GRASP: GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURES

MAURICIO G. C. RESENDE AND CELSO C. RIBEIRO

ABSTRACT. GRASP is a multi-start metaheuristic for combinatorial optimization problems, in which each iteration consists basically of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is investigated until a local minimum is found during the local search phase. The best overall solution is kept as the result. An intensification strategy based on path-relinking is frequently used to improve solution quality and to reduce computation times by exploring elite solutions previously found along the search. This chapter describes the basic components of GRASP, successful implementation strategies, and effective hybridizations with path-relinking and other metaheuristics. We also list some tricks to be used in the quest for good implementations. The bibliography is enriched by an account of relevant applications and by links to surveys, software, and additional sources of material.

1. Introduction

Metaheuristics are general high-level procedures that coordinate simple heuristics and rules to find good quality solutions to computationally difficult combinatorial optimization problems. Among them, we find simulated annealing (see Chapter 10), tabu search (see Chapter 9), genetic algorithms (see Chapter 4), scatter search (see Chapter 5), VNS (see Chapter 12), ant colonies (see Chapter 8), and others. The method described in this chapter represents another example of such a technique. Metaheuristics are based on distinct paradigms and offer different mechanisms to escape from locally optimal solutions. They are among the most effective solution strategies for solving combinatorial optimization problems in practice and they have been applied to a wide array of academic and real-world problems. The customization (or instantiation) of a metaheuristic to a given problem yields a heuristic for that problem.

In this chapter, we consider the combinatorial optimization problem of minimizing $f(S)$ over all solutions $S \in X$, which is defined by a finite set $E = \{e_1, \ldots, e_n\}$ (called the ground set), by a set of feasible solutions $X \subseteq 2^E$, and by an objective function $f : 2^E \rightarrow \mathbb{R}$. The ground set $E$, the objective function $f$, and the constraints defining the set of feasible solutions $X$ are defined and specific for each problem. We seek an optimal solution $S^* \in X$ such that $f(S^*) \leq f(S)$, $\forall S \in X$.

GRASP, which stands for Greedy Randomized Adaptive Search Procedures [48, 49], is a multi-start, or iterative metaheuristic, in which each iteration consists of two phases: construction and local search. The construction phase builds a
solution. If this solution is not feasible, a repair procedure should be applied to attempt to achieve feasibility. If feasibility cannot be reached, it is discarded and a new solution is created. Once a feasible solution is obtained, its neighborhood is investigated until a local minimum is found during the local search phase. The best overall solution is kept as the result.

Principles and building blocks of GRASP, which are also common to other metaheuristics, are reviewed in Section 2. A template for the basic GRASP algorithm is described in Section 3. The GRASP with path-relinking heuristic is considered in Section 4, where different strategies for the efficient implementation of path-relinking are discussed. Hybridizations of GRASP with data mining and other metaheuristics are reviewed in Section 5. Recommendations and good problem-solving practices leading to more efficient implementations are presented in Section 6. Finally, the last section provides several sources of additional information, with references and links to literature surveys, annotated bibliographies and source codes, tools and software for algorithm evaluation and comparison, and accounts of applications and parallel implementations of GRASP.

2. Principles and building blocks

Several principles and building blocks appear as components common to GRASP and other metaheuristics. They are often blended using different strategies and additional features that distinguish one metaheuristic from another.

2.1. Greedy algorithms. In a greedy algorithm, solutions are progressively built from scratch. At each iteration, a new element from the ground set \( E \) is incorporated into the partial solution under construction, until a complete feasible solution is obtained. The selection of the next element to be incorporated is determined by the evaluation of all candidate elements according to a greedy evaluation function. This greedy function usually represents the incremental increase in the cost function due to the incorporation of this element into the partial solution under construction. The greediness criterion establishes that an element with the smallest incremental increase is selected, with ties being arbitrarily broken. Figure 1 provides a template for a greedy algorithm for a minimization problem.

```
procedure GreedyAlgorithm
1. \( S \leftarrow \emptyset \);
2. Initialize the candidate set: \( C \leftarrow E \);
3. Evaluate the incremental cost \( c(e) \) for all \( e \in C \);
4. while \( C \neq \emptyset \) do
5. \quad Select an element \( s \in C \) with the smallest incremental cost \( c(s) \);
6. \quad Incorporate \( s \) into the current solution: \( S \leftarrow S \cup \{s\} \);
7. \quad Update the candidate set \( C \);
8. \quad Reevaluate the incremental cost \( c(e) \) for all \( e \in C \);
9. end;
10. return \( S \);
end.
```

Figure 1. Greedy algorithm for minimization.
The solutions obtained by greedy algorithms are not necessarily optimal. Greedy algorithms are often used to build initial solutions to be explored by local search or metaheuristics.

2.2. Randomization and greedy randomized algorithms. Randomization plays a very important role in algorithm design. Metaheuristics such as simulated annealing, GRASP, and genetic algorithms rely on randomization to sample the search space. Randomization can also be used to break ties, enabling different trajectories to be followed from the same initial solution in multistart methods, or sampling different parts of large neighborhoods. One particularly important use of randomization appears in the context of greedy algorithms.

Greedy randomized algorithms are based on the same principle guiding pure greedy algorithms. However, they make use of randomization to build different solutions at different runs. Figure 2 illustrates the pseudo-code of a greedy randomized algorithm for minimization. At each iteration, the set of candidate elements is formed by all elements that can be incorporated into the partial solution under construction without destroying feasibility. As before, the selection of the next element is determined by the evaluation of all candidate elements according to a greedy evaluation function. The evaluation of the elements by this function leads to the creation of a restricted candidate list (RCL) formed by the best elements, i.e. those whose incorporation into the current partial solution results in the smallest incremental costs. The element to be incorporated into the partial solution is randomly selected from those in the RCL. Once the selected element has been incorporated into the partial solution, the set of candidate elements is updated and the incremental costs are reevaluated.

```
procedure GreedyRandomizedAlgorithm(Seed)
1. \( S \leftarrow \emptyset \);
2. Initialize the candidate set: \( C \leftarrow E \);
3. Evaluate the incremental cost \( c(e) \) for all \( e \in C \);
4. \( \text{while } C \neq \emptyset \text{ do} \)
5. \hspace{1em} Build a list with the candidate elements having the smallest incremental costs;
6. \hspace{1em} Select an element \( s \) from the restricted candidate list at random;
7. \hspace{1em} Incorporate \( s \) into the solution: \( S \leftarrow S \cup \{ s \} \);
8. \hspace{1em} Update the candidate set \( C \);
9. \hspace{1em} Reevaluate the incremental cost \( c(e) \) for all \( e \in C \);
10. \hspace{1em} \text{end};
11. \hspace{1em} return \( S \);
end.
```

**Figure 2.** Greedy randomized algorithm for minimization.

Greedy randomized algorithms are used for a variety of purposes. For example, they are used in the construction phase of GRASP heuristics or to create initial solutions for population-based metaheuristics such as genetic algorithms or scatter search. Randomization is also a major component of metaheuristics, such as simulated annealing and VNS, in which a solution in the neighborhood of the current solution is randomly generated at each iteration.
2.3. **Neighborhoods.** A *neighborhood* of a solution $S$ is a set $N(S) \subseteq X$. Each solution $S' \in N(S)$ is reached from $S$ by an operation called a *move*. Normally, two neighbor solutions $S$ and $S' \in N(S)$ differ by only a few elements. Neighborhoods may also eventually contain infeasible solutions not in $X$.

A solution $S^*$ is a local optimum with respect to a given neighborhood $N$ if $f(S^*) \leq f(S), \forall S \in N(S^*)$. Local search methods are based on the exploration of solution neighborhoods, searching for improving solutions until a local optimum is found.

The definition of a neighborhood is not unique. Some implementations of metaheuristics make use of multiple neighborhood structures. A metaheuristic may also modify the neighborhood, by excluding some of the possible moves and introducing others. Such modifications might also require changes in the nature of solution evaluation. The strategic oscillation approach [61, 62] illustrates this intimate relationship between changes in neighborhood and changes in evaluation.

2.4. **Local search.** Solutions generated by greedy algorithms are not necessarily optimal, even with respect to simple neighborhoods. A local search technique attempts to improve solutions in an iterative fashion, by successively replacing the current solution by a better solution in a neighborhood of the current solution. It terminates when no better solution is found in the neighborhood. The pseudo-code of a basic local search algorithm for a minimization problem is given in Figure 3.

It starts from a solution $S$ and makes use of a neighborhood structure $N$.

```
procedure LocalSearch(S)
1. while $S$ is not locally optimal do
2. Find $S' \in N(S)$ with $f(S') < f(S)$;
3. $S \leftarrow S'$;
4. end;
5. return $S$;
end.
```

**Figure 3.** Local search algorithm for minimization.

The effectiveness of a local search procedure depends on several aspects, such as the neighborhood structure, the neighborhood search technique, the speed of evaluation of the cost function, and the starting solution. The neighborhood search may be implemented using either a best-improving or a first-improving strategy. In the case of a *best-improving* strategy, all neighbors are investigated and the current solution is replaced by the best neighbor. In the case of a *first-improving* strategy, the current solution moves to the first neighbor whose cost function value is smaller than that of the current solution.

2.5. **Restricted neighborhoods and candidate lists.** Glover and Laguna [63] point out that the use of strategies to restrict neighborhoods and to create candidate lists is essential to restrict the number of solutions examined in a given iteration in situations where the neighborhoods are very large or their elements are expensive to evaluate.

Their goal consists of attempting to isolate regions of the neighborhood containing desirable features and inserting them into a list of candidates for close
examination. The efficiency of candidate list strategies can be enhanced by the use of memory structures for efficient updates of move evaluations from one iteration to another. The effectiveness of a candidate list strategy should be evaluated in terms of the quality of the best solution found in some specified amount of computation time. Strategies such as aspiration plus, elite candidate list, successive filtering, sequential fan candidate list, and bounded change candidate list are reviewed in [63]. Ribeiro and Souza [127] used a candidate list strategy, based on quickly computed estimates of move values, to significantly speedup the search for the best neighbor in their tabu search heuristic for the Steiner problem in graphs. Moves with bad estimates were discarded. Restricted neighborhoods based on filtering out unpromising solutions with high evaluations are discussed, e.g., in [84, 117].

2.6. Intensification and diversification. Two important components of metaheuristics are intensification and diversification. Intensification strategies encourage move combinations and solution features historically found to be good or to return to explore attractive regions of the solution space more thoroughly. The implementation of intensification strategies enforces the investigation of neighborhoods of elite solutions and makes use of explicit memory to do so. Intensification is often implemented in GRASP heuristics by using path-relinking, as described below.

Diversification strategies encourage the search to examine unvisited regions of the solution space or to generate solutions that significantly differ from those previously visited. Penalty and incentive functions are often used in this context. Diversification is often implemented by means of perturbations which destroy the structure of the current solution. In the context of GRASP, they are used, for example, within hybridizations with the iterated local search (ILS) metaheuristic, as described in Section 5.

2.7. Path-relinking. Path-relinking was originally proposed by Glover [61] as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search [62, 64]. Starting from one or more elite solutions, paths in the solution space leading toward other elite solutions are generated and explored in the search for better solutions. To generate paths, moves are selected to introduce attributes in the current solution that are present in the elite guiding solution. Path-relinking may be viewed as a strategy that seeks to incorporate attributes of high quality solutions, by favoring these attributes in the selected moves.

The algorithm in Figure 4 illustrates the pseudo-code of the path-relinking procedure applied to a pair of solutions $S_s$ (starting solution) and $S_t$ (target solution). The procedure starts by computing the symmetric difference $\Delta(S_s, S_t)$ between the two solutions, i.e. the set of elements of the ground set $E$ that appear in one of them but not in the other. The symmetric difference also defines the set of moves that have to be successively applied to $S_s$ until $S_t$ is reached. At each step, the procedure examines all moves $m \in \Delta(S, S_t)$ from the current solution $S$ and selects the one which results in the least cost solution, i.e. the one which minimizes $f(S \oplus m)$, where $S \oplus m$ is the solution resulting from applying move $m$ to solution $S$. The best move $m^*$ is made, producing solution $S \oplus m^*$. The set of available moves is updated. If necessary, the best solution $\bar{S}$ is updated. The procedure terminates when $S_t$ is reached, i.e. when $\Delta(S, S_t) = \emptyset$. A path of solutions is thus generated linking $S_s$ to $S_t$ and $\bar{S}$ is the best solution in this path. Since there is no guarantee
that $\bar{S}$ is a local minimum, local search can be applied to it and the resulting local minimum is returned by the algorithm.

Path-relinking may also be viewed as a constrained local search strategy applied to the initial solution $S_s$, in which only a limited set of moves can be performed and uphill moves are allowed. Several alternatives have been considered and combined in successful implementations of path-relinking in conjunction with GRASP and other metaheuristics. They are reviewed in Section 4.

3. A template for GRASP

Each iteration of the original GRASP metaheuristic proposed in [48] may be divided in two main phases: construction and local search (see also [49, 108, 116, 117, 118, 120] for other surveys on GRASP and its extensions). These steps are repeated many times, characterizing a multistart metaheuristic. The construction phase builds a solution. If this solution is not feasible, it is either discarded or a repair heuristic is applied to achieve feasibility (examples of repair procedures can be found in [41, 42, 89, 93]). Once a feasible solution is obtained, its neighborhood is investigated until a local minimum is found during the local search phase. The best solution found over all iterations is returned.

The pseudo-code in Figure 5 illustrates the main blocks of a GRASP procedure for minimization, in which $\text{MaxIterations}$ iterations are performed and $\text{Seed}$ is used as the initial seed for the pseudo-random number generator.

An especially appealing characteristic of GRASP is the ease with which it can be implemented. Few parameters need to be set and tuned, and therefore development can focus on implementing efficient data structures to assure quick iterations. Basic implementations of GRASP rely exclusively on two parameters: the number $\text{MaxIterations}$ of iterations and the parameter used to limit the size of the

```plaintext
procedure PathRelinking($S_s, S_t$)
1. Compute the symmetric difference $\Delta(S_s, S_t)$;
2. $f \leftarrow \min \{f(S_s), f(S_t)\}$;
3. $\bar{S} \leftarrow \text{argmin}\{f(S_s), f(S_t)\}$;
4. $S \leftarrow S_s$;
5. while $\Delta(S, S_t) \neq \emptyset$ do
6. $m^* \leftarrow \text{argmin}\{f(S \oplus m) : m \in \Delta(S, S_t)\}$;
7. $\Delta(S \oplus m^*, S_t) \leftarrow \Delta(S, S_t) \setminus \{m^*\}$;
8. $S \leftarrow S \oplus m^*$;
9. if $f(S) < f$ then
10. $\bar{f} \leftarrow f(S)$;
11. $\bar{S} \leftarrow S$;
12. end if;
13. end while;
14. $\hat{S} \leftarrow \text{LocalSearch}(\bar{S})$;
15. return $\hat{S}$;
end.

Figure 4. Path-relinking procedure for minimization.
```
Figure 5. Template of a GRASP heuristic for minimization.

restricted candidate list within the greedy randomized algorithm used by the construction phase. In spite of its simplicity and ease of implementation, GRASP is a very effective metaheuristic and produces the best known solutions for many problems, see [57, 58, 59] for extensive surveys of applications of GRASP.

For the construction of the RCL used in the first phase we consider, without loss of generality, a minimization problem such as the one formulated in Section 1. As before, we denote by $c(e)$ the incremental cost associated with the incorporation of element $e \in E$ into the solution under construction. At any GRASP iteration, let $c_{\text{min}}$ and $c_{\text{max}}$ be, respectively, the smallest and the largest incremental costs.

The restricted candidate list is made up of the elements $e \in E$ with the best (i.e., the smallest) incremental costs $c(e)$. This list can be limited either by the number of elements (cardinality-based) or by their quality (value-based). In the first case, it is made up of the $p$ elements with the best incremental costs, where $p$ is a parameter. In this chapter, the RCL is associated with a threshold parameter $\alpha \in [0, 1]$. The restricted candidate list is formed by all “feasible” elements $e \in E$ which can be inserted into the partial solution under construction without destroying feasibility and whose quality is superior to the threshold value, i.e., $c(e) \in [c_{\text{min}}, c_{\text{min}} + \alpha(c_{\text{max}} - c_{\text{min}})]$. The case $\alpha = 0$ corresponds to a pure greedy algorithm, while $\alpha = 1$ is equivalent to a random construction. The pseudo code in Figure 6 is a refinement of the greedy randomized construction algorithm, whose pseudo-code appears in Figure 2.

GRASP may be viewed as a repetitive sampling technique. Each iteration produces a sample solution from an unknown distribution, whose mean value and variance are functions of the restrictive nature of the RCL. The pseudo code in Figure 6 shows that the parameter $\alpha$ controls the amounts of greediness and randomness in the algorithm. Resende and Ribeiro [117, 120] have shown that what often leads to good solutions are relatively low average solution values (i.e., close to the value of the purely greedy solution obtained with $\alpha = 0$) in the presence of a
Figure 6. Refined pseudo-code of the construction phase using parameter $\alpha$ for defining a quality threshold.

```
procedure GreedyRandomizedConstruction($\alpha$, Seed)
1. $S \leftarrow \emptyset$;
2. Initialize the candidate set: $C \leftarrow E$;
3. Evaluate the incremental cost $c(e)$ for all $e \in C$;
4. while $C \neq \emptyset$ do
5.   $c_{min} \leftarrow \min\{c(e) \mid e \in C\}$;
6.   $c_{max} \leftarrow \max\{c(e) \mid e \in C\}$;
7.   Build the restricted candidate list: $RCL \leftarrow \{e \in C \mid c(e) \leq c_{min} + \alpha(c_{max} - c_{min})\}$;
8. Choose $s$ at random from $RCL$;
9. Incorporate $s$ into solution: $S \leftarrow S \cup \{s\}$;
10. Update the candidate set $C$;
11. Reevaluate the incremental cost $c(e)$ for all $e \in C$;
end.
12. return $S$;
end.
```

relatively large variance (i.e., solutions obtained with a larger degree of randomness as $\alpha$ increases), such as is often the case for $\alpha = 0.2$.

Prais and Ribeiro [103] showed that using a single fixed value for the value of the RCL parameter $\alpha$ often hinders finding a high-quality solution, which eventually could be found if another value was used. An alternative is to use a different value of $\alpha$, chosen uniformly at random in the interval $[0, 1]$, at each GRASP iteration. Prais and Ribeiro [103] proposed another alternative, the Reactive GRASP extension of the basic procedure, in which the parameter $\alpha$ is self-tuned and its value is periodically modified according with the quality of the solutions previously obtained. Applications to other problems (see e.g. [58, 120]) have shown that Reactive GRASP outperforms the basic algorithm. These results motivated the study of the behavior of GRASP for different strategies for the variation of the value of the RCL parameter $\alpha$. The experiments reported in [103] show that implementation strategies based on the variation of $\alpha$ are likely to be more affective than one using a single fixed value for this parameter.

Two other randomized greedy approaches, with smaller worst-case complexities than that depicted in the pseudo-code of Figure 6 were proposed in [121]. Instead of combining greediness and randomness at each step of the construction procedure, the random plus greedy scheme applies randomness during the first $p$ construction steps to produce a random partial solution. Next, the algorithm completes the solution with one or more pure greedy construction steps. By changing the value of the parameter $p$, one can control the balance between greediness and randomness in the construction: larger values of $p$ correspond to solutions that are more random, with smaller values corresponding to greedier solutions. The sampled greedy construction provides a different way to combine randomness and greediness. This procedure is also controlled by a parameter $p$. At each step of the construction process, the procedure builds a restricted candidate list by sampling $\min\{p, |C|\}$ elements of the candidate set $C$. Each of the sampled elements is evaluated by the
greedy function and an element with the smallest greedy function value is added to the partial solution. These steps are repeated until there are no more candidate elements. As before, the balance between greediness and randomness can be controlled by changing the value of the parameter $p$, i.e. the number of candidate elements that are sampled. Small sample sizes lead to more random solutions, while large sample sizes lead to more greedy solutions.

4. GRASP WITH PATH-RELINKING

GRASP, as originally proposed, is a memoryless procedure in which each iteration does not make use of information gathered in previous iterations. Path-relinking is a major enhancement used for search intensification with GRASP. By adding memory structures to the basic procedure described above, path-relinking leads to significant improvements in solution time and quality.

The basic principles of path-relinking were described in Section 2.7. The use of path-relinking within a GRASP procedure was proposed in [78] and followed by extensions, improvements, and successful applications (see Section 7). Surveys of GRASP with path-relinking can be found in [116, 118, 120]. Different schemes have been proposed for the implementation of path-relinking. In essence, it has been applied as a post-optimization phase (between every pair of elite solutions in the pool of elite solutions) and as an intensification strategy (between every local optimum obtained after the local search phase and one or more elite solutions in the pool of elite solutions).

In this last context, path-relinking is applied to pairs of solutions, one of which is a locally optimal solution and the other is randomly chosen from a pool with a limited number $\text{MaxElite}$ of elite solutions found along the search. A simple strategy is to assign equal probabilities of being selected to each elite solution. Another strategy assigns probabilities proportional to the cardinality of the symmetric difference between the elite solution and the locally optimal solution. This strategy favors elite solutions that result in longer paths. One of these solutions is called the initial solution, while the other is the guiding solution. One or more paths in the solution space graph connecting these solutions may be explored in the search for better solutions. The pool of elite solutions is originally empty. Since we wish to maintain a pool of good but diverse solutions, each locally optimal solution obtained by local search is considered as a candidate to be inserted into the pool if it is sufficiently different from every other solution currently in the pool. If the pool already has $\text{MaxElite}$ solutions and the candidate is better than the worst of them, then a simple strategy is to have the candidate replace the worst elite solution. This strategy improves the quality of the elite set. Another strategy is to have the candidate replace an elite solution with worse objective function value that is most similar to it. This strategy improves the diversity of the elite set as well as its quality.

The pseudo-code in Figure 7 illustrates the main steps of a GRASP procedure using path-relinking to implement a memory-based intensification strategy.

Several alternatives for applying path-relinking to a pair of solutions $S$ and $S'$ have been considered and combined in the literature. These include forward, backward, back and forward, mixed, truncated, greedy randomized adaptive, and evolutionary path-relinking. All these alternatives involve trade-offs between computation time and solution quality.
procedure GRASPwithPathRelinking(MaxIterations, Seed)
1. Set $f^* \leftarrow \infty$;
2. Set $Pool \leftarrow \emptyset$;
3. for $k = 1, \ldots, \text{MaxIterations}$ do
4. $S \leftarrow \text{GreedyRandomizedAlgorithm}(\text{Seed})$;
5. if $S$ is infeasible then
6. $S \leftarrow \text{RepairSolution}(S)$;
7. endif;
8. $S \leftarrow \text{LocalSearch}(S)$;
9. if $k > 1$ then
10. Randomly select a solution $S' \in Pool$;
11. $S \leftarrow \text{PathRelinking}(S', S)$;
12. endif;
13. if $f(S) < f^*$ then
14. $S^* \leftarrow S$;
15. $f^* \leftarrow f(S)$;
16. endif;
17. Update $Pool$ with $S$ if it satisfies the membership conditions;
18. end for;
19. return $S^*$;
end.

Figure 7. Template of a GRASP with path-relinking heuristic for minimization.

In forward path-relinking, the GRASP local optimum $S$ is designated as the initial solution and the pool solution $S'$ is made the guiding solution. The roles of $S$ and $S'$ are interchanged in backward path-relinking. This scheme was originally proposed in Aiex et al. [7], Ribeiro et al. [128], and Resende and Ribeiro [116]. The main advantage of this approach over forward path-relinking comes from the fact that, in general, there are more high-quality solutions near pool elements than near GRASP local optima. Backward path-relinking explores more thoroughly the neighborhood around the pool solution, whereas forward path-relinking explores more thoroughly the neighborhood around the GRASP local optimum. Experiments in [7, 116] have confirmed that backward path-relinking usually outperforms forward path-relinking. Back and forward path-relinking combines forward and backward path-relinking, exploring two different paths. It finds solutions at least as good as forward path-relinking or backward path-relinking, but at the expense of taking about twice as long to run. Mixed path-relinking shares the benefits of back and forward path-relinking, in about the same time as forward or backward path-relinking alone. This is achieved by interchanging the roles of the initial and guiding solutions at each step of the path-relinking procedure. Ribeiro and Rosseti [126] have shown experimentally that it outperforms forward, backward, and back and forward path-relinking (see also [120]).

Other strategies have been proposed more recently. Truncated path-relinking can be applied to either forward, backward, back and forward, or mixed path-relinking. Instead of exploring the entire path, it takes only a fraction of those steps and consequently takes a fraction of the time to run. Since high-quality solutions tend
to be near the initial or guiding solutions, exploring part of the path near the extremities may produce solutions about as good as those found by exploring the entire path. Indeed, Resende et al. [111] showed experimentally that this is the case for instances of the max-min diversity problem. Greedy randomized adaptive path-relinking, introduced by Binato et al. [27], is a semi-greedy version of path-relinking. Instead of taking the best move in the symmetric difference still not performed, a restricted candidate list of good moves still not performed is set up and a randomly selected move from the RCL is applied. By applying this strategy several times between the initial and guiding solutions, several alternative paths can be explored. Resende and Werneck [121, 122] described an evolutionary path-relinking scheme applied to pairs of elite solutions and used as a post-optimization phase, in which the pool resulting from the GRASP with path-relinking iterations progressively evolves as a population. Similar schemes were also used in [7, 111].

5. Extensions

Hybridizations of GRASP with metaheuristics such as tabu search, simulated annealing, variable neighborhood search, iterated local search, and genetic algorithms have been reported in the literature.

Almost all the randomization effort in GRASP involves the construction phase, since the local search always stops at the first local optimum. VNS (Variable Neighborhood Search, see Chapter 12) relies almost entirely on the randomization of the local search to escape from local optima. Thus, GRASP and VNS may be considered as complementary and potentially capable of leading to effective hybrid methods. Festa et al. [56] studied different variants and combinations of GRASP and VNS for the MAX-CUT problem, finding and improving the best known solutions for some open instances in the literature. Other examples of hybrids of GRASP with VNS include [26, 32].

GRASP has also been used in conjunction with genetic algorithms. Basically, the greedy randomized strategy used in the construction phase of a GRASP is applied to generate the initial population for a genetic algorithm. We may cite, e.g., the genetic algorithm of Ahuja et al. [5] for the quadratic assignment problem, which makes use of the GRASP proposed by Li et al. [80] to create the initial population of solutions. A similar approach was used by Armony et al. [19], with the initial population made up of both randomly generated solutions and those built by a GRASP.

The hybridization of GRASP with tabu search was first studied by Laguna and González-Velarde [77]. Delmaire et al. [39] considered two approaches. In the first, GRASP is applied as a powerful diversification strategy in the context of a tabu search procedure. The second approach is an implementation of the Reactive GRASP algorithm, in which the local search phase is strengthened by tabu search. Two two-stage heuristics are proposed in [1] for solving the multi-floor facility layout problem. GRASP/TS applies a GRASP to find the initial layout and tabu search to refine it. Souza et al. [137] used a short-term tabu search procedure as a substitute for the standard local search in a GRASP heuristic for the capacitated minimum spanning tree problem.

Iterated local search (ILS) iteratively builds a sequence of solutions generated by the repeated application of local search and perturbation of the local optimum found by local search [25, 81, 83]. Ribeiro and Urrutia [129] presented a GRASP...
with ILS heuristic for the mirrored traveling tournament problem. In this case, the GRASP construction produces a solution which is passed on to the ILS procedure. The hybridization of GRASP with data mining techniques was introduced in [130]. This scheme uses a data mining algorithm to search for solution patterns that occur in high-quality elite solutions produced by the basic GRASP algorithm. These mined patterns are used as initial building blocks that guide the construction of new solutions that are submitted to local search. A survey of applications of DM-GRASP can be found in [134].

6. Tricks of the trade

(1) An especially appealing characteristic of GRASP is the ease with which it can be implemented. Few parameters need to be set and tuned. Therefore, algorithm development and coding can focus on implementing efficient data structures to ensure quick GRASP iterations.

(2) Most metaheuristics benefit from good initial solutions. Clever low-complexity algorithms leading to good feasible solutions can often be devised by examination of the problem structure. Good initial solutions lead to better final solutions and significantly reduce the time taken by local search.

(3) Using a single, fixed value for the restricted candidate list parameter $\alpha$ very often hinders finding a high-quality solution, which eventually could be found if another value was used. The use of strategies such as Reactive GRASP which vary the value of $\alpha$ may lead to better and more diverse solutions. The reactive approach leads to improvements over the basic GRASP in terms of robustness and solution quality, due to greater diversification and less reliance on parameter tuning. In addition to the original applications reported in [103, 104], it has also been applied in [13, 28, 29, 31, 39, 135]. Another simple strategy is to uniformly select at random a value for $\alpha$ at each GRASP iteration from the interval $[0, 1]$.

(4) Local search procedures may be implemented using a best-improving or a first-improving strategy, as well as any combination of them. In the case of the best-improving strategy, all neighbors are investigated and the current solution is replaced by the best neighbor. In the case of a first-improving strategy, the current solution moves to the first neighbor whose cost function value is smaller than that of the current solution. Both strategies quite often lead to same quality solutions, but in smaller computation times when the first-improving strategy is used. Premature convergence to a non-global local minimum is more likely to occur with a best-improving strategy.

(5) The definition of a neighborhood is not unique. Some implementations of metaheuristics make use of multiple neighborhood structures to improve solution quality and to speed up the search. Variable neighborhood descent (VND) allows the systematic exploration of multiple neighborhoods [67]. It 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. Furthermore, VND is also based on the empirical observation that, for many problems, local minima with respect to one or more neighborhoods are relatively close to each other [68]. 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. In the case of nested neighborhoods, the search is first confined to smaller neighborhoods. A larger neighborhood is explored only after a local minimum is found in the current, smaller neighborhood. Neighborhoods are not necessarily nested. Non-nested neighborhoods have been successfully used, e.g., by Aloise et al. [10]).

(6) Local search can be considerably accelerated with the use of appropriate data structures and efficient algorithms. All possible attempts should be made to improve the neighborhood search procedure. Algorithms should be coded to have minimum complexity. The use of circular lists to represent and search the neighborhood is very helpful. Candidate lists storing the move values may be easy to update or may be used as quick approximations to avoid their reevaluation at every iteration. We have seen several implementations in which the time taken by the first local search code dropped from several minutes to a few milliseconds in the final version.

(7) Path-relinking is a very effective strategy to improve solution quality and to reduce computation times, leading to more robust implementations. Any available knowledge about the problem structure should be used in the development of efficient algorithms to explore the most attractive strategy for path-relinking.

(8) Different metaheuristics make use of a number of common components, such as greedy constructions, local search, randomization, candidate lists, multiple neighborhoods, path-relinking, etc. Borrowing and incorporating principles from other metaheuristics lead to efficient hybridizations of GRASP, which often results in the best algorithm for some problem class.

(9) There is no universal, general purpose metaheuristic that gives the best results for every problem [141] (see Chapter 16). The structure of each problem should be explored to bring additional intelligence into the solution strategy. Knowledge, experience, and information available in the literature for similar problems are very helpful. However, one should not be obsessed with a fixed idea or bounded by strategies that worked for other problems but might not be appropriate for the one on hand. The best algorithm is always the one that most exploits the structure of your problem and gives the best results.

7. Sources of additional information

Surveys on GRASP [49, 117, 120], path-relinking [118], and its applications [57, 58, 59] can be found in the literature, to where the interested reader is referred for more details and references.


Time-to-target (TTT) plots display on the ordinate axis the probability that an algorithm will find a solution at least as good as a given target value within a given running time, shown on the abscissa axis. TTT plots were used by Feo,
Resende, and Smith [50] and have been advocated also by Hoos and Stützle [72, 71] as a way to characterize the running times of stochastic algorithms for combinatorial optimization. Aiex et al. [8] advocate and largely explored the use of TTT plots to evaluate and compare different randomized algorithms running on the same problem. The use of TTT plots has been growing ever since and they have been extensively applied in computational studies of sequential and parallel implementations of randomized algorithms (see, e.g., [117, 120, 126]. The foundations of the construction of time-to-target plots, together with their interpretation and applications, were surveyed by Aiex et al. [9]. This reference also describes a Perl language program to create time-to-target plots for measured CPU times that can be downloaded from http://www.research.att.com/~mgcr/tttplots.

The first application of GRASP described in the literature concerned the set covering problem [48]. GRASP has been applied to many problems in different areas, such as routing [18, 21, 36, 75, 105]; logic [40, 54, 98, 109, 113, 114]; covering and partitioning [12, 17, 48, 65]; location [1, 33, 37, 66, 69, 39, 73, 96, 139, 140]; minimum Steiner tree [32, 84, 85, 88, 128]; optimization in graphs [2, 3, 4, 20, 50, 55, 56, 70, 76, 79, 82, 85, 100, 107, 110, 115, 124, 128, 137]; assignment [5, 7, 47, 60, 80, 90, 91, 92, 94, 97, 99, 102, 104, 112, 133]; timetabling, scheduling, and manufacturing [6, 13, 11, 15, 22, 23, 24, 28, 31, 35, 45, 46, 51, 52, 74, 77, 129, 131, 132, 142, 143]; transportation [18, 45, 47, 135, 136]; power systems [29, 30, 44]; telecommunications [2, 14, 15, 34, 73, 101, 104, 106, 107, 116, 125, 138]; graph and map drawing [37, 53, 78, 82, 95, 115, 124]; biology [16, 43]; and VLSI [17], among others.

GRASP is a metaheuristic very well suited for parallel implementation, due to the independence of its iterations. Parallel cooperative versions of GRASP with path-relinking may also be implemented in parallel if a centralized pool of elite solutions is kept by one of the processors. Surveys and accounts of parallel implementations of GRASP in networks of workstations, clusters, and grids may be found in [38, 86, 87, 119, 123, 125, 126].

References


(Mauricio G. C. Resende) Algorithms and Optimization Research Department, AT&T Labs Research, 180 Park Avenue, Room C241, Florham Park, NJ 07932 USA.  
E-mail address: mgcr@research.att.com

(Celso C. Ribeiro) Department of Computer Science, Universidade Federal Fluminense, Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil.  
E-mail address: celso@ic.uff.br