Hybrid Metaheuristics: An Introduction

Christian Blum[1] and Andrea Roli[2]

[1] ALBCOM research group
Universitat Politècnica de Catalunya, Barcelona, Spain
cblum@lsi.upc.edu
[2] DEIS, Campus of Cesena
Alma Mater Studiorum Università di Bologna, Bologna, Italy
andrea.roli@unibo.it

Summary. In many real life settings, high quality solutions to hard optimization problems such as flight scheduling or load balancing in telecommunication networks are required in a short amount of time. Due to the practical importance of optimization problems for industry and science, many algorithms to tackle them have been developed. One important class of such algorithms are metaheuristics. The field of metaheuristic research has enjoyed a considerable popularity in the last decades. In this introductory chapter we first provide a general overview on metaheuristics. Then we turn towards a new and highly successful branch of metaheuristic research, namely the hybridization of metaheuristics with algorithmic components originating from other techniques for optimization. The chapter ends with an outline of the remaining book chapters.

1 Introduction

In the context of combinatorial optimization (CO), algorithms can be classified as either *complete* or *approximate* algorithms. Complete algorithms are guaranteed to find for every finite size instance of a CO problem an optimal solution in bounded time (see [80, 76]). Yet, for CO problems that are $\mathcal{NP}$-hard [42], no polynomial time algorithm exists, assuming that $\mathcal{P} \neq \mathcal{NP}$. Therefore, complete methods might need exponential computation time in the worst-case. This often leads to computation times too high for practical purposes. In approximate methods such as metaheuristics we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time. Thus, the use of metaheuristics has received more and more attention in the last 30 years. This was also the case in continuous optimization; due to other reasons: Metaheuristics are usually easier to implement than classical gradient-based techniques. Moreover, metaheuristics do not require gradient information. This is convenient for optimization problems where the objective function is only implicitly given (e.g.,

C. Blum and A. Roli: *Hybrid Metaheuristics: An Introduction*, Studies in Computational Intelligence (SCI) **114**, 1–30 (2008)
www.springerlink.com

when objective function values are obtained by simulation), or where the objective function is not differentiable.

The first two decades of research on metaheuristics were characterized by the application of rather standard metaheuristics. However, in recent years it has become evident that the concentration on a sole metaheuristic is rather restrictive. A skilled combination of a metaheuristic with other optimization techniques, a so called *hybrid metaheuristic*, can provide a more efficient behavior and a higher flexibility when dealing with real-world and large-scale problems. This can be achieved, for instance, by combining the complementary strengths of metaheuristics on one side and of complete methods such as branch & bound techniques or mathematical programming on the other side. In general, hybrid metaheuristic approaches can be classified as either *collaborative combinations* or *integrative combinations* (see [85, 86]). Collaborative combinations are based on the exchange of information between several optimization techniques run sequentially (or in parallel). This kind of combination is more related to cooperative and parallel search and we forward the interested reader to the specific literature on the subject [55, 24, 105, 98, 19, 18, 3]. Most contributions of this book deal with interesting and representative cases of integrative combinations.

In this introductory chapter we first aim at giving an overview over some of the most important metaheuristics. Then we deal with the hybridization of metaheuristics with other techniques for optimization. Finally, we shortly outline the books' contents, with the aim of providing a general guidance to the reader.

2 Basics

An optimization problem $\mathcal{P}$ can be described as a triple $(\mathcal{S}, \Omega, f)$, where

1. $\mathcal{S}$ is the search space defined over a finite set of decision variables X_i, $i = 1, \ldots, n$. In case these variables have discrete domains we deal with discrete optimization (or combinatorial optimization), and in case of continuous domains $\mathcal{P}$ is called a continuous optimization problem. Mixed variable problems also exist. Ω is a set of constraints among the variables;
2. $f : \mathcal{S} \rightarrow I\!R^+$ is the objective function that assigns a positive cost value to each element (or solution) of $\mathcal{S}$.

The goal is to find a solution $s \in \mathcal{S}$ such that $f(s) \leq f(s')$, $\forall\ s' \in \mathcal{S}$ (in case we want to minimize the objective function), or $f(s) \geq f(s')$, $\forall\ s' \in \mathcal{S}$ (in case the objective function must be maximized). In real-life problems the goal is often to optimize several objective functions at the same time. This form of optimization is labelled multi-objective optimization.

Approximate methods are generally based on two basic principles: *constructive heuristics* and *local search methods*.

2.1 Constructive Heuristics

Constructive heuristics are typically the fastest approximate methods. They generate solutions from scratch by adding opportunely defined solution components to an initially empty partial solution. This is done until a solution is complete or other stopping criteria are satisfied. A well-known class of constructive heuristics are *greedy heuristics*. They make use of a weighting function that assigns at each construction step a positive weight to each feasible solution component. The solution component with the highest weight is chosen at each construction step to extend the current partial solution. An example of a greedy heuristic is the *nearest neighbor heuristic* for the famous traveling salesman problem (TSP) [57].

2.2 Local Search Methods

Constructive heuristics are often very fast, yet they often return solutions of inferior quality when compared to local search algorithms. Local search algorithms start from some initial solution and iteratively try to replace the current solution by a better solution in an appropriately defined neighborhood of the current solution, where the neighborhood is formally defined as follows.

Definition 1. *A* **neighborhood structure** *is a function $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ that assigns to every $s \in \mathcal{S}$ a set of neighbors $\mathcal{N}(s) \subseteq \mathcal{S}$. $\mathcal{N}(s)$ is called the neighborhood of s. Often, neighborhood structures are implicitly defined by specifying the changes that must be applied to a solution s in order to generate all its neighbors. The application of such an operator that produces a neighbor $s' \in \mathcal{N}(s)$ of a solution s is commonly called a* **move**.

A neighborhood structure together with a problem instance define the topology of a so-called search (or fitness) landscape [100, 61, 40]. A search landscape can be visualized as a labelled graph in which the nodes are solutions (labels indicate their objective function value) and arcs represent the neighborhood relation between solutions. A solution $s^* \in \mathcal{S}$ is called a *globally minimal solution*[1] (or global minimum) if for all $s \in \mathcal{S}$ it holds that $f(s^*) \leq f(s)$. The set of all globally minimal solutions is henceforth denoted by $\mathcal{S}^*$. The introduction of a neighborhood structure enables us to additionally define the concept of *locally* minimal solutions.

Definition 2. *A* **locally minimal solution (or local minimum)** *with respect to a neighborhood structure $\mathcal{N}$ is a solution $\hat{s}$ such that $\forall\ s \in \mathcal{N}(\hat{s}) : f(\hat{s}) \leq f(s)$. We call $\hat{s}$ a strict locally minimal solution if $f(\hat{s}) < f(s)\ \forall\ s \in \mathcal{N}(\hat{s})$.*

[1] Without loss of generality, in the remainder of this chapter we restrict the discussion to minimization problems.

Algorithm 1 Iterative improvement local search

1: $s \leftarrow$ GenerateInitialSolution()
2: **while** $\exists\, s' \in \mathcal{N}(s)$ such that $f(s') < f(s)$ **do**
3: $\quad s \leftarrow$ ChooseImprovingNeighbor($\mathcal{N}(s)$)
4: **end while**

The most basic local search method is usually called *iterative improvement local search*, since each move is only performed if the resulting solution is better than the current solution. The algorithm stops as soon as it has reached a local minimum. The high level algorithm is sketched in Alg. 1.

Function ChooseImprovingNeighbor($\mathcal{N}(s)$) can be implemented in several ways. In the following we present the two most typical ones. The first way is called *first-improvement*. A first-improvement function scans the neighborhood $\mathcal{N}(s)$ and returns the first solution that is better than s. In contrast, a *best-improvement* function exhaustively explores the neighborhood and returns one of the solutions with the lowest objective function value. An iterative improvement procedure that uses a first-improvement function is called *first-improvement local search*, respectively *best-improvement local search* (or steepest descent local search) in the case of a best-improvement function. Both methods stop at local minima. Therefore, their performance strongly depends on the definition of the neighborhood structure $\mathcal{N}$. A deterministic iterative improvement local search algorithm partitions the search space $\mathcal{S}$ into so-called *basins of attraction* of local minima [84, 92]. The basin of attraction of a local minimum $\hat{s} \in \mathcal{S}$ is the set of all solutions s for which the search terminates in $\hat{s}$ when started from the initial solution s.[2]

2.3 Metaheuristics

In the 70ies, a new kind of approximate algorithm has emerged which tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. These methods are nowadays commonly called *metaheuristics*.[3] The term *metaheuristic*, first introduced in [45], derives from the composition of two Greek words. *Heuristic* derives from the verb *heuriskein* ($\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\nu$) which means "to find", while the suffix *meta* means "beyond, in an upper level". Before this term was widely adopted, metaheuristics were often called *modern heuristics* [88].

This class of algorithms includes[4]—but is not restricted to—ant colony optimization (ant colony optimization), evolutionary computation (EC) including

[2] The definition can also be extended to the case of stochastic local search algorithms.

[3] The increasing importance of metaheuristics is underlined by the biannual Metaheuristics International Conference (MIC). The 7th was held in Montreal in June 2007 (*www.crt.umontreal.ca/mic2007/*).

[4] In alphabetical order.

genetic algorithms (GAs), iterated local search (ILS), simulated annealing (SA), and tabu search (TS). Due to the generality of the metaheuristic concept it is hardly possible to give a precise definition of what a metaheuristic exactly is. In the following we quote some characterizations that appeared in the literature:

"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." I. Osman and G. Laporte in [78].

"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." S. Voß et al. in [109].

"Metaheuristics are typically high-level strategies which guide an underlying, more problem specific heuristic, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descent by allowing the local search to escape from local minima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more "intelligent" way than just providing random initial solutions. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). Many of the metaheuristic approaches rely on probabilistic decisions made during the search. But, the main difference to pure random search is that in metaheuristic algorithms randomness is not used blindly but in an intelligent, biased form." Stützle in [101].

"A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem." Metaheuristics Network at [26].

In short, we may characterize metaheuristics as high level strategies for exploring search spaces by using different methods. Of great importance for the functioning of a metaheuristic are the concepts called *diversification* and *intensification*. The term diversification generally refers to the exploration

of the search space, whereas the term intensification refers to the exploitation of the accumulated search experience. Each metaheuristic application is characterized by a balance between diversification and intensification. This is important, on one side to quickly identify regions in the search space with high quality solutions, and on the other side not to waste too much time in regions of the search space which are either already explored or which do not provide high quality solutions.

There are different ways to classify and describe metaheuristic algorithms, each of them being the result of a specific viewpoint. For example, we might classify metaheuristics as *nature-inspired metaheuristics vs. non-nature inspired metaheuristics.* This classification is based on the origins of the different algorithms. There are nature-inspired algorithms, such as evolutionary computation and ant colony optimization, and non nature-inspired ones such as tabu search and iterated local search. We might also classify metaheuristics as *memory-based vs. memory-less methods.* This classification scheme refers to the use metaheuristics make of the search history, that is, whether they use memory or not. Memory-less algorithms, for example, perform a Markov process, as the information they exclusively use to determine the next action is the current state of the search process. The use of memory is nowadays recognized as one of the fundamental elements of a powerful metaheuristic. Finally, metaheuristics may also be classified into methods that perform a *single point vs. population-based search.* This classification refers to the number of solutions used by a metaheuristic at any time. Generally, algorithms that work on a single solution at any time are referred to as *trajectory methods*. They comprise all metaheuristics that are based on local search, such as tabu search, iterated local search and variable neighborhood search. They all share the property that the search process describes a trajectory in the search space. Population-based metaheuristics, on the contrary, either perform search processes which can be described as the evolution of a set of points in the search space (as for example in evolutionary computation), or they perform search processes which can be described as the evolution of a probability distribution over the search space (as for example in ant colony optimization).

3 Overview on Important Metaheuristics

In the following, we outline the main principles of some of the most important metaheuristics. However, an introduction to such a vast research area has to focus on certain aspects and therefore has unfortunately to neglect other aspects. We refer the interested reader to [114, 49, 33, 93, 110] for works that deal also with other aspects such as, for example, software libraries. Furthermore, we refer to [11, 57] for a more detailed introduction to metaheuristics.

3.1 Metaheuristics Based on Local Search

The performance of simple iterative improvement local search procedures (see Sect. 2.2) is in general unsatisfactory. The quality of the obtained local minimum heavily depends on the starting point for the local search process. As the basin of attraction of a global minimum is generally not known, iterative improvement local search might end up in a poor quality local minimum. A first simple strategy of extending iterative improvement local search consists in the iterative application of the local search starting at each iteration from a different starting point, which may be randomly generated. For this type of multi-start local search it is sometimes possible to obtain theoretical performance guarantees. However, they are usually still far from being satisfactory (see, for example, [94]).

Therefore, several metaheuristic techniques have been developed with the aim of adding an *exploration* component to iterative improvement local search. This exploration component is responsible for guiding the exporation of the search space in the search for better and better local minima. Obviously, these algorithms need termination criteria other than simply reaching a local minimum. Commonly used termination criteria are a maximum CPU time, a maximum number of iterations, a solution s of sufficient quality is found, or a maximum number of iterations without improvement. In the following we present some of the most important local-search based metaheuristics.

Simulated Annealing

Simulated annealing (SA) is commonly said to be the oldest among the metaheuristics and was one of the first algorithms that had an explicit strategy to escape from local minima. The origins of the algorithm are in statistical mechanics (see the Metropolis algorithm [70]). The idea of SA is inspired by the annealing process of metal and glass, which assume a low energy configuration when first heated up and then cooled down sufficiently slowly. SA was first presented as a search algorithm for CO problems in [63] and [14]. The fundamental idea is to allow moves to solutions with objective function values that are worse than the objective function value of the current solution. This kind of move is often called uphill move.

The algorithmic framework of SA is described in Alg. 2. It works as follows. The algorithm starts by generating an initial solution in function GenerateInitialSolution(). The initial solution may be randomly or heuristically constructed. Then, the initial temperature value is determined in function SetInitialTemperature() such that the probability for an uphill move is quite high at the start of the algorithm. At each iteration a solution $s' \in \mathcal{N}(s)$ is randomly chosen in function PickNeighborAtRandom($\mathcal{N}(s)$). If s' is better than s (i.e., has a lower objective function value), then s' is accepted as new current solution. Otherwise, if the move from s to s' is an uphill move, s' is accepted with a probability which is a function of a temperature parameter T_k and

Algorithm 2 Simulated annealing (SA)

```
1: s ← GenerateInitialSolution()
2: k ← 0
3: T_k ← SetInitialTemperature()
4: while termination conditions not met do
5:    s' ← PickNeighborAtRandom(N(s))
6:    if (f(s') < f(s)) then
7:       s ← s'          {s' replaces s}
8:    else
9:       Accept s' as new solution with probability p(s' | T_k, s)  {see Equation 1}
10:   end if
11:   k ← k + 1
12:   T_k ← AdaptTemperature()
13: end while
```

$f(s') - f(s)$. Usually this probability is computed following the Boltzmann distribution:

$$\mathbf{p}(s' \mid T_k, s) = e^{-\frac{f(s')-f(s)}{T_k}} \quad . \tag{1}$$

The temperature T_k is adapted at each iteration according to a *cooling schedule* (or cooling scheme) in function AdaptTemperature(). The cooling schedule defines the value of T_k at each iteration k. The choice of an appropriate cooling schedule is crucial for the performance of the algorithm. At the beginning of the search the probability of accepting uphill moves should be high. Then, this probability should be gradually decreased during the search. Note that this is not necessarily done in a monotonic fashion.

The dynamic process described by basic SA is a *Markov chain* [30], because the choice of the next solution exclusively depends on the current solution, that is, the algorithm does not use memory. Nevertheless, the use of memory can be beneficial for SA approaches (see for example [15]). Interestingly, theoretical results on non-homogeneous Markov chains [1] state that under particular conditions on the cooling schedule, SA converges in probability to a global minimum for $k \to \infty$.

For representative applications of SA we refer the interested reader to [17, 107, 2, 59, 34]. Interesting variants of SA are Threshold Accepting, respectively the Great Deluge Algorithm [29, 28], and Extremal Optimization [12].

Tabu Search

The basic ideas of tabu search (TS) were introduced in [45], based on earlier ideas formulated in [44].[5] A description of the method and its concepts can be found in [47].

[5] Related ideas were labelled steepest ascent/mildest descent method in [50].

Algorithm 3 Tabu search (TS)

1: $s \leftarrow$ GenerateInitialSolution()
2: InitializeTabuLists($TL_1, \ldots, TL_r$)
3: **while** termination conditions not met **do**
4: $\quad \mathcal{N}_a(s) \leftarrow \{s' \in \mathcal{N}(s) \mid s'$ does not violate a tabu condition, or it satisfies at least one aspiration condition$\}$
5: $\quad s' \leftarrow \text{argmin}\{f(s'') \mid s'' \in \mathcal{N}_a(s)\}$
6: $\quad$ UpdateTabuLists($TL_1, \ldots, TL_r, s, s'$)
7: $\quad s \leftarrow s'$ $\quad$ {i.e., s' replaces s}
8: **end while**

The basic idea of TS is the explicit use of search history, both to escape from local minima and to implement a strategy for exploring the search space. A basic TS algorithm (see Alg. 3) uses *short term memory* in the form of so-called *tabu lists* to escape from local minima and to avoid cycles.[6] In standard TS algorithms tabu lists are implemented in a FIFO (first in first out) manner. The tabu lists generally store features of recently visited solutions. The algorithm may work with a different tabu list for each type of considered solution feature. At the start of the algorithm the tabu lists are initialized as empty lists in function InitializeTabuLists($TL_1, \ldots, TL_r$). For performing a move, the algorithm first determines those solutions from the neighborhood $\mathcal{N}(s)$ of the current solution s that contain solution features currently to be found in the tabu lists. These solutions are said to violate the tabu conditions. They are excluded from the neighborhood, resulting in a restricted set of neighbors $\mathcal{N}_a(s)$. However, note that storing only features of solutions allows the possibility that unvisited solutions of high quality are excluded from the set of neighbors. To overcome this problem, *aspiration criteria* are defined which allow to include a solution in the restricted set of neighbors even though it violates a tabu condition. The most commonly used aspiration criterion applies to solutions which are better than the best solution found so far. At each iteration the best solution s' from $\mathcal{N}_a(s)$ is chosen as the new current solution. Furthermore, in procedure UpdateTabuLists($TL_1, \ldots, TL_r, s, s'$) the corresponding features of this solution are added to the tabu lists and—in case the tabu lists have reached their maximally allowed length—the oldest solution features are deleted. The algorithm stops when a termination condition is met.

In general, the use of a tabu list prevents from returning to recently visited solutions, and may force the search to accept even uphill moves. The length l of a tabu list—known in the literature as the *tabu tenure*—controls the memory of the search process. With small tabu tenures the search will concentrate on small areas of the search space. On the opposite, a large tabu tenure forces the search process to explore larger regions, because it forbids

[6] A cycle is a sequence of moves that constantly repeats itself.

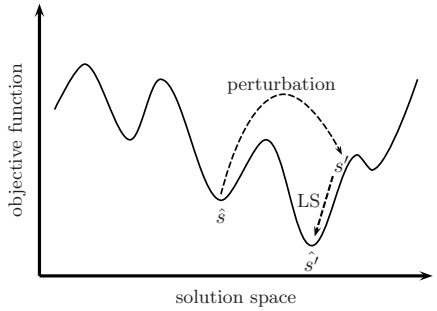


Fig. 1. A desirable ILS step: the current solution $\hat{s}$ is perturbed, then local search is applied to the perturbed solution s' and a new (even better) local minimum $\hat{s'}$ is found.

revisiting a higher number of solutions. The tabu tenure can be varied during the search, leading to more robust algorithms. An example can be found in [103], where the tabu tenure is periodically reinitialized at random from the interval $[l_{min}, l_{max}]$. A more advanced use of a dynamic tabu tenure is presented in [7, 6], where the tabu tenure is increased in case of evidence for repetitions of solutions (thus a higher diversification is needed), while it is decreased in case of no improvements (thus intensification should be boosted). More advanced ways of applying dynamic tabu tenures are described in [46].

The interested reader can find representative applications of TS in [103, 6, 21, 77, 43]. Further references to applications can be found in [47].

Iterated Local Search

Iterated local search (ILS) [101, 65, 64, 68] is a metaheuristic that is based on a simple but effective concept. Instead of repeatedly applying an iterative improvement local search to randomly generated starting solutions, an ILS algorithm produces the starting solution for the next iteration by perturbing the local minimum obtained by the previous application of iterative improvement local search. The hope is that the perturbation mechanism produces a solution that is located in the basin of attraction of a local minimum that is better than the current solution, and that is close to the current solution. Fig. 1 shows such a situtation graphically.

The pseudo-code of ILS is shown in Alg. 4. It works as follows. First, an initial solution is generated in function GenerateInitialSolution(). This solution is subsequently improved by the application of a local search method in function LocalSearch(s). The construction of initial solutions should be fast (computationally not expensive), and—if possible—initial solutions should be a good starting point for local search. The fastest way of producing an initial solution is often to generate it at random. Another possibility is to use

Algorithm 4 Iterated local search (ILS)

```
1: s ← GenerateInitialSolution()
2: ŝ ← LocalSearch(s)
3: while termination conditions not met do
4:    s' ← Perturbation(ŝ, history)
5:    ŝ' ← LocalSearch(s')
6:    ŝ ← ApplyAcceptanceCriterion(ŝ', ŝ, history)
7: end while
```

constructive heuristics such as greedy heuristics. At each iteration, the current solution $\hat{s}$ is perturbed in function Perturbation($\hat{s}$,*history*), resulting in a perturbed solution s'. The perturbation is usually non-deterministic in order to avoid cycling. The importance of the perturbation mechanism is obvious: a perturbation that is not strong enough might not enable the algorithm to escape from the basin of attraction of the current solution. On the other side, a perturbation that is too strong would make the algorithm similar to a random restart local search. The requirement on the perturbation method is to produce a starting point for local search such that a local minimum different from the current solution is reached. However, this new local minimum should be *closer* to the current solution than a local minimum produced by the application of the local search to a randomly generated solution. After the application of local search to the perturbed solution, the resulting solution $\hat{s'}$ may either be accepted as new current solution, or not. This is decided in function ApplyAcceptanceCriterion($\hat{s'}, \hat{s}, history$). Two extreme examples are (1) accepting the new local minimum only in case of improvement and (2) always accepting the new solution. Inbetween, there are several possibilities. For example, it is possible to adopt an acceptance criterion that is similar to the one of simulated annealing, like non-monotonic cooling schedules which exploit the history of the search process. For example, when the recent history of the search process indicates that intensification seems no longer effective, a diversification phase is needed and the solution reached after the local search phase is always accepted, or the probability to accept it is increased.

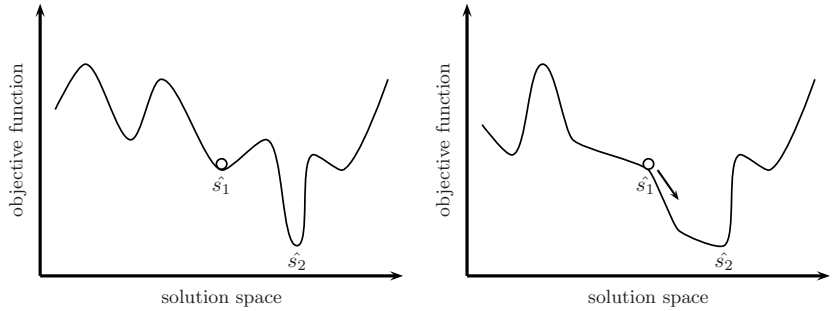
For examples of successful applications of ILS we refer the interested reader to [67, 60, 65, 22]. References to other applications can be found in [65].

Other Metaheuristics Based on Local Search

Besides the metaheuristics outlined above, there are some other ones that are based on local search. In the following we present shortly their basic principles.

Variable neighborhood search (VNS).

This metaheuristic was proposed in [52, 53]. The main idea of VNS is based on the fact that a change of the neighborhood structure changes the shape of the



(a) Search landscape 1 (b) Search landscape 2

Fig. 2. Two search landscapes defined by two different neighborhood structures. On the landscape that is shown in (a), the best-improvement local search stops at $\hat{s_1}$, while it proceeds till a better local minimum $\hat{s_2}$ on the landscape that is shown in (b).

search landscape. In fact, a local minimum with respect to a neighborhood function $\mathcal{N}_1$ is not necessarily a local minimum with respect to a different neighborhood function $\mathcal{N}_2$ (see Fig. 2 for an example).

Based on this observation, the main idea of VNS is to define more than one neighborhood structure, and to swap between different neighborhood structures in a strategic way during the search process. Observe that the process of changing neighborhoods (for example, in case of no improvements) corresponds to a diversification of the search. The effectiveness of VNS can be explained by the fact that a "bad" place on the search landscape given by a certain neighborhood structure could be a "good" place on the search landscape given by another neighborhood structure.[7] Moreover, a solution that is locally optimal with respect to a neighborhood is probably not locally optimal with respect to another neighborhood. Concerning applications of VNS we refer the interested reader to [51, 91, 113, 106]. More references can be found in [53].

Guided local search (GLS).

The strategy applied by GLS (see [112, 111]) for exploring the search space is conceptually very different to the strategies that are employed by the other local search based metaheuristics. This strategy consists in dynamically changing the objective function by *penalizing* solution features that occur frequently in visited solutions. The use of penalties aims at increasing the objective function value of solutions that contain these features. This change in the objective function results in a change of the search landscape. The aim is to make the current local minimum gradually "less desirable" over time in order to be

[7] A "good" place in the search space is an area from which a good local minimum can be reached.

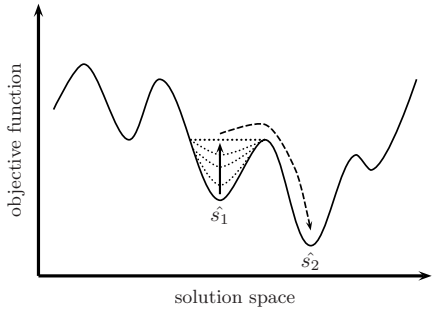


Fig. 3. Basic GLS idea: escaping from a valley in the landscape by increasing the objective function value of its solutions.

able to move to other areas of the search space. A pictorial description of this idea is given in Fig. 3. The interested reader is referred to [73, 62, 112] for applications of GLS.

Greedy randomized adaptive search procedure (GRASP).

This metaheuristic (see [31, 81]) is a simple technique that combines constructive heuristics with local search. At each iteration, a solution to the tackled problem is constructed in a randomized way. Subsequently the constructed solution is improved by the application of a local search algorithm. GRASP is a metaheuristic that does not make use of the search history. This is one of the reasons why GRASP is often outperformed by other metaheuristics. However, due to its simplicity, it is generally very fast and it is able to produce quite good solutions in a very short amount of computation time. It can be effective if—at least—two conditions are satisfied: (1) the solution construction mechanism samples the most promising regions of the search space, and (2) the solutions constructed by the constructive heuristic belong to basins of attraction of different local minima. Representative applications of GRASP include [8, 90, 82]. A detailed and annotated bibliography references many more applications [32].

3.2 Population-Based Metaheuristics

Population-based metaheuristics deal at each algorithm iteration with a set of solutions rather than with a single solution. From this set of solutions the population of the next iteration is produced by the application of certain operators. Population-based metaheuristics provide a natural, intrinsic way for the exploration of the search space. Yet, the final performance strongly depends on the way the population is manipulated. The most studied population-based methods are evolutionary computation (EC) and ant colony optimization (ant colony optimization). In EC algorithms, a population of individuals is modified by recombination and mutation operators, and in ant colony optimization a

colony of artificial ants is used to construct solutions guided by the pheromone trails and heuristic information.

Ant Colony Optimization

Ant colony optimization (ant colony optimization) [27, 9] is a metaheuristic approach that was inspired by the foraging behavior of real ants. Their way of foraging—as described by Deneubourg et al. in [23]—enables ants to find shortest paths between food sources and their nest. Initially, ants explore the area surrounding their nest in a random manner. As soon as an ant finds a source of food, it evaluates quantity and quality of the food and carries some of this food to the nest. During the return trip, the ant deposits a chemical pheromone trail on the ground. The quantity of pheromone deposited, which may depend on the quantity and quality of the food, will guide other ants to the food source. The indirect communication between the ants via the pheromone trails allows them to find shortest paths between their nest and food sources. This functionality of real ant colonies is exploited in artificial ant colonies in order to solve hard optimization problems.

In ant colony optimization algorithms the chemical pheromone trails are simulated via a parametrized probabilistic model that is called the *pheromone model*. It consists of a set of model parameters whose values are called the *pheromone values*. These values act as the memory that keeps track of the search process. The basic ingredient of ant colony optimization algorithms is a constructive heuristic that is used for probabilistically constructing solutions using the pheromone values. In general, the ant colony optimization approach attempts to solve a CO problem by iterating the following two steps:

- Solutions are constructed using a pheromone model, that is, a parametrized probability distribution over the solution space.
- The constructed solutions and possibly solutions that were constructed in earlier iterations are used to modify the pheromone values in a way that is deemed to bias future sampling toward high quality solutions.

A very general pseudo-code for the ant colony optimization metaheuristic is shown in Alg. 5. It consists of three algorithmic components that are gathered in the ScheduleActivities construct. At each iteration, the algorithm probabilistically generates a number of solutions to the tackled problem in function AntBasedSolutionConstruction() by assembling them from a set of solution components. Subsequently, the pheromone values are modified in function PheromoneUpdate(). A standard pheromone update consists of two parts. First, a *pheromone evaporation*, which uniformly decreases all the pheromone values, is performed. From a practical point of view, pheromone evaporation is needed to avoid a too rapid convergence of the algorithm toward a sub-optimal region. It implements a useful form of *forgetting*, favoring the exploration of new areas in the search space. Then, one or more solutions from the current and/or from earlier iterations are used to increase the values of

Algorithm 5 Ant colony optimization (ant colony optimization)

```
1: while termination conditions not met do
2:   ScheduleActivities
3:     AntBasedSolutionConstruction()
4:     PheromoneUpdate()
5:     DaemonActions()        {optional}
6:   end ScheduleActivities
7: end while
```

pheromone trail parameters on solution components that are part of these solutions. Other types of pheromone update are rather optional and mostly aim at the intensification or the diversification of the search process. Daemon actions (see function DaemonActions()) can be used to implement centralized actions; in contrast to the localized decision making of the solution construction process. Examples are the application of local search methods to the constructed solutions, or the collection of global information that can be used to decide whether it is useful or not to deposit additional pheromone to bias the search process from a non-local perspective. As a practical example, the daemon may decide to deposit extra pheromone on the solution components that belong to the best solution found so far.

In general, different versions of ant colony optimization algorithms mostly differ in the way they update the pheromone values. This also holds for two of the currently best-performing ant colony optimization variants in practice, which are Ant Colony System (ACS) [25] and $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [102].

We refer the interested reader to [41, 69, 10, 96] for applications of ACO algorithms. Further references to applications of ant colony optimization can be found in [27].

Evolutionary Computation

Evolutionary computation (EC) can be regarded as a metaphor for building, applying, and studying algorithms based on Darwinian principles of natural selection. The instances of algorithms that are based on evolutionary principles are called evolutionary algorithms (EAs) ([5]). EAs can be characterized as computational models of evolutionary processes. They are inspired by nature's capability to evolve living beings well adapted to their environment. At the core of each EA is a population of individuals. At each algorithm iteration a number of *reproduction* operators is applied to the individuals of the current population to generate the individuals of the population of the next generation. EAs might use operators called *recombination* or *crossover* to recombine two or more individuals to produce new individuals. They also can use *mutation* or *modification* operators which cause a self-adaptation of individuals. The driving force in EAs is the *selection* of individuals based on their *fitness*

Algorithm 6 Evolutionary computation (EC)

```
1: P ← GenerateInitialPopulation()
2: Evaluate(P)
3: while termination conditions not met do
4:    P' ← Recombine(P)
5:    P'' ← Mutate(P')
6:    Evaluate(P'')
7:    P ← Select(P'',P)
8: end while
```

(which might be based on the objective function, the result of a simulation experiment, or some other kind of quality measure). Individuals with a higher fitness have a higher probability to be chosen as members of the population of the next generation (or as parents for the generation of new individuals). This corresponds to the principle of *survival of the fittest* in natural evolution. It is the capability of nature to adapt itself to a changing environment, which gave the inspiration for EAs.

Alg. 6 shows the basic structure of EC algorithms. In this algorithm, P denotes the population of individuals. A population of offspring is generated by the application of *recombination* and *mutation* operators and the individuals for the next population are *selected* from the union of the old population and the offspring population.

There has been a variety of slightly different EAs proposed over the years. Three different strands of EAs were developed independently of each other over time. These are evolutionary programming (EP) as introduced by Fogel in [38] and Fogel et al. in [39], evolutionary strategies (ES) proposed by Rechenberg in [87] and genetic algorithms (GAs) initiated by Holland in [56] (see [48], [74], [89], and [108] for further references). EP arose from the desire to generate machine intelligence. While EP originally was proposed to operate on discrete representations of finite state machines, most of the present variants are used for continuous optimization problems. The latter also holds for most present variants of ES, whereas GAs are mainly applied to solve discrete problems. More recently, other members of the EA family such as for example genetic programming (GP) and scatter search (SS) were developed. Despite this subdivision into different strands, EAs can be understood from a unified point of view with respect to their main components and the way they explore the search space. Over the years there have been quite a few overviews and surveys about EC methods. Among those are the ones by Bäck [4], by Fogel et al. [36], by Kobler and Hertz [54], by Spears et al. [99], and by Michalewicz and Michalewicz [71]. In [13] a taxonomy of EAs is proposed.

EC algorithms have been applied to most CO problems and optimization problems in general. Recent successes are documented works such as [37, 16, 97]. For an extensive collection of references to EC applications we refer to [5].

4 Hybridization of Metaheuristics

The concept of hybrid metaheuristics has been commonly accepted only in recent years, even if the idea of combining different metaheuristic strategies and algorithms dates back to the 1980s. Today, we can observe a generalized common agreement on the advantage of combining components from different search techniques and the tendency of designing hybrid techniques is widespread in the fields of operations research and artificial intelligence. The consolidated interest around hybrid metaheuristics is also demonstrated by publications on classifications, taxonomies and overviews on the subject [86, 85, 104, 72].

In this book, we adopt the definition of hybrid metaheuristic in the broad sense of integration of a metaheuristic related concept with some other techniques (possibly another metaheuristic). We may distinguish between two categories: the first consists in designing a solver including components from a metaheuristic into another one, while the second combines metaheuristics with other techniques typical of fields such as operations research and artificial intelligence. A prominent representant of the first category is the use of trajectory methods into population based techniques or the use of a specific local search method into a more general trajectory method such as ILS. The second category includes hybrids resulting from the combination of metaheuristics with constraint programming (CP), integer programming (IP), tree-based search methods, data mining techniques, etc. Both categories contain countless instances and an exhaustive description is not possible. Nevertheless, we believe that a brief description of some notable examples could give the flavour of the ideas that characterize these forms of hybridization.

4.1 Component Exchange Among Metaheuristics

One of the most popular ways of metaheuristic hybridization consists in the use of trajectory methods inside population-based methods. Indeed, most of the successful applications of EC and ACO make use of local search procedures and the effectiveness of many memetic algorithms [75] relies indeed on this synergic integration. The reason for that becomes apparent when analyzing the respective strengths of trajectory methods and population-based methods.

The power of population-based methods is certainly to be found in the capability of recombining solutions to obtain new ones. In EC algorithms and scatter search explicit recombinations are implemented by one or more recombination operators. In ACO and EDAs recombination is implicit, because new solutions are generated by using a distribution over the search space which is a function of earlier populations. This enables the search process to perform a guided sampling of the search space, usually resulting in a coarse grained exploration. Therefore, these technique can effectively find promising areas of the search space.

The strength of trajectory methods is rather to be found in the way they explore a promising region in the search space. As in those methods local search is the driving component, a promising area in the search space is searched in a more structured way than in population-based methods. In this way the danger of being close to good solutions but "missing" them is not as high as in population-based methods. More formally, local search techniques efficiently drives the search toward the attractors, i.e., local optima or confined areas of the space in which many local optima are condensed.

In summary, population-based methods are better in identifying promising areas in the search space from which trajectory methods can quickly reach good local optima. Therefore, metaheuristic hybrids that can effectively combine the strengths of both population-based methods and trajectory methods are often very successful.

4.2 Integration of Metaheuristics With AI and OR Techniques

One of the most prominent research directions is the integration of metaheuristics with more classical artificial intelligence and operations research methods, such as CP and branch & bound or other tree search techniques. In the following we outline some of the possible ways of integration.

Metaheuristics and tree search methods can be sequentially applied or they can also be interleaved. For instance, a tree search method can be applied to generate a partial solution which will then be completed by a metaheuristic approach. Alternatively, metaheuristics can be applied to improve a solution generated by a tree-search method.

CP techniques can be used to reduce the search space or the neighborhood to be explored by a local search method. In CP, CO problems are modelled by means of variables, domains[8] and constraints, which can be mathematical (as for example in linear programming) or symbolic (also known as *global* constraints). Constraints encapsulate well defined parts of the problem into sub-problems, thus making it possible to design specialized solving algorithms for sub-problems that occur frequently. Every constraint is associated to a *filtering* algorithm that deletes values from a variable domain that do not contribute to feasible solutions.[9] Metaheuristics (especially trajectory methods) may use CP to efficiently explore the neighborhood of the current solution, instead of simply enumerating the neighbors or randomly sampling the neighborhood. A prominent example of such a kind of integration is Large Neighborhood Search [95] and related techniques. These approaches are effective mainly when the neighborhood to explore is very large, or when problems (such as many real-world problems) have additional constraints (usually called *side constraints*). A detailed overview of the possible ways of integration of CP and metaheuristics can be found in [35].

[8] We restrict the discussion to finite domains.

[9] The filtering could be either *complete* if the remaining domain values are guaranteed to be consistent, or *incomplete* otherwise.

Another possible combination consists in introducing concepts or strategies from either class of algorithms into the other. For example, the concepts of tabu list and aspiration criteria—known from Tabu search—can be used to manage the list of open nodes (i.e., the ones whose child nodes are not yet explored) in a tree search algorithm. Examples of these approaches can be found in [83, 20]. Tree-based search is also successfully integrated into ACO in [10], where beam search [79] is used for solution construction.

Integer and linear programming can be also effectively combined with metaheuristics. For instance, linear programming is often used either to solve a sub-problem or to provide dual information to a metaheuristic in order to select the most promising candidate solution or solution component [58, 66, 10].

The kinds of integration we shortly mentioned belong to the class of *integrative combinations* and are the main topic of this book. The other possible way of integration, called either *collaborative combinations* or also *cooperative search* consists in a loose form of hybridization, in that search is performed by possibly different algorithms that exchange information about states, models, entire sub-problems, solutions or search space characteristics. Typically, cooperative search algorithms consist of the parallel execution of search algorithms with a varying level of communication. The algorithms can be different or they can be instances of the same algorithm working on different models or running with different parameter settings. The algorithms composing a cooperative search system can be all approximate, all complete, or a mix of approximate and complete approaches. This area of research shares many issues with the design of parallel algorithms and we forward the interested reader to the specific literature on the subject [3, 55, 24, 105, 98].

5 Outline of the Book

The contributions collected in this book cover some of the main topics of hybrid metaheuristics. Most of the chapters are devoted to the integration of metaheuristics with other techniques from AI and OR, namely, mathematical programming, constraint programming and various combinations of complete and incomplete search techniques. Two additional chapters complement the collection by giving, respectively, an overview of hybrid metaheuristics for multi-objective problems and introducing multilevel refinement for enhancing the performance of standard metaheuristics. In the following we give an outline of the book by providing a short description of each chapter.

5.1 Chapter 2: Integer Linear Programming and Metaheuristics

The chapter by Raidl and Puchinger provides a comprehensive survey on the integration of metaheuristics and exact techniques such as integer linear

programming (ILP), cutting plane and column generation approaches, branch-and-cut, branch-and-price, and branch-and-cut-and-price. The authors point out that metaheuristics and exact approaches can be seen as complementary to a large degree, which makes it natural to combine ideas from both streams. They claim that hybrid optimizers are often significantly more effective in terms of running time and/or solution quality since they benefit from synergy. After discussing a structural classifications of strategies for combining metaheuristics and exact optimization techniques, the authors survey several types of different hybridization approaches, including the following ones:

1. Instead of simple heuristics, metaheuristics can be used for finding high-quality upper bounds within a branch & bound algorithm. Hereby, metaheuristics may be applied for deriving the initial solutions as well as deriving upper bounds for subproblems of the branch & bound tree.
2. Problem relaxations may be used for guiding the search process of a metaheuristic, because an optimal solution for a relaxation of the original problem often indicates in which areas of the original problems search space good or even optimal solutions may be found.
3. Generally, branch & bound algorithms choose the next tree node to be processed by a best-first strategy: choose a node with the smallest lower bound. However, with this strategy, high quality upper bounds are only found late in the search process. More sophisticated concepts aim to intensify branch & bound search—in the style of metaheuristics—in an initial phase to neighborhoods of promising incumbents in order to quickly identify high quality upper bounds.
4. In metaheuristics, candidate solutions are sometimes only indirectly or incompletely represented. In these situtations, an (intelligent) decoding function is needed in order to obtain complete solution to the problem at hand. This is the case, for example, in evolutionary algorithms and ant colony optimization. ILP techniques may successfully be used for the decoding step.
5. In cutting plane and column generation based methods the dynamic separation of cutting planes and the pricing of columns, respectively, is sometimes done by means of metaheuristics in order to speed up the optimization process.

5.2 Chapter 3: Relation Between Complete and Incomplete Search

An in-depth analysis of the relation between complete and incomplete search is the subject of the contribution by Prestwich. The author explores the boundaries between these two prototypical approaches for solving combinatorial and constraint satisfaction problems and discusses some possible integrations. The chapter starts with a description of the main complete and incomplete search techniques and then analyzes their peculiarities. Complete techniques explore exhaustively the whole search space and they are based on intelligent enumeration strategies which try to prune the search tree, for example by using

lower and upper bounds and nogoods. Incomplete search techniques are the ones which do not guarantee to find the optimal solution (in the case of a CO problem) or a feasible solution (in the case of a Constraint Satisfaction Problem (CSP)) in bounded time and metaheuristics are one of its most representative classes. The author points out that the boundaries between complete and incomplete search are quite blurred. Hybrid approaches which integrate metaheuristics into tree-search and viceversa are surveyed and strengths and weaknesses of the various techniques are discussed. Taking inspiration from the considerations between contrasting complete and incomplete search, in the second part of the chapter the author proposes a hybrid search scheme for CSPs called *Incomplete Dynamic Backtracking* (IDB). This technique tries to build a solution as done in dynamic backtracking style algorithms, i.e., it can backtrack to an already assigned variable without unassigning the intermediate ones, with the difference that the choice of variable(s) to backtrack to is completely free and there is no exhaustiveness to be guaranteed. IDB can also be viewed as a local search with a cost function given by the number of unassigned variables. This is an emblematic case of an algorithm laying the fuzzy border between complete and incomplete search.

5.3 Chapter 4: Hybridizations of Metaheuristics With Branch & Bound Derivates

In their chapter, Blum et al. give a closer look at two specific ways of hybridizing metaheuristics with branch & bound (derivatives). The first one concerns the use of branch & bound features within construction-based metaheuristics such as ant colony optimization or greedy randomized adaptive search procedures in order to increase their efficiency. In particular, the authors deal with the case of Beam-ACO, a hybrid algorithm that results from replacing the standard solution construction procedure of ant colony optimization with a probabilistic beam search, which is an incomplete branch & bound derivative. After explaining the algorithm in general terms, an application to the longest common subsequence problem is presented.

The second part of the chapter concerns the use of a memetic algorithm in order to increase the efficiency of branch & bound, respectively beam search. More specifically, the memetic algorithm is used to obtain upper bounds for open subproblems of the branch & bound tree. The quality of the resulting hybrid technique is demonstrated by means of the application to another classical string problem, the shortest common supersequence problem.

5.4 Chapter 5: Large Scale Neighborhood Search

The chapter by Chiarandini et al. presents some notable examples of so-called large scale neighborhood search. Local search techniques are characterized by following a trajectory in the state space, moving at each iteration from a solution s to a new one s' by means of a so-called move, that is, by choosing the

next solution s' in the *neighborhood* of s. Thus, the neighborhood of s is the set of candidates among which to choose the next solution. Small size neighborhoods are often preferred because of efficiency concerns, nevertheless they might make it difficult for the search to explore large portions of the search space or to move away from basins of attraction of local optima. In contrast, large scale neighborhoods, while enlarging the set of candidate (neighboring) solutions, can enable the search to enhance its exploration but at the price of higher computational time. The chapter by Chiarandini et al. discusses both exhaustive and heuristic algorithms for exploring large scale neighborhoods, in particular for the graph coloring problem and one of its extensions named graph set T-coloring problem. In the chapter, the authors emphasize the use of dynamic programming for the exploration of exponential size neighborhoods.

5.5 Chapter 6: Constructive Metaheuristics and Constraint Programming

The contribution by Meyer deals with another type of hybrid metaheuristic. More specifically, the author shows how constraint programming (CP) can be integrated with ACO. The key point in this study is that the construction phase in ACO can be performed by means of CP techniques; this approach can be particularly effective in problems in which the constraints make the search of a feasible solution hard, because in those cases the power of CP can be fully exploited. A dual, equivalent, perspective for combining CP and ACO is also discussed in which the learning mechanism of ACO is introduced in the variable/value selection mechanism during the labelling phase in CP. This phase consists in first choosing an unassigned variable and then selecting a value from the chosen variables' domain. This kind of tight integration of ACO and CP is compared against loose combination and pure versions of ACO and CP. The loose combination is implemented by (conceptually) running in parallel the two techniques and enabling a communication mechanism for exchanging partial solutions and bounds. Results on machine scheduling problem instances show that the tightly coupled approach is superior to the loosely coupled one and ACO and CP alone.

5.6 Chapter 7: Hybrid Metaheuristics for Packing Problems

The contribution by Ibaraki et al. shows the effectiveness of the integration of metaheuristics with several mathematical programming techniques on three variants of the two-dimensional packing problem. The problem consists in packing a set of items into a container with given size without overlaps between items. The authors consider first the formulation in which items are rectangular and have given and fixed size (in terms of width and height), then a variant in which sizes are adjustable within predefined limits and, finally, the problem with items without being restricted to assume a particular shape (also known as *irregular packing* or *nesting* problem). The point

made by the authors is that hybrid metaheuristics can profit from the recent advances in software for mathematical programming by exploiting dynamic, linear and nonlinear programming for solving to optimality some specific subproblems. Indeed, the approaches discussed in the chapter rely on solution coding schemes that make the search of good configurations easier by providing a mechanism for representing a set of solutions in a compact way. Such coding schemes require the definition of decoding algorithms to transform a coded solution into an actual and complete one. Solution decoding can be viewed as a subproblem that has to be solved a large number of times during search. Therefore, decoding algorithms should be as much efficient as possible. Ibaraki et al. show how metaheuristic methods, such as iterated local search, can be fruitfully combined with decoding algorithms based on dynamic and (non)linear programming.

5.7 Chapter 8: Hybrid Multi-objective Combinatorial Optimization

While most of the book deals with hybrid methods for combinatorial optimization problems with only one objective, the chapter by Ehrgott and Gandibleux presents an overview on hybrid techniques for multi-objective combinatorial optimization. In fact, many real world optimization problems can be modelled as combinatorial optimization problems with multiple and even conflicting objectives. The authors give among others the example of railway transportation, where the planning of railway network infrastructure has the goals of maximizing the number of trains that can use it and to maximize the robustness of solutions to disruptions in operation. After giving an overview over the most important non-hybrid metheuristic methods proposed for multi-objective optimization, the authors survey the currently existing hybrid approaches. Hereby the authors distinguish between hybridization approaches in order to make the search more aggressive, hybrization in order to drive a metaheuristic, hybridization for exploiting the complementary strength of different techniques, and hybridization with, for example, exact methods. Finally, the authors conclude their chapter with a section on current hybridization trends. One of these trends concerns the used of lower and upper bounds on the non-dominated frontier for deciding if an expensive local search procedure should be started from a solution, or not.

5.8 Chapter 9: The Multilevel Paradigm

Instead of dealing with a typical hybridization method, the chapter by Walshaw rather presents a framework for using metaheuristics and/or other optimization techniques in a potentially more efficient way. This framework is commonly referred to as the multilevel paradigm, or the multilevel method. Its application to combinatorial optimization problems is quite simple. Basically, it involves recursive coarsening to create a hierarchy of approximations to the

original problem. An initial solution is found, usually at the coarsest level, for example by some metaheuristic algorithm. Then this solution is iteratively refined at each level, coarsest to finest. The same metaheuristic used for finding a solution to the coarsest level may be used for this purpose. Solution extension (or projection) operators can transfer the solution from one level to another. While this strategy has been used for many years, for example, in multigrid techniques, its application in combinatorial optimization is rather new. This chapter gives a survey on recent developments in this directions.

References

1. E. H. L. Aarts, J. H. M. Korst, and P. J. M. van Laarhoven. Simulated annealing. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 91–120. John Wiley & Sons, Chichester, UK, 1997.
2. E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, UK, 1997.
3. E. Alba, editor. *Parallel Metaheuristics: A New Class of Algorithms*. John Wiley, 2005.
4. T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.
5. T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing Ltd, Bristol, UK, 1997.
6. R. Battiti and M. Protasi. Reactive Search, a history-base heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics*, 2:Article 2, 1997.
7. R. Battiti and G. Tecchiolli. The Reactive Tabu Search. *ORSA Journal on Computing*, 6(2):126–140, 1994.
8. S. Binato, W. J. Hery, D. Loewenstern, and M. G. C. Resende. A greedy randomized adaptive search procedure for job shop scheduling. In P. Hansen and C. C. Ribeiro, editors, *Essays and surveys on metaheuristics*, pages 59–79. Kluwer Academic Publishers, 2001.
9. C. Blum. Ant colony optimization. *Physics of Life Reviews*, 2(4):353–373, 2005.
10. C. Blum. Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591, 2005.
11. C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
12. S. Boettcher and A. G. Percus. Optimization with extremal dynamics. *Complexity*, 8:57–62, 2003.
13. P. Calégary, G. Coray, A. Hertz, D. Kobler, and P. Kuonen. A taxonomy of evolutionary algorithms in combinatorial optimization. *Journal of Heuristics*, 5:145–158, 1999.
14. V. Černý. A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.
15. P. Chardaire, J. L. Lutton, and A. Sutter. Thermostatistical persistency: A powerful improving concept for simulated annealing algorithms. *European Journal of Operational Research*, 86:565–579, 1995.

16. C. A. Coello Coello. An Updated Survey of GA-Based Multiobjective Optimization Techniques. *ACM Computing Surveys*, 32(2):109–143, 2000.
17. D. T. Connolly. An improved annealing scheme for the QAP. *European Journal of Operational Research*, 46:93–100, 1990.
18. T. G. Crainic and M. Toulouse. Introduction to the special issue on Parallel Meta-Heuristics. *Journal of Heuristics*, 8(3):247–249, 2002.
19. T. G. Crainic and M. Toulouse. Parallel Strategies for Meta-heuristics. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Norwell, MA, 2002.
20. F. Della Croce and V. T'kindt. A Recovering Beam Search algorithm for the one machine dynamic total completion time scheduling problem. *Journal of the Operational Research Society*, 53(11):1275–1280, 2002.
21. M. Dell'Amico, A. Lodi, and F. Maffioli. Solution of the Cumulative Assignment Problem with a well–structured Tabu Search method. *Journal of Heuristics*, 5:123–143, 1999.
22. M. L. den Besten, T. Stützle, and M. Dorigo. Design of iterated local search algorithms: An example application to the single machine total weighted tardiness problem. In E. J. W. Boers, J. Gottlieb, P. L. Lanzi, R. E. Smith, S. Cagnoni, E. Hart, G. R. Raidl, and H. Tijink, editors, *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2001*, volume 2037 of *Lecture Notes in Computer Science*, pages 441–452. Springer-Verlag, Berlin, Germany, 2001.
23. J.-L. Deneubourg, S. Aron, S. Goss, and J.-M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behaviour*, 3:159–168, 1990.
24. J. Denzinger and T. Offerman. On cooperation between evolutionary algorithms and other search paradigms. In *Proceedings of Congress on Evolutionary Computation – CEC'1999*, pages 2317–2324, 1999.
25. M. Dorigo and L. M. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.
26. M. Dorigo and T. Stützle. http://www.metaheuristics.net/, 2000. Visited in January 2003.
27. M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.
28. G. Dueck. New Optimization Heuristics. *Journal of Computational Physics*, 104:86–92, 1993.
29. G. Dueck and T. Scheuer. Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing. *Journal of Computational Physics*, 90:161–175, 1990.
30. W. Feller. *An Introduction to Probability Theory and its Applications*. John Whiley, 1968.
31. T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.
32. 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.
33. A. Fink and S. Voß. Generic metaheuristics application to industrial engineering problems. *Computers & Industrial Engineering*, 37:281–284, 1999.

34. M. Fleischer. Simulated Annealing: past, present and future. In C. Alexopoulos, K. Kang, W. R. Lilegdon, and G. Goldsman, editors, *Proceedings of the 1995 Winter Simulation Conference*, pages 155–161, 1995.
35. F. Focacci, F. Laburthe, and A. Lodi. Local Search and Constraint Programming. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Norwell, MA, 2002.
36. D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1994.
37. G. B. Fogel, V. W. Porto, D. G. Weekes, D. B. Fogel, R. H. Griffey, J. A. McNeil, E. Lesnik, D. J. Ecker, and R. Sampath. Discovery of RNA structural elements using evolutionary computation. *Nucleic Acids Research*, 30(23):5310–5317, 2002.
38. L. J. Fogel. Toward inductive inference automata. In *Proceedings of the International Federation for Information Processing Congress*, pages 395–399, Munich, 1962.
39. L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966.
40. C. Fonlupt, D. Robilliard, P. Preux, and E. G. Talbi. Fitness landscapes and performance of meta-heuristics. In S. Voß, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, 1999.
41. L. M. Gambardella and M. Dorigo. Ant Colony System hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3):237–255, 2000.
42. M. R. Garey and D. S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness*. W. H. Freeman, 1979.
43. M. Gendreau, G. Laporte, and J.-Y. Potvin. Metaheuristics for the capacitated VRP. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, volume 9 of *SIAM Monographs on Discrete Mathematics and Applications*, pages 129–154. SIAM, Philadelphia, 2002.
44. F. Glover. Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, 8:156–166, 1977.
45. F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13:533–549, 1986.
46. F. Glover. Tabu Search Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.
47. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.
48. D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, Reading, MA, 1989.
49. J. J. Grefenstette. A user's guide to GENESIS 5.0. Technical report, Navy Centre for Applied Research in Artificial Intelligence, Washington D.C., USA, 1990.
50. P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy, 1986.
51. P. Hansen and N. Mladenović. Variable Neighborhood Search for the p-Median. *Location Science*, 5:207–226, 1997.
52. P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voß, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-*

Heuristics: Advances and Trends in Local Search Paradigms for Optimization, chapter 30, pages 433–458. Kluwer Academic Publishers, 1999.
53. P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.
54. A. Hertz and D. Kobler. A framework for the description of evolutionary algorithms. *European Journal of Operational Research*, 126:1–12, 2000.
55. T. Hogg and C. P. Williams. Solving the really hard problems with cooperative search. In *Proceedings of AAAI93*, pages 213–235. AAAI Press, 1993.
56. J. H. Holland. *Adaption in natural and artificial systems.* The University of Michigan Press, Ann Harbor, MI, 1975.
57. H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications.* Elsevier, Amsterdam, The Netherlands, 2004.
58. T. Ibaraki and K. Nakamura. Packing problems with soft rectangles. In F. Almeida, M. Blesa, C. Blum, J. M. Moreno, M. Pérez, A. Roli, and M. Sampels, editors, *Proceedings of HM 2006 – 3rd International Workshop on Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, pages 13–27. Springer-Verlag, Berlin, Germany, 2006.
59. L. Ingber. Adaptive simulated annealing (ASA): Lessons learned. *Control and Cybernetics – Special Issue on Simulated Annealing Applied to Combinatorial Optimization*, 25(1):33–54, 1996.
60. D. S. Johnson and L. A. McGeoch. The traveling salesman problem: a case study. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, Chichester, UK, 1997.
61. T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD thesis, Univ. of New Mexico, Albuquerque, NM, 1995.
62. P. Kilby, P. Prosser, and P. Shaw. Guided Local Search for the Vehicle Routing Problem with time windows. In S. Voß, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 473–486. Kluwer Academic Publishers, 1999.
63. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
64. H. R. Lourenço, O. Martin, and T. Stützle. A beginner's introduction to Iterated Local Search. In *Proceedings of MIC'2001 – Meta–heuristics International Conference*, volume 1, pages 1–6, 2001.
65. H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.
66. V. Maniezzo. Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem. *INFORMS Journal on Computing*, 11(4):358–369, 1999.
67. O. Martin and S. W. Otto. Combining Simulated Annealing with Local Search Heuristics. *Annals of Operations Research*, 63:57–75, 1996.
68. O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326, 1991.
69. D. Merkle, M. Middendorf, and H. Schmeck. Ant Colony Optimization for Resource-Constrained Project Scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4):333–346, 2002.

70. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.
71. Z. Michalewicz and M. Michalewicz. Evolutionary computation techniques and their applications. In *Proceedings of the IEEE International Conference on Intelligent Processing Systems*, pages 14–24, Beijing, China, 1997. Institute of Electrical & Electronics Engineers, Incorporated.
72. M. Milano and A. Roli. MAGMA: A multiagent architecture for metaheuristics. *IEEE Trans. on Systems, Man and Cybernetics – Part B*, 34(2):925–941, 2004.
73. P. Mills and E. Tsang. Guided Local Search for solving SAT and weighted MAX-SAT Problems. In Ian Gent, Hans van Maaren, and Toby Walsh, editors, *SAT2000*, pages 89–106. IOS Press, 2000.
74. M. Mitchell. *An introduction to genetic algorithms*. MIT press, Cambridge, MA, 1998.
75. P. Moscato. Memetic algorithms: A short introduction. In F. Glover D. Corne and M. Dorigo, editors, *New Ideas in Optimization*. McGraw-Hill, 1999.
76. G. L. Nemhauser and A. L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, 1988.
77. E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job-shop problem. *Management Science*, 42(2):797–813, 1996.
78. I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:513–623, 1996.
79. P. S. Ow and T. E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26:297–307, 1988.
80. C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization – Algorithms and Complexity*. Dover Publications, Inc., New York, 1982.
81. L. S. Pitsoulis and M. G. C. Resende. Greedy Randomized Adaptive Search procedure. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of Applied Optimization*, pages 168–183. Oxford University Press, 2002.
82. M. Prais and C. C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.
83. S. Prestwich. Combining the Scalability of Local Search with the Pruning Techniques of Systematic Search. *Annals of Operations Research*, 115:51–72, 2002.
84. S. Prestwich and A. Roli. Symmetry breaking and local search spaces. In *Proceedings of CPAIOR 2005*, volume 3524 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2005.
85. J. Puchinger and G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In J. Mira and J. R. Álvarez, editors, *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation*, volume 3562 of *Lecture Notes in Computer Science*, pages 41–53. Springer-Verlag, Berlin, Germany, 2005.
86. G. R. Raidl. A unified view on hybrid metaheuristics. In F. Almeida, M. Blesa, C. Blum, J. M. Moreno, M. Pérez, A. Roli, and M. Sampels, editors, *Proceedings of HM 2006 – 3rd International Workshop on Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, Berlin, Germany, 2006.

87. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.
88. C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publishing, Oxford, England, 1993.
89. C. R. Reeves and J. E. Rowe. *Genetic Algorithms: Principles and Perspectives. A Guide to GA Theory*. Kluwer Academic Publishers, Boston (USA), 2002.
90. M. G. C. Resende and C. C. Ribeiro. A GRASP for graph planarization. *Networks*, 29:173–189, 1997.
91. C. C. Ribeiro and M. C. Souza. Variable neighborhood search for the degree constrained minimum spanning tree problem. *Discrete Applied Mathematics*, 118:43–54, 2002.
92. A. Roli. Symmetry-breaking and local search: A case study. In *SymCon'04 – 4th International Workshop on Symmetry and Constraint Satisfaction Problems*. 2004.
93. A. Schaerf, M. Cadoli, and M. Lenzerini. LOCAL++: A C++ framework for local search algorithms. *Software Practice & Experience*, 30(3):233–257, 2000.
94. G. R. Schreiber and O. C. Martin. Cut size statistics of graph bisection heuristics. *SIAM Journal on Optimization*, 10(1):231–251, 1999.
95. P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In M. Maher and J.-F. Puget, editors, *Principle and Practice of Constraint Programming – CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer-Verlag, 1998.
96. A. Shmygelska and H. H. Hoos. An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem. *BMC Bioinformatics*, 6(30):1–22, 2005.
97. M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Pérez-Uribe, and A. Stauffer. A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems. *IEEE Transactions on Evolutionary Computation*, 1(1):83–97, 1997.
98. L. Sondergeld and S. Voß. Cooperative intelligent search using adaptive memory techniques. In S. Voß, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, chapter 21, pages 297–312. Kluwer Academic Publishers, 1999.
99. W. M. Spears, K. A. De Jong, T. Bäck, D. B. Fogel, and H. de Garis. An overview of evolutionary computation. In P. B. Brazdil, editor, *Proceedings of the European Conference on Machine Learning (ECML-93)*, volume 667, pages 442–459, Vienna, Austria, 1993. Springer-Verlag.
100. P. F. Stadler. Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20:1–45, 1996. Also available as SFI preprint 95-07-067.
101. T. Stützle. *Local Search Algorithms for Combinatorial Problems – Analysis, Algorithms and New Applications*. DISKI – Dissertationen zur Künstliken Intelligenz. infix, Sankt Augustin, Germany, 1999.
102. T. Stützle and H. H. Hoos. $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.
103. É. D. Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991.
104. E.-G. Talbi. A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics*, 8(5):541–564, 2002.

105. M. Toulouse, T. G. Crainic, and B. Sansò. An experimental study of the systemic behavior of cooperative search algorithms. In S. Voß, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, chapter 26, pages 373–392. Kluwer Academic Publishers, 1999.
106. D. Urošević, J. Brimberg, and N. Mladenović. Variable neighborhood decomposition search for the edge weighted k-cardinality tree problem. *Computers & Operations Research*, 31:1205–1213, 2004.
107. P. J. M. Van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job Shop Scheduling by Simulated Annealing. *Operations Research*, 40:113–125, 1992.
108. M. D. Vose. *The simple genetic algorithm: foundations and theory*. MIT Press, Cambridge, MA, 1999.
109. S. Voß, S. Martello, I. H. Osman, and C. Roucairol, editors. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.
110. S. Voß and D. Woodruff, editors. *Optimization Software Class Libraries*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.
111. C. Voudouris. *Guided Local Search for Combinatorial Optimization Problems*. PhD thesis, Department of Computer Science, University of Essex, 1997. pp. 166.
112. C. Voudouris and E. Tsang. Guided Local Search. *European Journal of Operational Research*, 113(2):469–499, 1999.
113. A. S. Wade and V. J. Rayward-Smith. Effective local search for the Steiner tree problem. *Studies in Locational Analysis*, 11:219–241, 1997. Also in *Advances in Steiner Trees*, ed. by Ding-Zhu Du, J. M.Smith and J. H. Rubinstein, Kluwer, 2000.
114. D. Whitley. The GENITOR algorithm and selective pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the 3rd International Conference on Genetic Algorithms, ICGA 1989*, pages 116–121. Morgan Kaufmann Publishers, 1989.