\mathbf{y}$. The best solution x^* in this path is returned by the algorithm.

Let us denote the set of solutions spanned by the common elements of the n-vectors $\mathbf{x}$ and $\mathbf{y}$ as

$$S(\mathbf{x}, \mathbf{y}) := \{w \text{ feasible} \mid w_i = x_i = y_i, i \notin \Delta(\mathbf{x}, \mathbf{y})\} \setminus \{\mathbf{x}, \mathbf{y}\}. \tag{4}$$

Clearly, $|S(\mathbf{x}, \mathbf{y})| = 2^{n-d(\mathbf{x},\mathbf{y})} - 2$, where $d(\mathbf{x}, \mathbf{y}) = |\Delta(\mathbf{x}, \mathbf{y})|$. The underlying assumption of path-relinking is that there exist good-quality solutions in $S(\mathbf{x}, \mathbf{y})$, since this space consists of all solutions which contain the common elements of two good solutions $\mathbf{x}$ and $\mathbf{y}$. Since the size of this space is exponentially large, a greedy search is usually performed where a path of solutions

$$\mathbf{x} = \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{d(\mathbf{x},\mathbf{y})}, \mathbf{x}_{d(\mathbf{x},\mathbf{y})+1} = \mathbf{y}, \tag{5}$$

is built, such that $d(\mathbf{x}_i, \mathbf{x}_{i+1}) = 1, \quad i = 0, \ldots, d(\mathbf{x}, \mathbf{y})$, and the best solution from this path is chosen. Note that, since both $\mathbf{x}$ and $\mathbf{y}$ are, by construction, local optima in some neighborhood $N(\cdot)$[1], in order for $S(\mathbf{x}, \mathbf{y})$ to contain solutions which are not contained in the neighborhoods of $\mathbf{x}$ or $\mathbf{y}$, $\mathbf{x}$ and $\mathbf{y}$ must be sufficiently distant.

Figure 5 illustrates the pseudo-code of the path-relinking procedure applied to the pair of solutions $\mathbf{x}$ and $\mathbf{y}$. In line 1, the initial solution $\mathbf{x}$ is selected at random among the elite set elements and it usually differs sufficiently from the guiding solution $\mathbf{y}$. The loop in lines 6 through 14 computes a path of solutions $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{d(\mathbf{x},\mathbf{y})-2}$, and the solution x^* with the best objective function value is returned in line 15. This is achieved by advancing one solution at a time in a greedy

[1]The same metric $d(\mathbf{x}, \mathbf{y})$ is usually used.

```
algorithm Path-relinking($f(\cdot), \mathbf{x}, \mathcal{E}$)
1     Choose, at random, a pool solution $\mathbf{y} \in \mathcal{E}$ to relink with $\mathbf{x}$;
2     Compute symmetric difference $\Delta(\mathbf{x}, \mathbf{y})$;
3     $f^* := \min\{f(\mathbf{x}), f(\mathbf{y})\}$;
4     $x^* := \arg\min\{f(\mathbf{x}), f(\mathbf{y})\}$;
5     $x := \mathbf{x}$;
6     while $(\Delta(x, \mathbf{y}) \neq \emptyset) \rightarrow$
7         $m^* := \arg\min\{f(x \oplus m) \mid m \in \Delta(x, \mathbf{y})\}$;
8         $\Delta(x \oplus m^*, \mathbf{y}) := \Delta(x, \mathbf{y}) \setminus \{m^*\}$;
9         $x := x \oplus m^*$;
10        if $(f(x) < f^*)$ then
11            $f^* := f(x)$;
12            $x^* := x$;
13        endif;
14    endwhile;
15    $x^* :=$ LocalSearch($x^*, f(\cdot)$);
16    return $(x^*)$;
end Path-relinking
```

FIGURE 5. Pseudo-code of a generic path-relinking for a minimization problem.

manner. At each iteration, the procedure examines all moves $m \in \Delta(x, \mathbf{y})$ from the current solution x and selects the one which results in the least cost solution (line 7), i.e. the one which minimizes $f(x \oplus m)$, where $x \oplus m$ is the solution resulting from applying move m to solution x. The best move m^* is made, producing solution $x \oplus m^*$ (line 9). The set of available moves is updated (line 8). If necessary, the best solution x^* is updated (lines 10–13). $\Delta(x, \mathbf{y}) = \emptyset$. Since x^* is not guaranteed to be locally optimal, a local search is usually applied (line 15) and the locally optimal solution is returned.

In the literature, several contributions appeared showing how to hybridize path-relinking with other metaheuristics, both as diversification and as intensification phase.

4.3. **Tabu search.** Tabu search (TS) is a metaheuristic strategy introduced by Glover [23, 24, 25, 27] that makes use of memory structures to enable escape from local minima by allowing cost-increasing moves. During the search, short-term memory TS uses a special data structure called *tabu list* to store information about solutions generated in the last iterations[2]. The process starts from a given solution and, as any local search heuristic, it moves in iterations from the current solution s to some solution $t \in N(s)$. To avoid returning to a just-visited local minimum, reverse moves mov_t that lead back to that local minimum are forbidden, or made *tabu*, for a number of iterations that can be a priori fixed (fixed sized tabu list) or adaptively varying (variable sized tabu list).

Figure 6 shows pseudo-code for a short-term TS using a fixed k sized tabu list T, that, for ease of handling, stores the complete solutions t instead of the

[2]Usually, the tabu list stores all moves that reverse the effect of recent local search steps.

```
algorithm TS($x$, $f(\cdot)$, $k$)
    Let $N(x)$ be the neighborhood of $x$;
    $s := x$; $T := \emptyset$; $x_b := x$;
    while (stopping criterion not satisfied)$\rightarrow$
        $\hat{N}(s) := N(s) \setminus T$;
        $t :=$argmin$\{f(w) \mid w \in \hat{N}(s)\}$;
        if ($|T| \geq k$) then
            Remove from $T$ the oldest entry;
        endif
        $T := T \cup \{t\}$;
        if ($f(t) < f(x_b)$) then
            $x_b := t$;
        endif
        $s := t$;
    endwhile
    return($x_b$);
end TS
```

FIGURE 6. Short memory TS pseudo-code for a minimization problem.

```
algorithm SIMULATED-ANNEALING ($x$, $f(\cdot)$, $T$, Seed)
    $s := x$; $x_b := x$;
    while ($T > 0$ and stopping criterion not satisfied)$\rightarrow$
        $t :=$select-randomly(Seed, $N(s)$);
        if ($f(t) - f(s) < 0$) then
            $s := t$;
            if ($f(t) < f(x_b)$) then $x_b := t$;
            endif
        else $s := t$ with probability $e^{-(f(t)-f(s))/(K \cdot T))}$;
        endif
        Decrement $T$ according to a defined criterion;
    endwhile
    return ($x_b$);
end simulated-annealing
```

FIGURE 7. SA pseudo-code for a minimization problem.

corresponding moves mov_t. In TS literature, $|T|$ is called *tabù tenure* and clearly controls the memory of the search process.

4.4. **Simulated annealing.** Simulated annealing (SA) [36] is based on principles of mechanical statistics and on the idea of simulating the annealing process of a mechanical system. Pseudo-code of a generic simulated annealing is depicted in Figure 7.

As any stochastic local search procedure, SA is also given a starting solution x which is used to initialize the current solution s. At each iteration, it randomly selects a trial solution $t \in N(s)$. In perfect correspondence of mechanical systems state change rules, if t is an improving solution, then t is made the current solution.

Otherwise, t is made the current solution with probability given by

$$e^{-\frac{f(t)-f(s)}{K\cdot T}}, \tag{6}$$

where $f(x)$ is interpreted as the energy of the system in state x, K is the Boltzmann constant, and T a control parameter called the *temperature*.

There are many ways to implement SA, depending on the adopted stopping criterion and on the rule (*cooling schedule*) applied to decrement the temperature parameter T (line 10). Note that, the higher is the temperature T the higher is the probability of moving on a not improving solution t. Usually, starting from a high initial temperature T_0, at iteration k the cooling schedule changes the temperature by setting $T_{k+1} := T_k \cdot \gamma$, where $0 < \gamma < 1$.

Therefore, initial iterations can be thought of as a diversification phase, where a large part of the solution space is explored. As the temperature cools, fewer non-improving solutions are accepted and those cycles can be thought of as intensification cycles.

4.5. **Genetic algorithms and population-based heuristics.** Evolutionary metaheuristics such as genetic algorithms (GA) [30], ant colony optimization [11], scatter search [29, 38, 39], and evolutionary path-relinking [50] require the generation of an initial population of solutions.

Rooted in the mechanisms of evolution and natural genetics and therefore derived from the principles of natural selection and Darwin's evolutionary theory, the study of heuristic search algorithms with underpinnings in natural and physical processes began as early as the 1970s, when Holland [34] first proposed genetic algorithms. This type of evolutionary technique has been theoretically and empirically proven to be a robust search method [30] having a high probability of locating the global solution optimally in a multimodal search landscape.

In nature, competition among individuals results in the fittest individuals surviving and reproducing. This is a natural phenomenon called *the survival of the fittest*: the genes of the fittest survive, while the genes of weaker individuals die out. The reproduction process generates diversity in the gene pool. Evolution is initiated when the genetic material (chromosomes) from two parents recombines during reproduction. The exchange of genetic material among chromosomes is called *crossover* and can generate good combination of genes for better individuals. Another natural phenomenon called *mutation* causes regenerating lost genetic material. Repeated selection, mutation, and crossover cause the continuous evolution of the gene pool and the generation of individuals that survive better in a competitive environment.

In complete analogy with nature, once encoded each possible point in the search space of the problem into a suitable representation, a GA transforms a population of individual solutions, each with an associated *fitness* (or objective function value), into a new generation of the population. By applying genetic operators, such as crossover and mutation [37], a GA successively produces better approximations to the solution. At each iteration, a new generation of approximations is created by the process of selection and reproduction. In Figure 8 a simple genetic algorithm is described by the pseudo-code, where $P(k)$ is the population at iteration k.

In solving a given optimization problem $\mathcal{P}$, a GA consists of the following basic steps.

```
algorithm GA($f(\cdot)$)
1   Let $N(x)$ be the neighborhood of a solution $x$;
2   $k := 0$;
3   Initialize population $P(0)$; $x_b$ :=argmin$\{f(x) \mid x \in P(0)\}$;
4   while (stopping criterion not satisfied)$\rightarrow$
5       $k := k + 1$;
6       Select $P(k)$ from $P(k-1)$;
7       $t$ :=argmin$\{f(x) \mid x \in P(k)\}$;
8       if ($f(t) < f(x_b)$) then
9           $x_b := t$;
10      endif
11      Alter $P(k)$;
12  endwhile
13  return($x_b$);
end GA
```

FIGURE 8. Pseudo-code of a generic GA for a minimization problem.

(1) Randomly create an initial population $P(0)$ of individuals, i.e. solutions for $\mathcal{P}$.
(2) Iteratively perform the following substeps on the current generation of the population until the termination criterion has been satisfied.
   (a) Assign fitness value to each individual using the fitness function.
   (b) Select parents to mate.
   (c) Create children from selected parents by crossover and mutation.
   (d) Identify the best-so-far individual for this iteration of the GA.

Scatter Search (SS) operates on a *reference set* of solutions, that are combined to create new ones. One way to obtain a new solution is to linearly combine two reference set solutions. Unlike a GA, the reference set of solutions is relatively small, usually consisting of less than 20 solutions. At the beginning, a starting set of solutions is generated to guarantee a critical level of diversity and some local search procedure is applied to attempt to improve them. Then, a subset of the best solutions is selected as reference set, where the quality of a solution is evaluated both in terms of objective function and diversity with other reference set candidates. At each iteration, new solutions are generated by combining reference set solutions. One criterion used to select reference solutions for combination takes into account the convex regions spanned by the reference solutions.

Evolutionary path-relinking (EvPR) has been introduced by Resende and Werneck [50] and applied as a post-processing phase for GRASP with PR. In EvPR, the solutions in the pool are evolved as a series of populations $P(1), P(2), \ldots$ of equal size. The initial population $P(0)$ is the pool of elite solutions produced by GRASP with PR. In iteration k, PR is applied between a set of pairs of solutions in population $P(k)$ and, with the same rules used to test for membership in the pool of elite solutions, each resulting solution is tested for membership in population $P(k+1)$. This evolutionary process is repeated until no improvement is seen from one population to the next.

As just described, all above techniques are evolutionary metaheuristics requiring the generation of an initial population of solutions. Usually, these initial solutions are randomly generated, but another way to generate them is to use a GRASP.

Ahuja et al. [3] used a GRASP to generate the initial population of a GA for the quadratic assignment problem. Alvarez et al. [4] proposed a GRASP embedded scatter search for the multicommodity capacitated network design problem. Very recently, Contreras and Díaz used GRASP to initialize the reference set of scatter search for the single source capacitated facility location problem. GRASP with EvPR has been recently used in [51] for the uncapacitated facility location problem and in [48] for the max-min diversity problem.

4.6. **Variable neighborhood search.** Almost all randomization effort in implementations of the basic GRASP involves the construction phase. On the other hand, strategies such as Variable Neighborhood Search (VNS) and Variable Neighborhood Descent (VND) [32, 43] rely almost entirely on the randomization of the local search to escape from local optima. With respect to this issue, probabilistic strategies such as GRASP and VNS may be considered as complementary and potentially capable of leading to effective hybrid methods.

The variable neighborhood search (VNS) metaheuristic, proposed by Hansen and Mladenović [32], is based on the exploration of a dynamic neighborhood model. Contrary to other metaheuristics based on local search methods, VNS allows changes of the neighborhood structure along the search.

VNS explores increasingly distant neighborhoods of the current best found solution. Each step has three major phases: neighbor generation, local search, and jump. Let N_k, $k = 1, \dots, k_{max}$ be a set of pre-defined neighborhood structures and let $N_k(x)$ be the set of solutions in the kth-order neighborhood of a solution x. In the first phase, a neighbor $x' \in N_k(x)$ of the current solution is applied. Next, a solution x'' is obtained by applying local search to x'. Finally, the current solution jumps from x to x'' in case the latter improved the former. Otherwise, the order of the neighborhood is increased by one and the above steps are repeated until some stopping condition is satisfied.

```
algorithm VNS($x$, $f(\cdot)$, $k_{max}$, Seed)
1   $x_b := x$; $k := 1$;
2   while ($k \le k_{max}$)$\rightarrow$
3       $x'$ :=select-randomly(Seed, $N_k(x)$);
4       $x''$ :=LocalSearch($x'$, $f(\cdot)$);
5       if ($f(x'') < f(x')$) then
6           $x := x''$; $k := 1$;
7           if ($f(x'') < f(x_b)$) then $x_b := x''$;
8           endif
9       else $k := k + 1$;
10      endif
11  endwhile
12  return($x_b$);
end VNS
```

Figure 9. Pseudo-code of a generic VNS for a minimization problem.

Usually, until a stopping criterion is met, VNS generates at each iteration a solution x at random. In hybrid GRASP with VNS, where VNS is applied as local search, the starting solution is the output x of the GRASP construction procedure and the pseudo-code of a generic VNS local search is illustrated in Figure 9.

Examples of GRASP with VNS include [9] for the prize-collecting Steiner tree problem in graphs, [15] for the MAX-CUT problem, and [7] for the strip packing problem.

VND allows the systematic exploration of multiple neighborhoods and is based on the facts that a local minimum with respect to one neighborhood is not necessarily a local minimum with respect to another and that a global minimum is a local minimum with respect to all neighborhoods. VND also is based on the empirical observation that, for many problems, local minima with respect to one or more neighborhoods are relatively close to each other. Since a global minimum is a local minimum with respect to all neighborhoods, it should be easier to find a global minimum if more neighborhoods are explored.

```
algorithm VND($x$, $f(\cdot)$, $k_{max}$)
1    $x_b := x$; $s := x$; flag:=true;
2    while (flag)$\rightarrow$
3        flag:=false;
4        for $k = 1, \ldots, k_{max} \rightarrow$
5            if ($\exists t \in N_k(s) \mid f(t) < f(s)$) then
6                if ($f(t) < f(x_b)$) then $x_b := t$;
7                endif
8                $s := t$; flag:=true; break;
9            endif
10       endfor
11   endwhile
12   return($x_b$);
end VND
```

FIGURE 10. Pseudo-code of a generic VND for a minimization problem.

Let $N_k(x)$, for $k = 1, \ldots, k_{max}$, be k_{max} neighborhood structures of solution x. The search begins with a given starting solution x which is made the current solution s. Each major iteration (lines 2–11) searches for an improving solution t in up to k_{max} neighborhoods of s. If no improving solution is found in any of the neighborhoods, the search ends. Otherwise, t is made the current solution s and the search is applied starting from s.

4.7. **Iterated local search.** Iterated Local Search (ILS) [41] is a multistart heuristic that at each iteration k finds a locally optimal solution searched in the neighborhood of an initial solution obtained by perturbation of the local optimum found by local search at previous iteration $k - 1$. Figure 11 depicts the pseudo-code of a generic ILS metaheuristic.

The efficiency of ILS strongly depends on the perturbation (line 3) and acceptance criterion (line 5) rules. A "light" perturbation may cause local search to lead back to the starting solution t, while a "strong" perturbation may cause the search to resemble random multi-start. Usually, the acceptance criterion resembles SA,

```
algorithm ILS ($x$, $f(\cdot)$, history)
1     $t$ :=LocalSearch($x$, $f(\cdot)$); $x_b := t$;
2     while (stopping criterion not satisfied)$\rightarrow$
3         $s$ :=perturbation($t$, history);
4         $\hat{s}$ :=LocalSearch($s$, $f(\cdot)$);
5         $t$ :=AcceptanceCriterion($t$, $\hat{s}$, history);
6         if $(f(t) < f(x_b))$ then $x_b := t$;
7         endif
8     endwhile
9     return $(x_b)$;
end ils
```

Figure 11. ILS pseudo-code for a minimization problem.

i.e. $\hat{s}$ is accepted if it is an improving solution; otherwise, it is accepted with some properly chosen probability.

4.8. **Very-large scale neighborhood search.** As for any local search procedure, to efficiently search in the neighborhood of a solution, it is required that the neighborhood have a small size. Nevertheless, the larger the neighborhood, the better the quality of the locally optimal solution. Neighborhoods whose sizes grow exponentially as a function of problem dimension are called *very large scale neighborhoods* and they necessarily require efficient search techniques to be explored.

Ahuja et al. [1] presented a survey of methods called very-large scale neighborhood (VLSN) search. The following three classes of VLSN methods are described:

(1) variable-depth methods where exponentially large neighborhoods are searched with heuristics;
(2) a VLSN method that uses network flow techniques to identify improving neighborhood solutions;
(3) a VLSN method that explores neighborhoods for NP-hard problems induced by restrictions of the problems that are solved in polynomial time.

In particular, with respect to class 2, they define special neighborhood structures called multi-exchange neighborhoods. The search is based on the cyclic transfer neighborhood structure that transforms a cost-reducing exchange into a negative cost subset-disjoint cycle in an *improving graph* and then a modified shortest path label-correcting algorithm is used to identify these negative cycles.

Ahuja et al. in [2] present two generalizations of the best known neighborhood structures for the capacitated minimum spanning tree problem. The new neighborhood structures defined allow cyclic exchanges of nodes among multiple subtrees simultaneously. To judge the efficacy of the neighborhoods, local improvement and tabu search algorithms have been developed. Local improvement uses a GRASP construction mechanism to generate repeated starting solutions for local improvement.

5. Concluding remarks

Combinatorial optimization problems have both theoretical importance and practical impact, since they arise in an enormous range of applications in heterogeneous fields, from biological contexts to industries, and real life scenarios.

Many combinatorial optimization problems are tractable from a computational point of view, in the sense that they can be exactly solved with an algorithm whose computational complexity is in the worst case bounded by a function that grows polynomially in the size of the instance to be solved. Unfortunately, for many others combinatorial problems, such an algorithm has not been yet proposed and it is unlikely that it ever exists. Those computationally intractable problems can be approached by applying an approximation technique, if it is provable that such an algorithm exists, otherwise, they are usually heuristically solved.

In this chapter, we have briefly surveyed the basics of exact, approximation, and heuristic techniques for combinatorial optimization problems. Special attention has been devoted to the most applied metaheuristics. Among these, we highlighted path-relinking, tabu search, simulated annealing, genetic algorithms and population-based heuristics, variable neighborhood search and variable neighborhood descent, iterated local search, very large scale neighborhood local search.

References

[1] R.K. Ahuja, O. Ergun, J.B. Orlin, and A.P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Appl. Math.*, 123:75–102, 2002.

[2] R.K. Ahuja, J.B. Orlin, and D. Sharma. Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem. *Mathematical Programming*, 91:71–97, 2001.

[3] R.K. Ahuja, J.B. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research*, 27:917–934, 2000.

[4] A.M. Alvarez, J.L. Gonzalez-Velarde, and K. De Alba. GRASP embedded scatter search for the multicommodity capacitated network design problem. *J. of Heuristics*, 11:233–257, 2005.

[5] E.B. Baum. Iterated descent: A better algorithm for local search in combinatorial optimization problems. Technical report, California Institute of Technology, 1986.

[6] R. Bellman. *Dynamic programming.* Princeton University Press, 1957.

[7] J.D. Beltrán, J.E. Calderón, R.J. Cabrera, J.A.M. Pérez, and J.M. Moreno-Vega. GRASP/VNS hybrid for the strip packing problem. In *Proceedings of Hybrid Metaheuristics (HM2004)*, pages 79–90, 2004.

[8] D.P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, 3rd edition edition, 2005.

[9] S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 38:50–58, 2001.

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

[11] M. Dorigo and T. Stützle. *Ant Colony Optimization.* MIT Press, 2004.

[12] D. Hochbaum (editor). *Approximation algorithms for* NP*-hard problems.* PWS Publishing, 1996.

[13] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[14] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *J. of Global Optimization*, 6:109–133, 1995.

[15] P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized heuristics for the MAX-CUT problem. *Optimization Methods and Software*, 7:1033–1058, 2002.

[16] 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.

[17] 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.

[18] P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys on Metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

[19] P. Festa and M.G.C. Resende. Hybrid GRASP heuristics. *Studies in Computational Intelligence*, 203:75–100, 2009.

[20] P. Festa and M.G.C. Resende. GRASP: Basic components and enhancements. *Telecommunication Systems*, 46(3):253–271, 2011.

[21] P. Festa and M.G.C. Resende. Hybridizations of GRASP with path-relinking. *Studies in Computational Intelligence*, 434:135–155, 2013.

[22] M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* W.H. Freeman and Company, New York, 1979.

[23] F. Glover. Tabu search – Part I. *ORSA J. on Computing*, 1:190–206, 1989.

[24] F. Glover. Tabu search – Part II. *ORSA J. on Computing*, 2:4–32, 1990.

[25] F. Glover. Tabu search and adaptive memory programing – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

[26] F. Glover. Multi-start and strategic oscillation methods – Principles to exploit adaptive memory. In M. Laguna and J.L. Gonzáles-Velarde, editors, *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pages 1–24. Kluwer, 2000.

[27] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, 1997.

[28] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

[29] F. Glover, M. Laguna, and R. Martí. Scatter Search. In A. Ghosh and S. Tsutsui, editors, *Advances in Evolutionary Computation: Theory and Applications*, pages 519–537. Kluwer Academic Publishers, 2003.

[30] D.E Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley, 1989.

[31] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, *Meta-heuristics, Advances and trends in local search paradigms for optimization*, pages 433–458. Kluwer Academic Publishers, 1998.

[32] P. Hansen and N. Mladenović. Developments of variable neighborhood search. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 415–439. Kluwer Academic Publishers, 2002.

[33] J.P. Hart and A.W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.

[34] J.H. Holland. *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, Ann Arbor, Mich., USA, 1975.

[35] T. Ibaraki. Enumerative approaches to combinatorial optimization. *Annals of Operations Research*, 10:3–342, 1988.

[36] S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *J. of Statistical Physics*, 34:975–986, 1984.

[37] J.R. Koza, F.H. Bennett III, D. Andre, and M.A. Keane. *Genetic Programming III, Darwinian Invention and Problem Solving.* Morgan Kaufmann Publishers, 1999.

[38] M. Laguna. Scatter Search. In P.M. Pardalos and M.G.C. Resende, editors, *Handbook of Applied Optimization*, pages 183–193. Oxford University Press, 2002.

[39] M. Laguna and R. Martí. *Scatter Search: Methodology and Implementations in C.* Kluwer, 2003.

[40] E.L. Lawler and D.E. Wood. Branch-and-Bound Methods: A Survey. *Operations Research*, 14(4):699–719, 1966.

[41] H.R. Lourenço, O.C. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, 2003.

[42] M. Minoux. *Mathematical Programming: Theory and Algorithms.* Wiley, 1986.

[43] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

[44] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, 1988.

[45] I.H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:513, 1996.

[46] C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: Algorithms and complexity.* Prentice-Hall, 1982.

[47] P.M. Pardalos and M.G.C. Resende, editors. *Handbook of Applied Optimization.* Oxford University Press, 2002.

[48] M.G.C. Resende, R. Martí, M. Gallego, and A. Duarte. GRASP and path relinking for the max-min diversity problem. Technical report, AT&T Labs Research, Florham Park, NJ–USA, 2008.

[49] M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2003.

[50] M.G.C. Resende and C.C. Ribeiro. A hybrid heuristic for the p-median problem. *J. of Heuristics*, 10:59–88, 2004.

[51] M.G.C. Resende and R.F. Werneck. A hybrid multistart heuristic for the uncapacitated facility location problem. *European J. of Operational Research*, 174:54–68, 2006.

[52] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

[53] J.F. Shapiro. *Mathematical Programming: Structures and Algorithms*. Wiley, 1979.

[54] M. Sniedovich. *Dynamic Programming: Foundations and Principles*. Taylor & Francis, 2010.

[55] V. Vazirani. *Approximation algorithms*. Springer-Verlag, 2001.

[56] S. Voss, S. Martello, I.H. Osman, and C. Roucairo, editors. *Meta-heuristics: Andvances and trends in local search paradigms for optimization*. Kluwer, 1999.

[57] D.P. Williamson and D.B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

(Paola Festa) Department of Mathematics and Applications, University of Napoli FEDERICO II, Italy.

E-mail address: `paola.festa@unina.it`