Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison

Christian Blum
Université Libre de Bruxelles

Andrea Roli
Università degli Studi di Bologna

(Preprint)

Abstract

The field of metaheuristics for the application to combinatorial optimization problems is a rapidly growing field of research. This is due to the importance of combinatorial optimization problems for the scientific as well as the industrial world. We give a survey of the nowadays most important metaheuristics from a conceptual point of view. We outline the different components and concepts that are used in the different metaheuristics in order to analyze their similarities and differences. Two very important concepts in metaheuristics are intensification and diversification. These are the two forces that largely determine the behaviour of a metaheuristic. They are in some way contrary but also complementary to each other. We introduce a framework, that we call the *I&D* frame, in order to put different intensification and diversification components into relation with each other. Outlining the advantages and disadvantages of different metaheuristic approaches we conclude by pointing out the importance of hybridization of metaheuristics as well as the integration of metaheuristics and other methods for optimization.

1 Introduction

Many optimization problems of both practical and theoretical importance concern themselves with the choice of a "best" configuration of a set of variables to achieve some goals. They seem to divide naturally into two categories: those where solutions are encoded with *real-valued* variables, and those where solutions are encoded with *discrete* variables. Among the latter ones we find a class of problems called Combinatorial Optimization (CO) problems. According to [126], in CO problems, we are looking for an object from a finite – or possibly countably infinite – set. This object is typically an integer number, a subset, a permutation, or a graph structure.

Definition 1.1. *A* **Combinatorial Optimization** *problem $P = (\mathcal{S}, f)$ can be defined by:*

- *a set of variables $X = \{x_1, \ldots, x_n\}$;*
- *variable domains $D_1, \ldots, D_n$;*
- *constraints among variables;*
- *an objective function f to be minimized,*[1] *where $f : D_1 \times \ldots \times D_n \to \mathbb{R}^+$;*

The set of all possible feasible assignments is

$$\mathcal{S} = \{s = \{(x_1, v_1), \ldots, (x_n, v_n)\} \mid v_i \in D_i, s \text{ satisfies all the constraints}\} \ .$$

$\mathcal{S}$ is usually called a search (or solution) space, as each element of the set can be seen as a candidate solution. To solve a combinatorial optimization problem one has to find a solution $s^* \in \mathcal{S}$ with minimum objective function value, that is, $f(s^*) \leq f(s)\ \forall s \in S$. s^* is called a globally optimal solution of $(\mathcal{S}, f)$ and the set $\mathcal{S}^* \subseteq \mathcal{S}$ is called the set of globally optimal solutions.

[1] As maximizing an objective function f is the same as minimizing $-f$, in this work we will deal, without loss of generality, with minimization problems.

Examples for CO problems are the Travelling Salesman problem (TSP), the Quadratic Assignment problem (QAP), Timetabling and Scheduling problems. Due to the practical importance of CO problems, many algorithms to tackle them have been developed. These 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 [126, 122]). Yet, for CO problems that are $\mathcal{NP}$-hard [63], 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. Thus, the use of approximate methods to solve CO problems has received more and more attention in the last 30 years. In approximate methods we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time.

Among the basic approximate methods we usually distinguish between *constructive* methods and *local search* methods. Constructive algorithms generate solutions from scratch by adding – to an initially empty partial solution – components, until a solution is complete. They are typically the fastest approximate methods, 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 1.2. *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.*

The introduction of a neighborhood structure enables us to define the concept of *locally* minimal solutions.

Definition 1.3. *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 $f(\hat{s}) < f(s)$ $\forall\, s \in \mathcal{N}(\hat{s})$.*

In the last 20 years, a new kind of approximate algorithm has emerged which basically 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*.[2] The term *metaheuristic*, first introduced in [67], derives from the composition of two Greek words. *Heuristic* derives from the verb *heuriskein* (ευρισκειν) 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* [137].

This class of algorithms includes[3] – but is not restricted to – Ant Colony Optimization (ACO), Evolutionary Computation (EC) including Genetic Algorithms (GA), Iterated Local Search (ILS), Simulated Annealing (SA), and Tabu Search (TS). Up to now there is no commonly accepted definition for the term metaheuristic. It is just in the last few years that some researchers in the field tried to propose a definition. In the following we quote some of them:

"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." [125].

"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." [164].

"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 optima. This

[2] The increasing importance of metaheuristics is underlined by the biannual Metaheuristics International Conference (MIC). The 5th is being held in Kyoto in August 2003 (*http://www-or.amp.i.kyoto-u.ac.jp/mic2003/*).

[3] in alphabetical order

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." [153].

"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." [110].

Summarizing, we outline fundamental properties which characterize metaheuristics:

- Metaheuristics are strategies that "guide" the search process.
- The goal is to efficiently explore the search space in order to find (near–)optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics permit an abstract level description.
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Todays more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

In short we could say that metaheuristics are high level strategies for exploring search spaces by using different methods. Of great importance hereby is that a dynamic balance is given between *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. These terms stem from the Tabu Search field [70] and it is important to clarify that the terms *exploration* and *exploitation* are sometimes used instead, for example in the Evolutionary Computation field [48], with a more restricted meaning. In fact, the notions of exploitation and exploration often refer to rather short term strategies tied to randomness, whereas intensification and diversification also refer to medium and long term strategies based on the usage of memory. The use of the terms diversification and intensification in their initial meaning becomes more and more accepted by the whole field of metaheuristics. Therefore, we use them throughout the paper. The balance between diversification and intensification as mentioned above 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.

The search strategies of different metaheuristics are highly dependent on the philosophy of the metaheuristic itself. Comparing the strategies used in different metaheuristics is one of the goals of Section 5. There are several different philosophies apparent in the existing metaheuristics. Some of them can be seen as "intelligent" extensions of local search algorithms. The goal of this kind of metaheuristic is to escape from local minima in order to proceed in the exploration of the search space and to move on to find other hopefully better local minima. This is for example the case in Tabu Search, Iterated Local Search, Variable Neighborhood Search, GRASP and Simulated Annealing. These metaheuristics (also called trajectory

methods) work on one or several neighborhood structure(s) imposed on the members (the solutions) of the search space.

We can find a different philosophy in algorithms like Ant Colony Optimization and Evolutionary Computation. They incorporate a learning component in the sense that they implicitly or explicitly try to learn correlations between decision variables to identify high quality areas in the search space. This kind of metaheuristic performs, in a sense, a biased sampling of the search space. For instance, in Evolutionary Computation this is achieved by recombination of solutions and in Ant Colony Optimization by sampling the search space in every iteration according to a probability distribution.

The structure of this work is as follows. There are several approaches to classify metaheuristics according to their properties. In Section 2 we briefly list and summarize different classification approaches. Section 3 and Section 4 are devoted to a description of the most important metaheuristics nowadays. Section 3 describes the most relevant trajectory methods and in Section 4 we outline population-based methods. Section 5 aims at giving a unifying view on metaheuristics with respect to the way they achieve intensification and diversification. This is done by the introduction of a unifying framework, the *I&D frame*. Finally, Section 6 offers some conclusions and an outlook to the future.

We believe that it is hardly possible to produce a completely accurate survey of metaheuristics that is doing justice to every viewpoint. Moreover, a survey of an immense area such as metaheuristics has to focus on certain aspects and therefore has unfortunately to neglect other aspects. Therefore, we want to clarify at this point that this survey is done from the conceptual point of view. We want to outline the different concepts that are used in different metaheuristics in order to analyze the similarities and the differences between them. We do not go into the implementation of metaheuristics, which is certainly an important aspect of metaheuristics research with respect to the increasing importance of efficiency and software reusability. We refer the interested reader to [170, 78, 52, 145, 165].

2 Classification of metaheuristics

There are different ways to classify and describe metaheuristic algorithms. Depending on the characteristics selected to differentiate among them, several classifications are possible, each of them being the result of a specific viewpoint. We briefly summarize the most important ways of classifying metaheuristics.

Nature-inspired vs. non-nature inspired. Perhaps, the most intuitive way of classifying metaheuristics is based on the origins of the algorithm. There are nature-inspired algorithms, like Genetic Algorithms and Ant Algorithms, and non nature-inspired ones such as Tabu Search and Iterated Local Search. In our opinion this classification is not very meaningful for the following two reasons. First, many recent hybrid algorithms do not fit either class (or, in a sense, they fit both at the same time). Second, it is sometimes difficult to clearly attribute an algorithm to one of the two classes. So, for example, one might ask the question if the use of memory in Tabu Search is not nature-inspired as well.

Population-based vs. single point search. Another characteristic that can be used for the classification of metaheuristics is the number of solutions used at the same time: Does the algorithm work on a population or on a single solution at any time? Algorithms working on single solutions are called *trajectory methods* and encompass local search-based metaheuristics, like Tabu Search, Iterated Local Search and Variable Neighborhood Search. They all share the property of describing a trajectory in the search space during the search process. Population-based metaheuristics, on the contrary, perform search processes which describe the evolution of a set of points in the search space.

Dynamic vs. static objective function. Metaheuristics can also be classified according to the way they make use of the objective function. While some algorithms keep the objective function given in the problem representation "as it is", some others, like Guided Local Search (GLS), modify it during the search. The idea behind this approach is to escape from local minima by modifying the search landscape. Accordingly, during the search the objective function is altered by trying to incorporate information collected during the search process.

One vs. various neighborhood structures. Most metaheuristic algorithms work on one single neighborhood structure. In other words, the fitness landscape topology does not change in the course of the algorithm. Other metaheuristics, such as Variable Neighborhood Search (VNS), use a set of neighbor-

hood structures which gives the possibility to diversify the search by swapping between different fitness landscapes.

Memory usage vs. memory-less methods. A very important feature to classify metaheuristics is the use they make of the search history, that is, whether they use memory or not.[4] Memory-less algorithms perform a Markov process, as the information they exclusively use to determine the next action is the current state of the search process. There are several different ways of making use of memory. Usually we differentiate between the use of short term and long term memory. The first usually keeps track of recently performed moves, visited solutions or, in general, decisions taken. The second is usually an accumulation of synthetic parameters about the search. The use of memory is nowadays recognized as one of the fundamental elements of a powerful metaheuristic.

In the following we describe the most important metaheuristics according to the single point vs. population-based search classification, which divides metaheuristics into trajectory methods and population-based methods. This choice is motivated by the fact that this categorization permits a clearer description of the algorithms. Moreover, a current trend is the hybridization of methods in the direction of the integration of single point search algorithms in population-based ones. In the following two sections we give a detailed description of nowadays most important metaheuristics.

3 Trajectory Methods

In this section we outline metaheuristics called *trajectory methods*. The term trajectory methods is used because the search process performed by these methods is characterized by a trajectory in the search space. Hereby, a successor solution may or may not belong to the neighborhood of the current solution.

The search process of trajectory methods can be seen as the evolution in (discrete) time of a discrete dynamical system [9, 35]. The algorithm starts from an initial state (the initial solution) and describes a trajectory in the state space. The system dynamics depends on the strategy used; simple algorithms generate a trajectory composed of two parts: a *transient* phase followed by an *attractor* (a fixed point, a cycle or a complex attractor). Algorithms with advanced strategies generate more complex trajectories which can not be subdivided in those two phases. The characteristics of the trajectory provide information about the behavior of the algorithm and its effectiveness with respect to the instance that is tackled. It is worth underlining that the dynamics is the result of the combination of algorithm, problem representation and problem instance. In fact, the problem representation together with the neighborhood structures define the search landscape; the algorithm describes the strategy used to explore the landscape and, finally, the actual search space characteristics are defined by the problem instance to be solved.

We will first describe basic local search algorithms, before we proceed with the survey of more complex strategies. Finally, we deal with algorithms that are general explorative strategies which may incorporate other trajectory methods as components.

3.1 Basic Local Search: Iterative Improvement

The basic local search is usually called *iterative improvement*, since each move[5] is only performed if the resulting solution is better than the current solution. The algorithm stops as soon as it finds a local minimum. The high level algorithm is sketched in Figure 1.

The function $\mathsf{Improve}(\mathcal{N}(s))$ can be in the extremes either a *first improvement*, or a *best improvement* function, or any intermediate option. The former scans the neighborhood $\mathcal{N}(s)$ and chooses the first solution that is better than s, the latter exhaustively explores the neighborhood and returns one of the solutions with the lowest objective function value. Both methods stop at local minima. Therefore, their performance strongly depends on the definition of S, f and $\mathcal{N}$. The performance of iterative improvement procedures on CO problems is usually quite unsatisfactory. Therefore, several techniques have been developed to prevent algorithms from getting trapped in local minima, which is done by adding mechanisms that allow them to escape from local minima. This also implies that the termination conditions of metaheuristic algorithms

[4] Here we refer to the use of adaptive memory, in contrast to rather rigid memory, as used for instance in Branch & Bound.

[5] A move is the choice of a solution s' from the neighborhood $\mathcal{N}(s)$ of a solution s.

```
s ← GenerateInitialSolution()
repeat
    s ← Improve(N(s))
until no improvement is possible
```

Figure 1: Algorithm: Iterative Improvement

```
s ← GenerateInitialSolution()
T ← T_0
while termination conditions not met do
   s' ← PickAtRandom(N(s))
   if (f(s') < f(s)) then
      s ← s'          % s' replaces s
   else
      Accept s' as new solution with probability p(T, s', s)
   endif
   Update(T)
endwhile
```

Figure 2: Algorithm: Simulated Annealing (SA)

are more complex than simply reaching a local minimum. Indeed, possible termination conditions include: maximum CPU time, a maximum number of iterations, a solution s with $f(s)$ less than a predefined threshold value is found, or the maximum number of iterations without improvements is reached.

3.2 Simulated Annealing

Simulated Annealing (SA) is commonly said to be the oldest among the metaheuristics and surely one of the first algorithms that had an explicit strategy to avoid local minima. The origins of the algorithm are in statistical mechanics (Metropolis algorithm) and it was first presented as a search algorithm for CO problems in [99] and [20]. The fundamental idea is to allow moves resulting in solutions of worse quality than the current solution (uphill moves) in order to escape from local minima. The probability of doing such a move is decreased during the search. The high level algorithm is described in Figure 2.

The algorithm starts by generating an initial solution (either randomly or heuristically constructed) and by initializing the so-called temperature parameter T. Then, at each iteration a solution $s' \in \mathcal{N}(s)$ is randomly sampled and it is accepted as new current solution depending on $f(s)$, $f(s')$ and T. s' replaces s if $f(s') < f(s)$ or, in case $f(s') \geq f(s)$, with a probability which is a function of T and $f(s') - f(s)$. The probability is generally computed following the Boltzmann distribution $\exp(-\frac{f(s')-f(s)}{T})$.

The temperature T is decreased[6] during the search process, thus at the beginning of the search the probability of accepting uphill moves is high and it gradually decreases, converging to a simple iterative improvement algorithm. This process is analogous to the annealing process of metals and glass, which assume a low energy configuration when cooled with an appropriate cooling schedule. Regarding the search process, this means that the algorithm is the result of two combined strategies: random walk and iterative improvement. In the first phase of the search, the bias toward improvements is low and it permits the exploration of the search space; this erratic component is slowly decreased thus leading the search to converge to a (local) minimum. The probability of accepting uphill moves is controlled by two factors: the difference of the objective functions and the temperature. On the one hand, at fixed temperature, the higher the difference $f(s') - f(s)$, the lower the probability to accept a move from s to s'. On the other hand, the higher T, the higher the probability of uphill moves.

[6] T is not necessarily decreased in a monotonic fashion. Elaborate cooling schemes also incorporate an occasional increase of the temperature.

The choice of an appropriate cooling schedule is crucial for the performance of the algorithm. The cooling schedule defines the value of T at each iteration k, $T_{k+1} = Q(T_k, k)$, where $Q(T_k, k)$ is a function of the temperature and of the iteration number. Theoretical results on non-homogeneous Markov chains [1] state that under particular conditions on the cooling schedule, the algorithm converges in probability to a global minimum for $k \to \infty$. More precisely:

$$\exists \Gamma \in I\!R \quad \text{s.t.} \quad \lim_{k\to\infty} \mathbf{p}(\text{global minimum found after } k \text{ steps}) = 1$$
$$\text{iff} \quad \sum_{k=1}^{\infty} exp(\frac{\Gamma}{T_k}) = \infty$$

A particular cooling schedule that fulfils the hypothesis for the convergence is the one that follows a logarithmic law: $T_{k+1} = \frac{\Gamma}{\log(k+k_0)}$ (where k_0 is a constant). Unfortunately, cooling schedules which guarantee the convergence to a global optimum are not feasible in applications, because they are too slow for practical purposes. Therefore, faster cooling schedules are adopted in applications. One of the most used follows a geometric law: $T_{k+1} = \alpha T_k$, where $\alpha \in (0,1)$, which corresponds to an exponential decay of the temperature.

The cooling rule may vary during the search, with the aim of tuning the balance between diversification and intensification. For example, at the beginning of the search, T might be constant or linearly decreasing, in order to sample the search space; then, T might follow a rule such as the geometric one, to converge to a local minimum at the end of the search. More successful variants are *non-monotonic* cooling schedules (e.g., see [124, 106]). Non-monotonic cooling schedules are characterized by alternating phases of cooling and reheating, thus providing an oscillating balance between diversification and intensification.

The cooling schedule and the initial temperature should be adapted to the particular problem instance, since the cost of escaping from local minima depends on the structure of the search landscape. A simple way of empirically determining the starting temperature T_0 is to initially sample the search space with a random walk to roughly evaluate the average and the variance of objective function values. But also more elaborate schemes can be implemented [91].

The dynamic process described by SA is a *Markov chain* [49], as it follows a trajectory in the state space in which the successor state is chosen depending only on the incumbent one. This means that basic SA is memory-less. However, the use of memory can be beneficial for SA approaches (see for example [21]).

SA has been applied to several CO problems, such as the Quadratic Assignment Problem (QAP) [23] and the Job Shop Scheduling (JSS) problem [162]. References to other applications can be found in [2, 91, 53]. SA is nowadays used as a component in metaheuristics, rather than applied as stand-alone search algorithm. Variants of SA called Threshold Accepting and The Great Deluge Algorithm were presented by [45] and [44].

3.3 Tabu Search

Tabu Search (TS) is among the most cited and used metaheuristics for CO problems. TS basic ideas were first introduced in [67], based on earlier ideas formulated in [66].[7] A description of the method and its concepts can be found in [70]. TS explicitly uses the history of the search, both to escape from local minima and to implement an explorative strategy. We will first describe a simple version of TS, to introduce the basic concepts. Then, we will explain a more applicable algorithm and finally we will discuss some improvements.

The simple TS algorithm (see Figure 3) applies a best improvement local search as basic ingredient and uses a *short term memory* to escape from local minima and to avoid cycles. The short term memory is implemented as a *tabu list* that keeps track of the most recently visited solutions and forbids moves toward them. The neighborhood of the current solution is thus restricted to the solutions that do not belong to the tabu list. In the following we will refer to this set as *allowed set*. At each iteration the best solution from the allowed set is chosen as the new current solution. Additionally, this solution is added to the tabu list and one of the solutions that were already in the tabu list is removed (usually in a FIFO order). Due to this dynamic restriction of allowed solutions in a neighborhood, TS can be considered as a dynamic

[7] Related ideas were labelled steepest ascent/mildest descent method in [79].

$s \leftarrow$ GenerateInitialSolution()
$TabuList \leftarrow \emptyset$
while termination conditions not met **do**
  $s \leftarrow$ ChooseBestOf($\mathcal{N}(s) \setminus TabuList$)
  Update($TabuList$)
endwhile

Figure 3: Algorithm: Simple Tabu Search (TS)

neighborhood search technique [153]. The algorithm stops when a termination condition is met. It might also terminate if the allowed set is empty, that is, if all the solutions in $\mathcal{N}(s)$ are forbidden by the tabu list.[8]

The use of a tabu list prevents from returning to recently visited solutions, therefore it prevents from endless cycling[9] and forces the search to accept even uphill moves. The length l of the tabu list (i.e., 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 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 [157], 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 [12, 11], where the tabu tenure is increased if there is evidence for repetitions of solutions (thus a higher diversification is needed), while it is decreased if there are no improvements (thus intensification should be boosted). More advanced ways to create dynamic tabu tenure are described in [68].

However, the implementation of short term memory as a list that contains complete solutions is not practical, because managing a list of solutions is highly inefficient. Therefore, instead of the solutions themselves, solution *attributes* are stored.[10] Attributes are usually components of solutions, moves, or differences between two solutions. Since more than one attribute can be considered, a tabu list is introduced for each of them. The set of attributes and the corresponding tabu lists define the *tabu conditions* which are used to filter the neighborhood of a solution and generate the allowed set. Storing attributes instead of complete solutions is much more efficient, but it introduces a loss of information, as forbidding an attribute means assigning the tabu status to probably more than one solution. Thus, it is possible that unvisited solutions of good quality are excluded from the allowed set. To overcome this problem, *aspiration criteria* are defined which allow to include a solution in the allowed set even if it is forbidden by tabu conditions. Aspiration criteria define the aspiration conditions that are used to construct the allowed set. The most commonly used aspiration criterion selects solutions which are better than the current best one. The complete algorithm, as described above, is reported in Figure 4.

Tabu lists are only one of the possible ways of taking advantage of the history of the search. They are usually identified with the usage of short term memory. Information collected during the whole search process can also be very useful, especially for a strategic guidance of the algorithm. This kind of long term memory is usually added to TS by referring to four principles: *recency*, *frequency*, *quality* and *influence*. Recency-based memory records for each solution (or attribute) the most recent iteration it was involved in. Orthogonally, frequency-based memory keeps track of how many times each solution (attribute) has been visited. This information identifies the regions (or the subsets) of the solution space where the search was confined, or where it stayed for a high number of iterations. This kind of information about the past is usually exploited to diversify the search. The third principle (i.e., quality) refers to the accumulation and extraction of information from the search history in order to identify good solution components. This information can be usefully integrated in the solution construction. Other metaheuristics (e.g., Ant Colony Optimization) explicitly use this principle to learn about good combinations of solution components. Finally, influence is a property regarding choices made during the search and can be used to indicate which

[8] Strategies for avoiding to stop the search when the allowed set is empty include the choice of the least recently visited solution, even if it is tabu.

[9] Cycles of higher period are possible, since the tabu list has a finite length l which is smaller than the cardinality of the search space.

[10] In addition to storing attributes, some longer term TS strategies also keep complete solutions (e.g., elite solutions) in the memory.

$s \leftarrow$ GenerateInitialSolution()
InitializeTabuLists($TL_1, \ldots, TL_r$)
$k \leftarrow 0$
while termination conditions not met **do**
  $AllowedSet(s,k) \leftarrow \{s' \in \mathcal{N}(s) \mid s$ does not violate a tabu condition,
    or it satisfies at least one aspiration condition$\}$
  $s \leftarrow$ ChooseBestOf($AllowedSet(s,k)$)
  UpdateTabuListsAndAspirationConditions()
  $k \leftarrow k+1$
endwhile

Figure 4: Algorithm: Tabu Search (TS)

while termination conditions not met **do**
  $s \leftarrow$ ConstructGreedyRandomizedSolution() % see Figure 6
  ApplyLocalSearch(s)
  MemorizeBestFoundSolution()
endwhile

Figure 5: Algorithm: Greedy Randomized Adaptive Search Procedure (GRASP)

choices have shown to be the most critical. In general, the TS field is a rich source of ideas. Many of these ideas and strategies have been and are currently adopted by other metaheuristics.

TS has been applied to most CO problems; examples for successful applications are the Robust Tabu Search to the QAP [157], the Reactive Tabu Search to the MAXSAT problem [11], and to assignment problems [31]. TS approaches dominate the Job Shop Scheduling (JSS) problem area (see for example [123]) and the Vehicle Routing (VR) area [64]. Further current applications can be found at [156].

3.4 Explorative Local Search methods

In this section we present more recently proposed trajectory methods. These are the Greedy Randomized Adaptive Search Procedure (GRASP), Variable Neighborhood Search (VNS), Guided Local Search (GLS) and Iterated Local Search (ILS).

3.4.1 GRASP

The Greedy Randomized Adaptive Search Procedure (GRASP), see [50, 131], is a simple metaheuristic that combines constructive heuristics and local search. Its structure is sketched in Figure 5. GRASP is an iterative procedure, composed of two phases: solution construction and solution improvement. The best found solution is returned upon termination of the search process.

The solution construction mechanism (see Figure 6) is characterized by two main ingredients: a dynamic constructive heuristic and randomization. Assuming that a solution s consists of a subset of a set of elements (solution components), the solution is constructed step-by-step by adding one new element at a time. The choice of the next element is done by picking it uniformly at random from a candidate list. The elements are ranked by means of a heuristic criterion that gives them a score as a function of the (myopic) benefit if inserted in the current partial solution. The candidate list, called *restricted candidate list* (RCL), is composed of the best α elements. The heuristic values are updated at each step, thus the scores of elements change during the construction phase, depending on the possible choices. This constructive heuristic is called *dynamic*, in contrast to the *static* one which assigns a score to elements only before starting the construction. For instance, one of the static heuristics for the TSP is based on arc costs: the lower the cost

```
s ← ∅          % s denotes a partial solution in this case
α ← DetermineCandidateListLength()          % definition of the RCL length
while solution not complete do
   RCL_α ← GenerateRestrictedCandidateList(s)
   x ← SelectElementAtRandom(RCL_α)
   s ← s ∪ {x}
   UpdateGreedyFunction(s)          % update of the heuristic values (see text)
endwhile
```

Figure 6: Greedy randomized solution construction

of an arc, the higher its score. An example of a dynamic heuristic is the *cheapest insertion* heuristic, where the score of an element is evaluated depending on the current partial solution.

The length α of the restricted candidate list determines the strength of the heuristic bias. In the extreme case of $\alpha = 1$ the best element would be added, thus the construction would be equivalent to a deterministic Greedy Heuristic. On the opposite, in case $\alpha = n$ the construction would be completely random (indeed, the choice of an element from the candidate list is done at random). Therefore, α is a critical parameter which influences the sampling of the search space. In [131] the most important schemes to define α are listed. The simplest scheme is, trivially, to keep α constant; it can also be changed at each iteration, either randomly or by means of an adaptive scheme.

The second phase of the algorithm is a local search process, which may be a basic local search algorithm such as iterative improvement, or a more advanced technique such as SA or TS. GRASP can be effective if two conditions are satisfied:

- the solution construction mechanism samples the most promising regions of the search space;
- the solutions constructed by the constructive heuristic belong to basins of attraction of different locally minimal solutions;

The first condition can be met by the choice of an effective constructive heuristic and an appropriate length of the candidate list, whereas the second condition can be met by choosing the constructive heuristic and the local search in a way such that they fit well.

The description of GRASP as given above indicates that a basic GRASP does not use the history of the search process.[11] The only memory requirement is for storing the problem instance and for keeping the best so-far solution. 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. Furthermore, it can be successfully integrated into other search techniques. Among the applications of GRASP we mention the JSS problem [13], the graph planarization problem [142] and assignment problems [132]. A detailed and annotated bibliography references many more applications [51].

3.4.2 Variable Neighborhood Search

Variable Neighborhood Search (VNS) is a metaheuristic proposed in [81, 82], which explicitly applies a strategy based on dynamically changing neighborhood structures. The algorithm is very general and many degrees of freedom exist for designing variants and particular instantiations.[12]

At the initialization step, a set of neighborhood structures has to be defined. These neighborhoods can be arbitrarily chosen, but often a sequence $|\mathcal{N}_1| < |\mathcal{N}_2| < \ldots < |\mathcal{N}_{k_{max}}|$ of neighborhoods with increasing cardinality is defined.[13] Then an initial solution is generated, the neighborhood index is initialized and the

[11] However, some extensions in this direction are cited in [131], and an example for a metaheuristic method using an adaptive greedy procedure depending on search history is Squeaky Wheel Optimization (SWO) [95].

[12] The variants described in the following are also described in [81, 82].

[13] In principle they could be one included in the other, $\mathcal{N}_1 \subset \mathcal{N}_2 \subset \ldots \subset \mathcal{N}_{k_{max}}$. Nevertheless, such a sequence might produce an inefficient search, because a large number of solutions could be revisited.

```
Select a set of neighborhood structures N_k, k = 1, ..., k_max
s ← GenerateInitialSolution()
while termination conditions not met do
  k ← 1
  while k < k_max do          % Inner loop
    s' ← PickAtRandom(N_k(s))          % Shaking phase
    s'' ← LocalSearch(s')
    if (f(s'') < f(s)) then
      s ← s''
      k ← 1
    else
      k ← k + 1
    endif
  endwhile
endwhile
```

Figure 7: Algorithm: Variable Neighborhood Search (VNS)

algorithm iterates until a stopping condition is met (see Figure 7). VNS' main cycle is composed of three phases: *shaking*, *local search* and *move*. In the *shaking* phase a solution s' in the k-th neighborhood of the current solution s is randomly selected. Then, s' becomes the local search starting point. The local search can use any neighborhood structure and is not restricted to the set of neighborhood structures $\mathcal{N}_k$, $k = 1, \cdots, k_{max}$. At the end of the local search process (terminated as soon as a predefined termination condition is verified) the new solution s'' is compared with s and, if it is better, it replaces s and the algorithm starts again with $k = 1$. Otherwise, k is incremented and a new shaking phase starts using a different neighborhood.

The objective of the shaking phase is to perturb the solution so as to provide a good starting point for the local search. The starting point should belong to the basin of attraction of a different local minimum than the current one, but should not be "too far" from s, otherwise the algorithm would degenerate into a simple random multi-start. Moreover, choosing s' in the neighborhood of the current best solution is likely to produce a solution that maintains some good features of the current one.

The process of changing neighborhoods in case of no improvements corresponds to a diversification of the search. In particular the choice of neighborhoods of increasing cardinality yields a progressive diversification. The effectiveness of this dynamic neighborhood strategy can be explained by the fact that a "bad" place on the search landscape given by one neighborhood could be a "good" place on the search landscape given by another neighborhood.[14] Moreover, a solution that is locally optimal with respect to a neighborhood is probably not locally optimal with respect to another neighborhood. These concepts are known as "*One Operator, One Landscape*" and explained in [93, 94]. The core idea is that the neighborhood structure determines the topological properties of the search landscape, i.e., each neighborhood defines one landscape. The properties of a landscape are in general different from those of other landscapes, therefore a search strategy performs differently on them (see an example in Figure 8).

This property is directly exploited by a local search called Variable Neighborhood Descent (VND). In VND a *best improvement* local search (see Section 3.1) is applied, and, in case a local minimum is found, the search proceeds with another neighborhood structure. The VND algorithm can be obtained by substituting the inner loop of the VNS algorithm (see Figure 7) with the following pseudo-code:

```
s' ← ChooseBestOf(N_k(s))
if (f(s') < f(s)) then          % i.e., if a better solution is found in N_k(s)
  s ← s'
else          % i.e., s is a local minimum
```

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

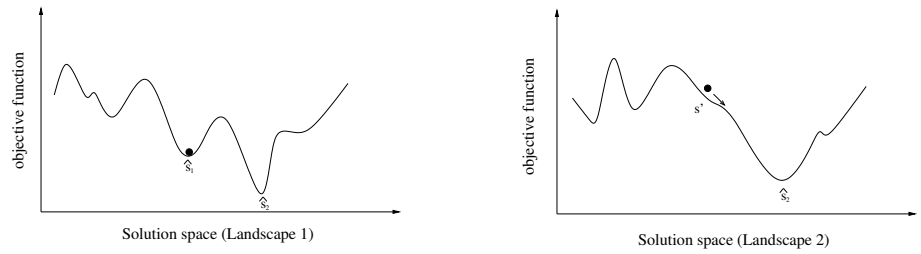

Figure 8: Two search landscapes defined by two different neighborhoods. On the landscape that is shown in the graphic on the left, the best improvement local search stops at $\hat{s_1}$, while it proceeds till a better local minimum $\hat{s_2}$ on the landscape that is shown in the graphic on the right.

$k \leftarrow k+1$
endif

As can be observed from the description as given above, the choice of the neighborhood structures is the critical point of VNS and VND. The neighborhoods chosen should exploit different properties and characteristics of the search space, that is, the neighborhood structures should provide different *abstractions* of the search space. A variant of VNS is obtained by selecting the neighborhoods in such a way as to produce a problem decomposition (the algorithm is called Variable Neighborhood Decomposition Search – VNDS). VNDS follows the usual VNS scheme, but the neighborhood structures and the local search are defined on sub-problems. For each solution, all attributes (usually variables) are kept fixed except for k of them. For each k a neighborhood structure $\mathcal{N}_k$ is defined. Local search only regards changes on the variables belonging to the sub-problem it is applied to. The inner loop of VNDS is the following:

$s' \leftarrow$ PickAtRandom($\mathcal{N}_k(s)$) % s and s' differ in k attributes
$s'' \leftarrow$ LocalSearch(s',Attributes) % only moves involving the k attributes are allowed
if $(f(s'') < f(s))$ **then**
  $s \leftarrow s''$
  $k \leftarrow 1$
else
  $k \leftarrow k+1$
endif

The decision whether to perform a move can be varied as well. The acceptance criterion based on improvements is strongly steepest descent-oriented and it might not be suited to effectively explore the search space. For example, when local minima are clustered, VNS can quickly find the best optimum in a cluster, but it has no guidance to leave that cluster and find another one. Skewed VNS (SVNS) extends VNS by providing a more flexible acceptance criterion that takes also into account the distance from the current solution.[15] The new acceptance criterion is the following: besides always accepting improvements, worse solutions can be accepted if the distance from the current one is less than a value $\alpha\rho(s,s'')$. The function $\rho(s,s'')$ measures the distance between s and s'' and α is a parameter that weights the importance of the distance between the two solutions in the acceptance criterion. The inner loop of SVNS can be sketched as follows:

if $(f(s'') - \alpha\rho(s,s'') < f(s))$ **then**
  $s \leftarrow s''$
  $k \leftarrow 1$
else
  $k \leftarrow k+1$
endif

[15] A distance measure between solutions has thus to be formally defined.

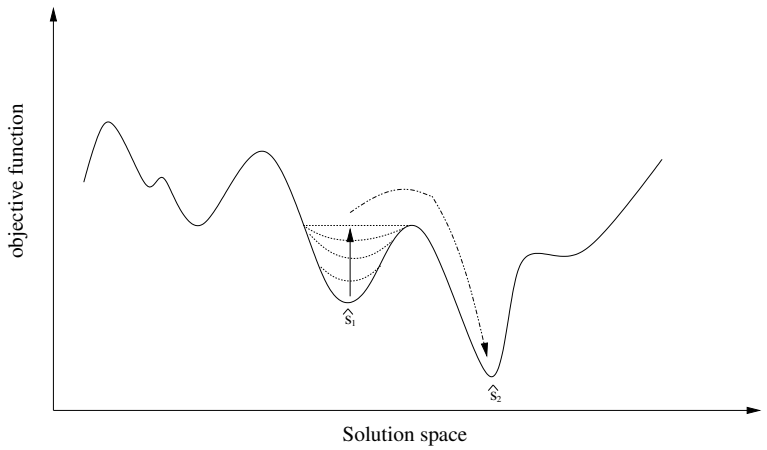

Figure 9: Basic GLS idea: escaping from a valley in the landscape by increasing the objective function value of its solutions.

VNS and its variants have been successfully applied to graph based CO problems such as the p-Median problem [80], the degree constrained minimum spanning tree problem [143], the Steiner tree problem [168] and the k-Cardinality Tree (KCT) problem [116]. References to more applications can be found in [82].

3.4.3 Guided Local Search

Tabu Search and Variable Neighborhood Search explicitly deal with dynamic neighborhoods with the aim of efficiently and effectively exploring the search space. A different approach for guiding the search is to dynamically change the objective function. Among the most general methods that use this approach is Guided Local Search (GLS) [167, 166].

The basic GLS principle is to help the search to gradually move away from local minima by changing the search landscape. In GLS the set of solutions and the neighborhood structure are kept fixed, while the objective function f is dynamically changed with the aim of making the current local optimum "less desirable". A pictorial description of this idea is given in Figure 9.

The mechanism used by GLS is based on *solution features*, which may be any kind of properties or characteristics that can be used to discriminate between solutions. For example, solution features in the TSP could be arcs between pairs of cities, while in the MAXSAT problem they could be the number of unsatisfied clauses. An indicator function $I_i(s)$ indicates whether the feature i is present in solution s:

$$I_i(s) = \begin{cases} 1 & : \text{ if feature } i \text{ is present in solution } s \\ 0 & : \text{ otherwise } . \end{cases}$$

The objective function f is modified to yield a new objective function f' by adding a term that depends on the m features:

$$f'(s) = f(s) + \lambda \sum_{i=1}^{m} p_i \cdot I_i(s) \quad ,$$

where p_i are called *penalty parameters* and λ is called the *regularization parameter*. The penalty parameters weight the importance of the features: the higher p_i, the higher the importance of feature i, thus the higher the cost of having that feature in the solution. The regularization parameter balances the relevance of features with respect to the original objective function.

The algorithm (see Figure 10) works as follows. It starts from an initial solution and applies a local search method until a local minimum is reached. Then the array $\mathbf{p} = (p_1, \ldots, p_m)$ of penalties is updated by incrementing some of the penalties and the local search is started again. The penalized features are those that have the maximum *utility*:

```
s ← GenerateInitialSolution()
while termination conditions not met do
   s ← LocalSearch(s, f')
   for all feature i with maximum utility Util(s,i) do
      p_i ← p_i + 1
   endfor
   Update(f',p)          % p is the penalty vector
endwhile
```

Figure 10: Algorithm: Guided Local Search (GLS)

$$Util(s,i) = I_i(s) \cdot \frac{c_i}{1+p_i} \quad ,$$

where c_i are *costs* assigned to every feature i giving a heuristic evaluation of the relative importance of features with respect to others. The higher the cost, the higher the utility of features. Nevertheless, the cost is scaled by the penalty parameter to prevent the algorithm from being totally biased toward the cost and to make it sensitive to the search history.

The penalties update procedure can be modified by adding a multiplicative rule to the simple incrementing rule (that is applied at each iteration). The multiplicative rule has the form: $p_i \leftarrow p_i \cdot \alpha$, where $\alpha \in (0,1)$. This rule is applied with a lower frequency than the incrementing one (for example every few hundreds of iterations) with the aim of smoothing the weights of penalized features so as to prevent the landscape from becoming too rugged. It is important to note that the penalties update rules are often very sensitive to the problem instance.

GLS has been successfully applied to the weighted MAXSAT [114], the VR problem [98], the TSP and the QAP [167].

3.4.4 Iterated Local Search

We conclude this presentation of explorative strategies with Iterated Local Search (ILS), the most general scheme among the explorative strategies. On the one hand, its generality makes it a framework for other metaheuristics (such as VNS); on the other hand, other metaheuristics can be easily incorporated as subcomponents. ILS is a simple but powerful metaheuristic algorithm [153, 152, 105, 104, 108]. It applies local search to an initial solution until it finds a local optimum; then it perturbs the solution and it restarts local search. The importance of the perturbation is obvious: too small a perturbation might not enable the system to escape from the basin of attraction of the local optimum just found. On the other side, too strong a perturbation would make the algorithm similar to a random restart local search.

A local search is effective if it is able to find good local minima, that is, if it can find the basin of attraction of those states. When the search space is wide and/or when the basins of attraction of good local optima are small,[16] a simple multi-start algorithm is almost useless. An effective search could be designed as a trajectory only in the set of local optima $\hat{\mathcal{S}}$, instead of in the set $\mathcal{S}$ of all the states. Unfortunately, in most cases there is no feasible way of introducing a neighborhood structure for $\hat{\mathcal{S}}$. Therefore, a trajectory along local optima $\hat{s_1}, \hat{s_2}, \ldots, \hat{s_t}$ is performed, without explicitly introducing a neighborhood structure, by applying the following scheme:

1. Execute local search (LS) from an initial state s until a local minimum $\hat{s}$ is found.
2. *Perturb* $\hat{s}$ and obtain s'.
3. Execute LS from s' until a local minimum $\hat{s'}$ is reached.
4. On the basis of an *acceptance criterion* decide whether to set $\hat{s} \leftarrow \hat{s'}$.
5. Goto step 2.

[16] The basin of attraction size of a point s (in a finite space), is defined as the fraction of initial states of trajectories which converge to point s.

$s_0 \leftarrow$ GenerateInitialSolution()
$\hat{s} \leftarrow$ LocalSearch(s_0)
while termination conditions not met **do**
  $s' \leftarrow$ Perturbation($\hat{s}, history$)
  $\hat{s'} \leftarrow$ LocalSearch(s')
  $\hat{s} \leftarrow$ ApplyAcceptanceCriterion($\hat{s}, \hat{s'}, history$)
endwhile

Figure 11: Algorithm: Iterated Local Search (ILS)

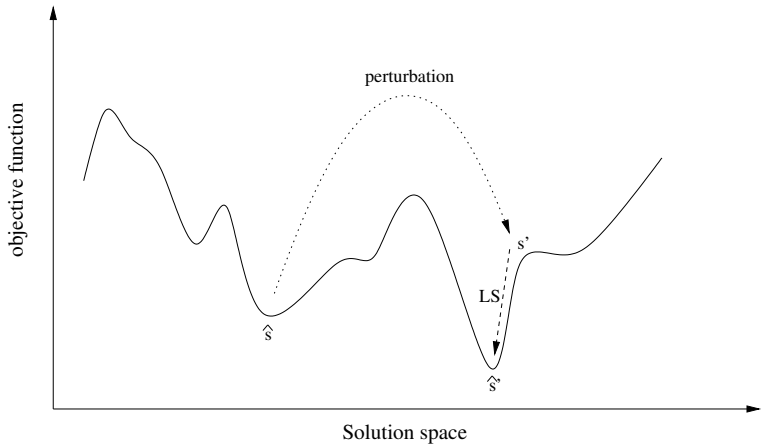

Figure 12: A desirable ILS step: the local minimum $\hat{s}$ is perturbed, then LS is applied and a new local minimum is found.

The requirement on the *perturbation* of $\hat{s}$ is to produce a starting point for local search such that a local minimum different from $\hat{s}$ is reached. However, this new local minimum should be *closer* to $\hat{s}$ than a local minimum produced by a random restart. The *acceptance criterion* acts as a counterbalance, as it filters and gives feedback to the perturbation action, depending on the characteristics of the new local minimum. A high level description of ILS as it is described in [105] is given in Figure 11. Figure 12 shows a possible (lucky) ILS step.

The design of ILS algorithms has several degrees of freedom in the choice of the initial solution, perturbation and acceptance criteria. A key role is played by the *history* of the search which can be exploited both in form of short and long term memory.

The construction of initial solutions should be fast (computationally not expensive), and initial solutions should be a good starting point for local search. The fastest way of producing an initial solution is to generate it at random; however, this is the easiest way for problems that are constrained, whilst in other cases the construction of a feasible solution requires also constraint checking. Constructive methods, guided by heuristics, can also be adopted. It is worth underlining that an initial solution is considered a good starting point depending on the particular LS applied and on the problem structure, thus the algorithm designer's goal is to find a trade-off between speed and quality of solutions.

The perturbation is usually non-deterministic in order to avoid cycling. Its most important characteristic is the *strength*, roughly defined as the amount of changes made on the current solution. The strength can be either fixed or variable. In the first case, the distance between $\hat{s}$ and s' is kept constant, independently of the problem size. However, a variable strength is in general more effective, since it has been experimentally found that, in most of the problems, the bigger the problem size, the larger should be the strength. More sophisticated schemes are possible; for example, the strength can be adaptive: it increases when more diversification is needed and it decreases when intensification seems preferable. VNS and its variants belong to this category. A second choice is the mechanism to perform perturbations. This may be a

random mechanism, or the perturbation may be produced by a (semi–)deterministic method (e.g., a LS different from the one used in the main algorithm).

The third important component is the acceptance criterion. Two extreme examples consist in (1) accepting the new local optimum only in case of improvement and (2) in always accepting the new solution. In-between, there are several possibilities. For example, it is possible to adopt a kind of annealing schedule: accept all the improving new local optima and accept also the non-improving ones with a probability that is a function of the temperature T and the difference of objective function values. In formulae:

$$\mathbf{p}(Accept(\hat{s},\hat{s}',history)) = \begin{cases} 1 & \text{if } f(\hat{s}') < f(\hat{s}) \\ \exp(-\frac{f(\hat{s}')-f(\hat{s})}{T} & \text{otherwise} \end{cases}$$

The cooling schedule can be either monotonic (non-increasing in time) or non-monotonic (adapted to tune the balance between diversification and intensification). The non-monotonic schedule is particularly effective if it exploits the history of the search, in a way similar to the Reactive Tabu Search [157] mentioned at the end of the section about Tabu Search. When intensification seems no longer effective, a diversification phase is needed and the temperature is increased.

Examples for successful applications of ILS are to the TSP [107, 92], to the QAP [105], and to the Single Machine Total Weighted Tardiness (SMTWT) problem [32]. References to other applications can be found in [105].

4 Population-based methods

Population-based methods deal in every iteration of the algorithm with a set (i.e. a population) of solutions[17] rather than with a single solution. As they deal with a population of solutions, population-based algorithms provide a natural, intrinsic way for the exploration of the search space. Yet, the final performance depends strongly on the way the population is manipulated. The most studied population-based methods in combinatorial optimization are Evolutionary Computation (EC) and Ant Colony Optimization (ACO). In EC algorithms, a population of individuals is modified by recombination and mutation operators, and in ACO a colony of artificial ants is used to construct solutions guided by the pheromone trails and heuristic information.

4.1 Evolutionary Computation

Evolutionary Computation (EC) algorithms are inspired by nature's capability to evolve living beings well adapted to their environment. EC algorithms can be succinctly characterized as computational models of evolutionary processes. At each iteration a number of operators is applied to the individuals of the current population to generate the individuals of the population of the next generation (iteration). Usually, EC algorithms use operators called *recombination* or *crossover* to recombine two or more individuals to produce new individuals. They also use *mutation* or *modification* operators which cause a self-adaptation of individuals. The driving force in evolutionary algorithms is the *selection* of individuals based on their *fitness* (this can be the value of an objective function or 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 iteration (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 EC algorithms.

There has been a variety of slightly different EC algorithms proposed over the years. Basically they fall into three different categories which have been developed independently from each other. These are Evolutionary Programming (EP) developed by [57] and [58], Evolutionary Strategies (ES) proposed by [136] and Genetic Algorithms initiated by [89] (see [73], [115], [139] and [163] 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

[17] In general, especially in EC algorithms, we talk about a population of individuals rather than solutions.

```
P ← GenerateInitialPopulation()
Evaluate(P)
while termination conditions not met do
   P' ← Recombine(P)
   P'' ← Mutate(P')
   Evaluate(P'')
   P ← Select(P'' ∪ P)
endwhile
```

Figure 13: Algorithm: Evolutionary Computation (EC)

combinatorial optimization problems. Over the years there have been quite a few overviews and surveys about EC methods. Among those are the ones by [5], by [55], by [149] and by [112]. [17] propose a taxonomy of EC algorithms.

In the following we provide a "combinatorial optimization"-oriented introduction to EC algorithms. For doing this, we follow an overview work by [86], which gives, in our opinion, a good overview of the different components of EC algorithms and of the possibilities to define them.

Figure 13 shows the basic structure of every EC algorithm. 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. The main features of an EC algorithm are outlined in the following.

Description of the individuals: EC algorithms handle populations of individuals. These individuals are not necessarily solutions of the considered problem. They may be partial solutions, or sets of solutions, or any object which can be transformed into one or more solutions in a structured way. Most commonly used in combinatorial optimization is the representation of solutions as bit-strings or as permutations of n integer numbers. Tree-structures or other complex structures are also possible. In the context of Genetic Algorithms, individuals are called *genotypes*, whereas the solutions that are encoded by individuals are called *phenotypes*. This is to differentiate between the representation of solutions and solutions themselves. The choice of an appropriate representation is crucial for the success of an EC algorithm. Holland's schema analysis [89] and Radcliffe's generalization to formae [134] are examples of how theory can help to guide representation choices.

Evolution process: In each iteration it has to be decided which individuals will enter the population of the next iteration. This is done by a selection scheme. To choose the individuals for the next population exclusively from the offspring is called *generational replacement*. If it is possible to transfer individuals of the current population into the next population, then we deal with a so-called *steady state* evolution process.

Most EC algorithms work with populations of fixed size keeping at least the best individual always in the current population. It is also possible to have a variable population size. In case of a continuously shrinking population size, the situation where only one individual is left in the population (or no crossover partners can be found for any member of the population) might be one of the stopping conditions of the algorithm.

Neighborhood structure: A neighborhood function $\mathcal{N}_{EC} : I \to 2^I$ on the set of individuals I assigns to every individual $i \in I$ a set of individuals $\mathcal{N}_{EC}(i) \subseteq I$ which are permitted to act as recombination partners for i to create offspring. If an individual can be recombined with any other individual (as for example in the simple GA) we talk about *unstructured* populations, otherwise we talk about *structured* populations. An example for an EC algorithm that works on structured populations is the Parallel Genetic Algorithm proposed by [119].

Information sources: The most common form of information sources to create offspring (i.e., new individuals) is a couple of parents (two-parent crossover). But there are also recombination operators that operate on more than two individuals to create a new individual (multi-parent crossover), see [46]. More recent developments even use population statistics for generating the individuals of the next population. Examples are the recombination operators called Gene Pool Recombination [121] and Bit-Simulated

Crossover [155] which make use of a distribution over the search space given by the current population to generate the next population.

Infeasibility: An important characteristic of an EC algorithm is the way it deals with infeasible individuals. When recombining individuals the offspring might be potentially infeasible. There are basically three different ways to handle such a situation. The most simple action is to *reject* infeasible individuals. Nevertheless, for many problems (e.g., for timetabling problems) it might be very difficult to find feasible individuals. Therefore, the strategy of *penalizing* infeasible individuals in the function that measures the quality of an individual is sometimes more appropriate (or even unavoidable). The third possibility consists in trying to *repair* an infeasible solution (see [47] for an example).

Intensification strategy: In many applications it proved to be quite beneficial to use improvement mechanisms to improve the fitness of individuals. EC algorithms that apply a local search algorithm to every individual of a population are often called Memetic Algorithms [117, 118]. While the use of a population ensures an exploration of the search space, the use of local search techniques helps to quickly identify "good" areas in the search space.
Another intensification strategy is the use of recombination operators that explicitly try to combine "good" parts of individuals (rather than, for example, a simple one-point crossover for bit-strings). This may guide the search performed by EC algorithms to areas of individuals with certain "good" properties. Techniques of this kind are sometimes called *linkage learning* or *building block learning* (see [74, 161, 169, 83] as examples). Moreover, generalized recombination operators have been proposed in the literature, which incorporate the notion of "neighborhood search" into EC. An example can be found in [135].

Diversification strategy: One of the major difficulties of EC algorithms (especially when applying local search) is the premature convergence toward sub-optimal solutions. The most simple mechanism to diversify the search process is the use of a mutation operator. The simple form of a mutation operator just performs a small random perturbation of an individual, introducing a kind of *noise*. In order to avoid premature convergence there are ways of maintaining the population diversity. Probably the oldest strategies are *crowding* [28] and its close relative, *preselection*. Newer strategies are *fitness sharing* [75], respectively *niching*, whereby the reproductive fitness allocated to an individual in a population is reduced proportionally to the number of other individuals that share the same region of the search space.

This concludes the list of the main features of EC algorithms. EC algorithms have been applied to most CO problems and optimization problems in general. Recent successes were obtained in the rapidly growing bioinformatics area (see for example [56]), but also in multi-objective optimization [22], and in evolvable hardware [147]. For an extensive collection of references to EC applications we refer to [6]. In the following two subsections we are going to introduce two other populations-based methods which are sometimes also regarded as being EC algorithms.

4.1.1 Scatter Search and Path Relinking

Scatter Search and its generalized form called Path Relinking [69, 71] differ from EC algorithms mainly by providing unifying principles for joining (or recombining) solutions based on generalized path constructions in Euclidean or neighborhood spaces. They also incorporate some ideas originating from Tabu Search methods, as, for example, the use of adaptive memory and associated memory-exploiting mechanisms. The template for Scatter Search (respectively, Path Relinking) is shown in Figure 14.

Scatter Search (respectively, Path Relinking) is a search strategy that generates a set of solutions from a chosen set of *reference solutions* corresponding to feasible solutions to the problem under consideration. This is done by making combinations of subsets of the current set of reference solutions. The resulting solutions are called *trial solutions*. These trial solutions may be infeasible solutions and are therefore usually modified by means of a repair procedure that transforms them into feasible solutions. An improvement mechanism is then applied in order to try to improve the set of trial solutions (usually this improvement procedure is a local search). These improved solutions form the set of *dispersed solutions*. The new set of reference solutions that will be used in the next iteration is selected from the current set of reference solutions and the newly created set of dispersed solutions. The components of the pseudo-code, which is shown in Figure 14, are explained in the following:

Initial Phase:
SeedGeneration()
repeat
  DiversificationGenerator()
  Improvement()
  ReferenceSetUpdate()
until the reference set is of cardinality n

Scatter Search/Path Relinking Phase:
repeat
  SubsetGeneration()
  SolutionCombination()
  Improvement()
  ReferenceSetUpdate()
until termination criteria met

Figure 14: Algorithm: Scatter Search and Path Relinking

SeedGeneration(): One or more seed solutions, which are arbitrary trial solutions, are created and used to initiate the remainder of the method.

DiversificationGenerator(): This is a procedure to generate a collection of diverse trial solutions from an arbitrary trial solution (or seed solution).

Improvement(): In this procedure, an improvement mechanism – usually a local search – is used to transform a trial solution into one or more enhanced trial solutions. Neither the input nor the output solutions are required to be feasible, though the output solutions will more usually be expected to be so. It might be necessary to apply repair methods to infeasible solutions.

ReferenceSetUpdate(): The procedure for updating the reference set is responsible for building and maintaining a reference set consisting of a number of "best" solutions found in the course of the algorithm. The attribute "best" covers features such as quality of solutions and diversity of solutions (the solutions in the reference set should be of good quality and they should be diverse).

SubsetGeneration(): This method operates on the reference set, to produce a subset of its solutions as a basis for creating combined solutions.

SolutionCombination(): A procedure to transform a given subset of solutions produced by the subset generation method into one or more combined solutions. In Scatter Search, which was introduced for solutions encoded as points in the Euclidean space, new solutions are created by building linear combinations of reference solutions using both positive and negative weights. This means that trial solutions can be both, inside and outside the convex region spanned by the reference solutions. In Path Relinking the concept of combining solutions by making linear combinations of reference points is generalized to neighborhood spaces. Linear combinations of points in the Euclidean space can be re-interpreted as paths between and beyond solutions in a neighborhood space. To generate the desired paths, it is only necessary to select moves that satisfy the following condition: upon starting from an *initiating solution*, the moves must progressively introduce attributes contributed by a *guiding solution*. Multi-parent path generation possibilities emerge in Path Relinking by considering the combined attributes provided by a set of guiding solutions, where these attributes are weighted to determine which moves are given higher priority.

Scatter Search enjoys increasing interest in recent years. Among other problems it has been applied to multi-objective assignment problems [100] and the Linear Ordering Problem (LOP) [18]. For further references we refer to [72]. Path relinking is often used as a component in metaheuristics such as Tabu Search [102] and GRASP [3, 101].

$P \leftarrow$ InitializePopulation()
while termination criteria not met **do**
  $P_{sel} \leftarrow$ Select(P) % $P_{sel} \subseteq P$
  $\mathbf{p}(\mathbf{x}) = \mathbf{p}(\mathbf{x} \mid P_{sel}) \leftarrow$ EstimateProbabilityDistribution()
  $P \leftarrow$ SampleProbabilityDistribution()
endwhile

Figure 15: Algorithm: Estimation of Distribution Algorithms (EDAs)

4.1.2 Estimation of Distribution Algorithms

In the last decade more and more researchers tried to overcome the drawbacks of usual recombination operators of EC algorithms, which are likely to break good building blocks. So, a number of algorithms – sometimes called Estimation of Distribution Algorithms (EDA) [120] – have been developed (see Figure 15 for the algorithmic framework). These algorithms, which have a theoretical foundation in probability theory, are also based on populations that evolve as the search progresses. EDAs use probabilistic modelling of promising solutions to estimate a distribution over the search space which is then used to produce the next generation by sampling the search space according to the estimated distribution. After every iteration the distribution is re-estimated. For a survey of EDAs see [128].

One of the first EDAs that was proposed for Combinatorial Optimization is called Population-based Incremental Learning (PBIL) [7, 8]. The objective of this method is to create a real valued probability vector (each position corresponds to a binary decision variable) which – when used to sample the search space – generates high quality solutions with high probability. Initially, the values of the probability vector are initialized to 0.5 (for each variable there is equal probability to be set to 0 or 1). The goal of shifting the values of this probability vector in order to generate high quality solutions is accomplished as follows: a number of solution vectors are generated according to the probability vector. Then the probability vector is shifted toward the generated solution vector(s) with highest quality. The distance that the probability vector is shifted depends on the learning rate parameter. Then, a mutation operator is applied to the probability vector. After that, the cycle is repeated. The probability vector can be regarded as a prototype vector for generating high quality solution vectors with respect to the available knowledge about the search space. The drawback of this method is the fact that it does not automatically provide a way to deal with constrained problems. In contrast to PBIL, which estimates a distribution of promising solutions assuming that the decision variables are independent, various other approaches try to estimate distributions taking into account dependencies between decision variables. An example for EDAs regarding pairwise dependencies between decision variables is MIMIC [27] and an example for multivariate dependencies is the Bayesian Optimization Algorithm (BOA) [127].

The field of EDAs is still quite young and much of the research effort is focused on methodology rather than high-performance applications. Applications to Knapsack problems, the Job Shop Scheduling (JSS) problem, and other CO problems can be found in [103].

4.2 Ant Colony Optimization

Ant Colony Optimization (ACO) is a metaheuristic approach proposed in [37, 41, 39]. In the course of this section we keep close to the description of ACO as given in [38]. The inspiring source of ACO is the foraging behavior of real ants. This behavior – as described by [33] – enables ants to find shortest paths between food sources and their nest. While walking from food sources to the nest and vice versa, ants deposit a substance called *pheromone* on the ground. When they decide about a direction to go, they choose with higher probability paths that are marked by stronger pheromone concentrations. This basic behavior is the basis for a cooperative interaction which leads to the emergence of shortest paths.

ACO algorithms are based on a parametrized probabilistic model – the *pheromone model* – that is used to model the chemical pheromone trails. Artificial ants incrementally construct solutions by adding oppor-

```
InitializePheromoneValues(T)
while termination conditions not met do
  for all ants a ∈ A do
    s_a ← ConstructSolution(T,H)
  endfor
  ApplyOnlineDelayedPheromoneUpdate(T,{s_a | a ∈ A})
endwhile
```

Figure 16: Algorithm: Ant System (AS)

tunely defined solution components to a partial solution under consideration.[18] For doing that, artificial ants perform randomized walks on a completely connected graph $\mathcal{G} = (\mathcal{C}, \mathcal{L})$ whose vertices are the solution components $\mathcal{C}$ and the set $\mathcal{L}$ are the connections. This graph is commonly called *construction graph*. When a constrained CO problem is considered, the problem constraints Ω are built into the ants' constructive procedure in such a way that in every step of the construction process only feasible solution components can be added to the current partial solution. In most applications, ants are implemented to build feasible solutions, but sometimes it is unavoidable to also let them construct infeasible solutions. Components $c_i \in \mathcal{C}$ can have associated a *pheromone trail parameter* $\mathcal{T}_i$, and connections $l_{ij} \in \mathcal{L}$ can have associated a *pheromone trail parameter* $\mathcal{T}_{ij}$. The set of all pheromone trail parameters is denoted by $\mathcal{T}$. The values of these parameters – the *pheromone values* – are denoted by τ_i, respectively τ_{ij}. Furthermore, components and connections can have associated a *heuristic value* η_i, respectively η_{ij}, representing *a priori* or run time heuristic information about the problem instance. The set of all heuristic values is denoted by $\mathcal{H}$. These values are used by the ants to make probabilistic decisions on how to move on the construction graph. The probabilities involved in moving on the construction graph are commonly called *transition probabilities*. The first ACO algorithm proposed in the literature is called Ant System (AS) [41]. The pseudo-code for this algorithm is shown in Figure 16. For the sake of simplicity we restrict the following description of AS to pheromone trail parameters and heuristic information on solution components.

In this algorithm, $\mathcal{A}$ denotes the set of ants and s_a denotes the solution constructed by ant $a \in \mathcal{A}$. After the initialization of the pheromone values, at each step of the algorithm each ant constructs a solution. These solutions are then used to update the pheromone values. The components of this algorithm are explained in more detail in the following.

InitializePheromoneValues($\mathcal{T}$): At the beginning of the algorithm the pheromone values are initialized to the same small value $ph > 0$.

ConstructSolution($\mathcal{T}$,$\mathcal{H}$): In the construction phase an ant incrementally builds a solution by adding solution components to the partial solution constructed so far. The probabilistic choice of the next solution component to be added is done by means of transition probabilities, which in AS are determined by the following *state transition rule*:

$$\mathbf{p}(c_r|s_a[c_l]) \quad = \quad \begin{cases} \frac{[\eta_r]^\alpha [\tau_r]^\beta}{\sum_{c_u \in J(s_a[c_l])} [\eta_u]^\alpha [\tau_u]^\beta} & \text{if} \quad c_r \in J(s_a[c_l]) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

In this formula α and β are parameters to adjust the relative importance of heuristic information and pheromone values and $J(s_a[c_l])$ denotes the set of solution components that are allowed to be added to the partial solution $s_a[c_l]$, where c_l is the last component that was added.

ApplyOnlineDelayedPheromoneUpdate($\mathcal{T}$,$\{s_a \mid a \in \mathcal{A}\}$): Once all ants have constructed a solution, the

[18] Therefore, the ACO metaheuristic can be applied to any CO problem for which a constructive procedure can be defined.

```
while termination conditions not met do
  ScheduleActivities
    AntBasedSolutionConstruction()
    PheromoneUpdate()
    DaemonActions()          % optional
  end ScheduleActivities
endwhile
```

Figure 17: Algorithm: Ant Colony Optimization (ACO)

online delayed pheromone update rule is applied:

$$\tau_j \leftarrow (1-\rho)\cdot\tau_j + \sum_{a\in\mathcal{A}} \Delta\tau_j^{s_a} \tag{2}$$

where

$$\Delta\tau_j^{s_a} = \begin{cases} F(s_a) & \text{if } c_j \text{ is a component of } s_a \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where $F : \mathcal{S} \mapsto \mathbb{R}^+$ is a function that satisfies $f(s) < f(s') \Rightarrow F(s) \geq F(s'), \forall s \neq s' \in \mathcal{S}$. $F(\cdot)$ is commonly called the quality function. Furthermore, $0 < \rho \leq 1$ is the pheromone evaporation rate. This pheromone update rule aims at an increase of pheromone on solution components that have been found in high quality solutions.

In the following we describe the more general ACO metaheuristic, which is based on the same basic principles as AS. The ACO metaheuristic framework that is shown in Figure 17 covers all the improvements and extensions of AS which have been developed over the years. It consists of three parts gathered in the ScheduleActivities construct. The ScheduleActivities construct does not specify how these three activities are scheduled and synchronized. This is up to the algorithm designer.

AntBasedSolutionConstruction(): An ant constructively builds a solution to the problem by moving through nodes of the construction graph $\mathcal{G}$. Ants move by applying a stochastic local decision policy that makes use of the pheromone values and the heuristic values on components and/or connections of the construction graph (e.g., see the state transition rule of AS). While moving, the ant keeps in memory the partial solution it has built in terms of the path it was walking on the construction graph.

PheromoneUpdate(): When adding a component c_j to the current partial solution, an ant can update the pheromone trail(s) τ_i and/or τ_{ij} (in case the ant was walking on connection l_{ij} in order to reach component c_j). This kind of pheromone update is called *online step-by-step pheromone update*. Once an ant has built a solution, it can retrace the same path backward (by using its memory) and update the pheromone trails of the used components and/or connections according to the quality of the solution it has built. This is called *online delayed pheromone update*.
Pheromone evaporation is the process by means of which the pheromone trail intensity on the components decreases over time. 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.

DaemonActions(): Daemon actions can be used to implement centralized actions which cannot be performed by single ants. Examples are the use of a local search procedure applied to the solutions built by the ants, 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 can observe the path found by each ant in the colony and choose to deposit extra pheromone on the components used by the ant that built the best solution. Pheromone updates performed by the daemon

are called *offline pheromone updates*.

Within the ACO metaheuristic framework, as shortly described above, the currently best performing versions in practise are Ant Colony System (ACS) [40] and $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [154]. In the following we are going to briefly outline the peculiarities of these algorithms.

Ant Colony System (ACS). The ACS algorithm has been introduced to improve the performance of AS. ACS is based on AS but presents some important differences. First, the daemon updates pheromone trails offline: At the end of an iteration of the algorithm – once all the ants have built a solution – pheromone is added to the arcs used by the ant that found the best solution from the start of the algorithm. Second, ants use a different decision rule to decide to which component to move next in the construction graph. The rule is called *pseudo-random-proportional* rule. With this rule, some moves are chosen deterministically (in a greedy manner), others are chosen probabilistically with the usual decision rule. Third, in ACS, ants perform only online step-by-step pheromone updates. These updates are performed to favor the emergence of other solutions than the best so far.

$\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS). $\mathcal{MM}$AS is also an extension of AS. First, the pheromone trails are only updated offline by the daemon (the arcs that were used by the iteration best ant or the best ant since the start of the algorithm receive additional pheromone). Second, the pheromone values are restricted to an interval $[\tau_{min}, \tau_{max}]$ and the pheromone trails are initialized to their maximum value τ_{max}. Explicit bounds on the pheromone trails prevent that the probability to construct a solution falls below a certain value greater than 0. This means that the chance of finding a global optimum never vanishes during the course of the algorithm.

Recently, researchers have been dealing with finding similarities between ACO algorithms and probabilistic learning algorithms such as EDAs. An important step into this direction was the development of the Hyper-Cube Framework for Ant Colony Optimization (HC-ACO) [16]. An extensive study on this subject has been presented in [172], where the authors present a unifying framework for so-called Model-Based Search (MBS) algorithms. Also, the close relation of algorithms like Population-Based Incremental Learning (PBIL) [8] and the Univariate Marginal Distribution Algorithm (UMDA) [120] to ACO algorithms in the Hyper-Cube Framework has been shown. We refer the interested reader to [172] for more information on this subject. Furthermore, connections of ACO algorithms to Stochastic Gradient Descent (SGD) algorithms are shown in [111].

Successful applications of ACO include the application to routing in communication networks [36], the application to the Sequential Ordering Problem (SOP) [62], and the application to Resource Constraint Project Scheduling (RCPS) [109]. Further references to applications of ACO can be found in [42, 43].

5 A unifying view on Intensification and Diversification

In this section we take a closer look at the concepts of *intensification* and *diversification* as the two powerful forces driving metaheuristic applications to high performance. We give a view on metaheuristics that is characterized by the way intensification and diversification are implemented. Although the relevance of these two concepts is commonly agreed, so far there is no unifying description to be found in the literature. Descriptions are very generic and metaheuristic specific. Therefore most of them can be considered incomplete and sometimes they are even opposing. Depending on the paradigm behind a particular metaheuristic, intensification and diversification are achieved in different ways. Even so, we propose a unifying view on intensification and diversification. Furthermore, this discussion could lead to the goal-directed development of hybrid algorithms combining concepts originating from different metaheuristics.

5.1 Intensification and Diversification

Every metaheuristic approach should be designed with the aim of effectively and efficiently exploring a search space. The search performed by a metaheuristic approach should be "clever" enough to both inten-

sively explore areas of the search space with high quality solutions, and to move to unexplored areas of the search space when necessary. The concepts for reaching these goals are nowadays called intensification and diversification. These terms stem from the TS field [70]. In other fields – such as the EC field – related concepts are often denoted by exploitation (related to intensification) and exploration (related to diversification). However, the terms exploitation and exploration have a somewhat more restricted meaning. In fact, the notions of exploitation and exploration often refer to rather short term strategies tied to randomness, whereas intensification and diversification refer to rather medium and long term strategies based on the usage of memory. As the various different ways of using memory become increasingly important in the whole field of metaheuristics, the terms intensification and diversification are more and more adopted and understood in their original meaning.
An implicit reference to the concept of "locality" is often introduced when intensification and diversification are involved. The notion of "area" (or "region") of the search space and of "locality" can only be expressed in a fuzzy way, as they always depend on the characteristics of the search space as well as on the definition of metrics on the search space (distances between solutions).
The literature provides several high level descriptions of intensification and diversification. In the following we cite some of them.

"Two highly important components of Tabu Search are intensification and diversification strategies. Intensification strategies are based on modifying choice rules to encourage move combinations and solution features historically found good. They may also initiate a return to attractive regions to search them more thoroughly. Since elite solutions must be recorded in order to examine their immediate neighborhoods, explicit memory is closely related to the implementation of intensification strategies. **The main difference between intensification and diversification is that during an intensification stage the search focuses on examining neighbors of elite solutions. [...] The diversification stage on the other hand encourages the search process to examine unvisited regions and to generate solutions that differ in various significant ways from those seen before**." [70]
Later in the same book, Glover and Laguna write: "In some instances we may conceive of intensification as having the function of an intermediate term strategy, while diversification applies to considerations that emerge in the longer run."
Furthermore, they write: "Strategic oscillation is closely linked to the origins of tabu search, and provides a means **to achieve an effective interplay between intensification and diversification**."

"After a local minimizer is encountered, all points in its attraction basin lose any interest for optimization. The search should avoid wasting excessive computing time in a single basin and diversification should be activated. On the other hand, in the assumptions that neighbors have correlated cost function values, some effort should be spent in searching for better points located close to the most recently found local minimum point (intensification). **The two requirements are conflicting and finding a proper balance of diversification and intensification is a crucial issue in heuristics**." [10].

"**A metaheuristic will be successful on a given optimization problem if it can provide a balance between the exploitation of the accumulated search experience and the exploration of the search space to identify regions with high quality solutions in a problem specific, near optimal way**." [153].

"Intensification is to search carefully and intensively around good solutions found in the past search. Diversification, on the contrary, is to guide the search to unvisited regions. These terminologies are usually used to explain the basic elements of Tabu Search, but these are essential to all the metaheuristic algorithms. In other words, various metaheuristic ideas should be understood from the view point of these two concepts, and **metaheuristic algorithms should be designed so that intensification and diversification play balanced roles.**" [171].

"Holland frames adaption as a tension between exploration (the search for new, useful adaptations) and exploitation (the use and propagation of these adaptations). The tension comes about since any move toward exploration – testing previously unseen schemas or schemas whose instances seen so far have low fitness – takes away from the exploitation of tried and true schemas. In any system (e.g., a population

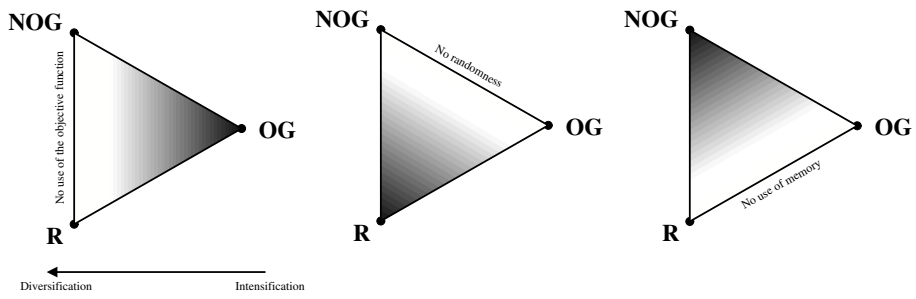

Figure 18: The *I&D frame* provides a unified view on intensification and diversification in metaheuristics (**OG** = I&D components solely guided by the objective function, **NOG** = I&D components solely guided by one or more function other than the objective function, **R** = I&D components solely guided by randomness).

of organisms) required to face environments with some degree of unpredictability, **an optimal balance between exploration and exploitation must be found**. The system has to keep trying out new possibilities (or else it could "over-adapt" and be inflexible in the face of novelty), but it also has to continually incorporate and use past experience as a guide for future behavior." M. Mitchel citing J.H. Holland in 1998.

All these descriptions share the common view that there are two forces for which an appropriate balance has to be found. Sometimes these two forces were described as opposing forces. However, lately some researchers raised the question on how opposing intensification and diversification really are.
In 1998, [48] started a discussion about that in the field of Evolutionary Computation. They question the common opinion about EC algorithms, that they explore the search space by the genetic operators, while exploitation is achieved by selection. In their paper they give examples of operators that one cannot unambiguously label as being either intensification or diversification. So, for example, an operator using a local search component to improve individuals is not merely a mechanism of diversification because it also comprises a strong element of intensification (e.g., in Memetic Algorithms). Another example is the heuristically guided recombination of good quality solutions. If the use of the accumulated search experience is identified with intensification, then a recombination operator is not merely a means of diversification, it also – as in the example above – has a strong intensification component.
Especially the TS literature advocates the view that intensification and diversification cannot be characterized as opposing forces. For example, in [70], the authors write: "Similarly, as we have noted, intensification and diversification are not opposed notions, for the best form of each contains aspects of the other, along a spectrum of alternatives."

Intensification and diversification can be considered as effects of algorithm components. In order to understand similarities and differences among metaheuristics, a framework may be helpful in providing a unified view on intensification and diversification components. We define an *I&D component* as any algorithmic or functional component that has an intensification and/or a diversification effect on the search process. Accordingly, examples of I&D components are genetic operators, perturbations of probability distributions, the use of tabu lists, or changes in the objective function. Thus, I&D components are operators, actions, or strategies of metaheuristic algorithms.

In contrast to the still widely spread view that there are components that have either an intensification or a diversification effect, there are many I&D components that have both. In I&D components that are commonly labelled as intensification, the intensification component is stronger than the diversification component, and vice versa. To clarify this, we developed a framework to put I&D components of different metaheuristics into relation with each other. We called this framework – shown in Figure 18 – the *I&D frame*.

We depict the space of all I&D components as a triangle with the three corners corresponding to three

extreme examples of I&D components. The corner denoted by **OG** corresponds to I&D components solely guided by the objective function of the problem under consideration. An example of an I&D component which is located very close to the corner **OG** is the steepest descent choice rule in local search. The corner denoted by **NOG** covers all I&D components guided by one or more functions other than the objective function, again without using any random component. An example for such a component is a deterministic restart mechanism based on global frequency counts of solution components. The third corner, which is denoted by **R**, comprises all I&D components that are completely random. This means that they are not guided by anything. For example, a restart of an EC approach with random individuals is located in that corner. From the description of the corners it becomes clear that corner **OG** corresponds to I&D components with a maximum intensification effect and a minimum diversification effect. On the other hand, corners **NOG**, **R** and the segment between the two corners correspond to I&D components with a maximum diversification effect and a minimum intensification effect.[19] All I&D components can be located somewhere on or inbetween the three corners, where the intensification effect is becoming smaller the further away a mechanism is located from **OG**. At the same time the diversification effect is growing. In step with this gradient is the use I&D components make of the objective function. The less an I&D component is using the objective function, the further away from corner **OG** it has to be located. There is also a second gradient to be found in this frame (which is shown in the second graphic of Figure 18). Corner **R** stands for complete randomness. The less randomness is involved in an I&D component, the further away from corner **R** it has to be located. Finally, a third gradient describes the influence of criteria different from the objective function, which generally stem from the exploitation of the search history that is in some form kept in the memory. In the following we analyze some basic I&D components intrinsic to the basic versions of the metaheuristics with respect to the I&D frame.

5.2 Basic I&D components of metaheuristics

The I&D components occurring in metaheuristics can be divided in basic (or intrinsic) ones and strategic ones. The basic I&D components are the ones that are defined by the basic ideas of a metaheuristic. On the other side, strategic I&D components are composed of techniques and strategies the algorithm designer adds to the basic metaheuristic in order to improve the performance by incorporating medium and long term strategies. Many of these strategies were originally developed in the context of a specific metaheuristic. However, it becomes more and more apparent that many of these strategies can also be very useful when applied in other metaheuristics. In the following, we exemplary choose some basic I&D components that are inherent to a metaheuristic and explain them in the context of the I&D frame. With that we show that most of the basic I&D components have an intensification character as well as a diversification character.

For many components and strategies of metaheuristics it is obvious that they involve an intensification as well as a diversification component, because they make an explicit use of the objective function. For example, the basic idea of TS is a neighbor choice rule using one or more tabu lists. This I&D component has two effects on the search process. The restriction of the set of possible neighbors in every step has a diversifying effect on the search, whereas the choice of the best neighbor in the restricted set of neighbors (the best non-tabu move) has an intensifying effect on the search. The balance between these two effects can be varied by the length of the tabu list. Shorter tabu lists result in a lower influence of the diversifying effect, whereas longer tabu lists result in an overall higher influence of the diversifying effect. The location of this component in Figure 18 is on the segment between corner **OG** and **NOG**. The shorter the tabu lists, the closer is the location to corner **OG**, and vice versa.
Another example for such an I&D component is the probabilistic acceptance criterion in conjunction with the cooling schedule in SA. The acceptance criterion is guided by the objective function and it also involves a changing amount of randomness. The decrease of the temperature parameter drives the system from diversification to intensification eventually leading to a convergence[20] of the system. Therefore this I&D component is located in the interior of the I&D space between corners **OG**, **NOG** and **R**.
A third example is the following one. Ant Colony Optimization provides an I&D component that manages

[19] There is no quantitative difference between corners **NOG** and **R**. The difference is rather qualitative.
[20] Here we use the term convergence in the sense of getting stuck in the basin of attraction of a local minimum.

the update of the pheromone values. This component has the effect of changing the probability distribution that is used to sample the search space. It is guided by the objective function (solution components found in better solutions than others are updated with a higher amount of pheromone) and it is also influenced by a function applying the pheromone evaporation. Therefore this component is located on the line between corners **OG** and **NOG**. The effect of this mechanism is basically the intensification of the search, but there is also a diversifying component that depends on the greediness of the pheromone update (the less greedy or deterministic, the higher is the diversifying effect).

For other strategies and components of metaheuristics it is not immediately obvious that they have both, an intensification and a diversification effect. An example is the random selection of a neighbor from the neighborhood of a current solution, as it is done for example in the kick-move mechanism of ILS. Initially one might think that there is no intensification involved and that this mechanism has a pure diversification effect caused by the use of randomness. However, for the following reason this is not the case. Many strategies (such as the kick-move operator mentioned above) involve the explicit or implicit use of a neighborhood. A neighborhood structures the search space in the sense that it defines the topology of the so-called *fitness landscape* [150, 151, 93, 97], which can be visualized as a labelled graph. In this graph, nodes are solutions (labels indicate their objective function value) and arcs represent the neighborhood relation between states.[21] A fitness landscape can be analyzed by means of statistical measures. One of the common measures is the *auto-correlation*, that provides information about how much the fitness will change when a move is made from one point to a neighboring one. Different landscapes differ in their *ruggedness*. A landscape with small (average) fitness differences between neighboring points is called *smooth* and it will usually have just a few local optima. In contrast, a landscape with large (average) fitness differences is called *rugged* and it will be usually characterized by many local optima. Most of the neighborhoods used in metaheuristics provide some degree of smoothness that is higher than the one of a fitness landscape defined by a random neighborhood. This means that such a neighborhood is, in a sense, preselecting for every solution a set of neighbors for which the average fitness is not too different. Therefore, even when a solution is randomly selected from a set of neighbors, the objective function guidance is implicitly present. The consequence is that even for a random kick-move there is some degree of intensification involved, as far as a non-random neighborhood is considered.

For a mutation operator of an EC method that is doing a random change of a solution it is neither immediately clear that it can have both, an intensification as well as a diversification effect. In the following we assume a bit-string representation and a mutation operator that is characterized by flipping every bit of a solution with a certain probability. The implicit neighborhood used by this operator is the completely connected neighborhood. However, the neighbors have different probabilities to be selected. The ones that are (with respect to the Hamming distance) closer to the solution to which the operator is applied to, have a higher probability to be generated by the operator. With this observation we can use the same argument as above in order to show an implicit use of objective function guidance. The balance between intensification and diversification is determined by the probability to flip each bit. The higher this probability, the higher the diversification effect of the operator. In contrast, the lower this probability, the higher the intensification effect of this operator.

On the other side, there are some strategies that are often labelled as intensification supposedly without having any diversifying effect. One example is the selection operator in EC algorithms. However, nearly all selection operators involve some degree of randomness (e.g., proportionate selection, tournament selection) and are therefore located somewhere between corners **OG**, **NOG** and **R** of the I&D frame. This means that they also have a diversifying effect. The balance between intensification and diversification depends on the function that assigns the selection probabilities. If the differences between the selection probabilities are quite high, the intensification effect is higher, and similarly for the other extreme of having only small differences between the selection probabilities.
Even an operator like the neighbor choice rule of a steepest descent local search, which might be regarded

[21] The discussion of definitions and analysis of fitness landscapes is beyond the scope of this paper. We forward the interested reader to [150, 151, 93, 94, 59, 90, 97, 138].

Metaheuristic	I&D component
SA	acceptance criterion + cooling schedule
TS	neighbor choice (tabu lists)
	aspiration criterion
EC	recombination
	mutation
	selection
ACO	pheromone update
	probabilistic construction
ILS	black box local search
	kick-move
	acceptance criterion
VNS	black box local search
	neighborhood choice
	shaking phase
	acceptance criterion
GRASP	black box local search
	restricted candidate list
GLS	penalty function

Table 1: I&D-components intrinsic to the basic metaheuristics

as pure intensification, has a diversifying component in the sense that the search is "moving" in the search space with respect to a neighborhood. A neighborhood can be regarded as a function other than the objective function, making implicit use of the objective function. Therefore, a steepest descent local search is located between corners **OG** and **NOG**, and has both, a strong intensification effect but also a weak diversification character.

Based on these observations we conclude that probably most of the basic I&D components used in metaheuristics have both, an intensification and a diversification effect. However, the balance between intensification and diversification might be quite different for different I&D components. Table 1 attempts to summarize the basic I&D components that are inherent to the different metaheuristics.

5.3 Strategic control of intensification and diversification

The right balance between intensification and diversification is needed to obtain an effective metaheuristic. Moreover, this balance should not be fixed or only changing into one direction (e.g., continuously increasing intensification). This balance should rather be dynamical. This issue is often treated in the literature, both implicitly and explicitly, when strategies to guide search algorithms are discussed.

The distinction between intensification and diversification is often interpreted with respect to the temporal horizon of the search. Short term search strategies can be seen as the iterative application of tactics with a strong intensification character (for instance, the repeated application of greedy moves). When the horizon is enlarged, usually strategies referring to some sort of diversification come into play. Indeed, a general strategy usually proves its effectiveness especially in the long term.

The simplest strategy that coordinates the interplay of intensification and diversification and can achieve an oscillating balance between them is the restart mechanism: under certain circumstances (e.g., local optimum is reached, no improvements after a specific number of algorithm cycles, stagnation, no diversity) the algorithm is restarted. The goal is to achieve a sufficient coverage of the search space in the long run, thus the already visited regions should not be explored again. The computationally least expensive attempt to address this issue is a random restart. Every algorithm applying this naive diversification mechanism therefore incorporates an I&D component located in corner **R** of the I&D frame.

Usually, the most effective restart approaches make use of the search history. Examples for such restart strategies are the ones based on concepts such as *global frequency* and *global desirability*. The concept

of *global frequency* is well known from TS applications. In this concept, the number of occurrences of solution components is counted during the run of the algorithm. These numbers, called the global frequency numbers, are then used for changing the heuristic constructive method, for example to generate a new population for restarting an EC method or the initial solution for restarting a trajectory method. Similarly, the concept of *global desirability* (which keeps for every solution component the objective function value of the best solution it had been a member of) can be used to restart algorithms with a bias toward good quality solutions. I&D components based on global frequency can be located in corner **NOG**, while global desirability-based components are located along the segment **NOG-OG**. Examples of the use of non-random restarts can be found also in population-based methods. In EC algorithms the new population can be generated by applying constructive heuristics[22] (line **R-OG**). In ACO, this goal is addressed by smoothing or resetting pheromone values [154]. In the latter case, if the pheromone reset is also based on the search history, the action is located inside the I&D frame.

There are also strategies explicitly aimed at dynamically changing the balance between intensification and diversification during the search. A fairly simple strategy is used in SA, where an increase in diversification and simultaneous decrease in intensification can be achieved by "re-heating" the system and then cooling it down again (which corresponds to increasing parameter T and decreasing it again according to some scheme). Such a cooling scheme is called *non-monotonic* cooling scheme (e.g., see [106] or [124]). Another example can be found in Ant Colony System (ACS). This ACO algorithm uses an additional I&D component aimed at introducing diversification during the solution construction phase. While an ant is walking on the construction graph to construct a solution it reduces the pheromone values on the nodes/arcs of the construction graph that it visits. This has the effect to reduce for the other ants the probability of taking the same path. This additional pheromone update mechanism is called *step-by-step online pheromone update rule*. The interplay between this component and the other pheromone update rules (*online delayed pheromone update rules* and *online pheromone update rule*) leads to an oscillating balance between intensification and diversification.

Some more advanced strategies can be found in the literature. Often, they are described with respect to the particular metaheuristic in which they are applied. However, many of them are very general and can be easily adapted and reused also in a different context. A very effective example is *Strategic Oscillation* [70].[23] This strategy can be applied both to constructive methods and improvement algorithms. Actions are invoked with respect to a critical level (*oscillation boundary*), which usually corresponds to a steady state of the algorithm. Examples for steady states of an algorithm are local minima, completion of solution constructions, or the situation were no components can be added to a partial solution such that it can be completed to a feasible solution. The oscillation strategy is defined by a pattern indicating the way to approach the critical level, to cross it and to cross it again from the other side. This pattern defines the distance of moves from the boundary and the duration of phases (of intensification and diversification). Different patterns generate different strategies; moreover, they can also be adaptive and change depending on the current state and history of the search process. Other representative examples of general strategies that dynamically coordinate intensification and diversification can be found in [11, 14, 15].

Furthermore, strategies are not restricted to single actions (e.g., variable assignments, moves), but may also guide the application of coordinated sequences of moves. Examples of such a strategy are given by so-called *ejection chain* procedures [70, 140, 141]. These procedures provide a mechanism to perform compound moves, i.e., compositions of different types of moves. For instance, in a problem defined over a graph (e.g., the VRP), it is possible to define two different moves: insertion and exchange of nodes; a compound move can thus be defined as the combination of an insertion and an exchange move. These procedures describe general strategies to combine the application of different neighborhood structures, thus they provide an example of a general diversification/intensification interplay. Further examples of strategies which can be interpreted as mechanisms to produce compositions of interlinked moves can also be found in the literature concerning the integration of metaheuristics and complete techniques [19, 146].

[22] See, for example, [60, 77].

[23] Indeed, in [70] and in the literature related to TS, many strategies are described and discussed.

In conclusion, we would like to stress again that most metaheuristic components have both an intensification and a diversification effect. The higher the objective function bias, the higher the intensification effect. In contrast, diversification is achieved by following guiding criteria other than the objective function and also by the use of randomness. With the introduction of the I&D frame, metaheuristics can be analyzed by their signature in the I&D frame. This can be a first step toward the systematic design of metaheuristics, combining I&D components of different origin.

5.4 Hybridization of metaheuristics

We conclude our work by discussing a very promising research issue: the hybridization of metaheuristics. In fact, many of the successful applications that we have cited in previous sections are hybridizations. In the following we distinguish different forms of hybridization. The first one consists of including components from one metaheuristic into another one. The second form concerns systems that are sometimes labelled as cooperative search. They consist of various algorithms exchanging information in some way. The third form is the integration of approximate and systematic (or complete) methods. For a taxonomy of hybrid metaheuristics see [158].

Component exchange among metaheuristics. One of the most popular ways of hybridization concerns the use of trajectory methods in population-based methods. Most of the successful applications of EC and ACO make use of local search procedures. 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 based on the concept 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 allows to make guided steps in the search space which are usually "larger" than the steps done by trajectory methods. In other words, a solution resulting from a recombination in population-based methods is usually more "different" from the parents than, say, a predecessor solution to a successor solution (obtained by applying a move) in TS. We also have "big" steps in trajectory methods like ILS and VNS, but in these methods the steps are usually not guided (these steps are rather called "kick move" or "perturbation" indicating the lack of guidance). It is interesting to note, that in all population-based methods there are mechanisms in which good solutions found during the search influence the search process in the hope to find better solutions in-between those solutions and current solutions. In Path Relinking this idea is implemented in the most explicit way. The basic elements are guiding solutions (which are the good solutions found) and initiating solutions. New solutions are produced by applying moves to decrease the distance between the resulting solution and the guiding solution. In EC algorithms this is often obtained by keeping the best (or a number of best) solution(s) found since the beginning of the respective run of the algorithm in the population. This is called a *steady state* evolution process. Scatter Search performs a steady state process by definition. In some ACO implementations (see for example [154, 15]) a pheromone updating schedule is applied such that in a situation where the algorithm has nearly converged to a solution, only the best found solution since the start of the algorithm is used for updating the pheromone trails. This corresponds to "changing direction" and directing the search process toward a very good solution in the hope to find better ones on the way.

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.

In summary, population-based methods are better in identifying promising areas in the search space, whereas trajectory methods are better in exploring promising areas in the search space. Thus, metaheuristic hybrids that in some way manage to combine the advantage of population-based methods with the strength of trajectory methods are often very successful.

Cooperative search. A loose form of hybridization is provided by *cooperative search* [88, 87, 4, 34, 159, 148, 160], which consists of a search performed by possibly different algorithms that exchange infor-

mation about states, models, entire sub-problems, solutions or other 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 parameters setting. The algorithms composing a cooperative search system can be all approximate, all complete, or a mix of approximate and complete approaches.

Cooperative search nowadays receives more attention, which is among other reasons due to the increasing research on parallel implementations of metaheuristics. The aim of research on parallelization of metaheuristics is twofold. First, metaheuristics should be redesigned to make them suitable for parallel implementation in order to exploit intrinsic parallelism. Second, an effective combination of metaheuristics has to be found, both to combine different characteristics and strengths and to design efficient communication mechanisms. Since the aim of this paper is to provide an overview of the core ideas and strategies of metaheuristics, we refer to [25, 24] for a survey on the state-of-the-art in parallel metaheuristics.

Integrating metaheuristics and systematic methods. Concluding this discussion on hybrid metaheuristics, we briefly overview the integration of metaheuristics and systematic search techniques. This approach has recently produced very effective algorithms especially when applied to real-world problems. Discussions on similarities, differences and possible integration of metaheuristics and systematic search can be found in [61, 65, 84, 70]. A very successful example of such an integration is the combination of metaheuristics and Constraint Programming (CP) [54, 129, 130, 26]. CP enables to model a CO problem by means of variables, domains[24] and constraints, which can be mathematical or symbolic (*global*). The latter ones involve a set of variables and describe subproblems, thus reducing the modelling complexity by encapsulating well defined parts of the problem into single constraints. Every constraint is associated to a *filtering* algorithm that deletes those values from a variable domain that do not contribute to feasible solutions. A CP system can be seen as the interaction of components (constraints) which communicate through shared variables. Constraints are activated as soon as a the domain of any variable involved has been changed. Then, they perform a propagation phase, i.e., they apply the filtering algorithm. This behavior stops as soon as there are no more values that can be removed from the domains or at least one domain is empty (i.e., no feasible solution exists). Since the complexity of the full constraint propagation is often exponential, the filtering is usually not complete. Therefore, at the end of the propagation phase some domains may still contain unfeasible values. Hence, a search phase is started, such as Branch & Bound. A survey on the integration of metaheuristics and CP is provided by [54].

There are three main approaches for the integration of metaheuristics (especially trajectory methods) and systematic techniques (CP and tree search):

- A metaheuristic and a systematic method are sequentially applied (their execution can be also interleaved). For instance, the metaheuristic algorithm is run to produce some solutions which are then used as heuristic information by the systematic search. Vice versa, the systematic algorithm can be run to generate a partial solution which will then be completed by the metaheuristic.

- Metaheuristics use CP and/or tree search to efficiently explore the neighborhood, instead of simply enumerating the neighbors or randomly sampling the neighborhood.

- The third possibility consists of introducing concepts or strategies from either class of algorithms into the other. For example, the concepts of tabu list and aspiration criteria – defined in 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.

The first approach can be seen as an instance of cooperative search and it represents a rather loose integration.

The second approach combines the advantages of a fast search space exploration by means of a metaheuristic with the efficient neighborhood exploration performed by a systematic method. A prominent example of such a kind of integration is *Large Neighborhood Search* and related approaches [146, 19]. These approaches are effective mainly when the neighborhood to explore is very large. Moreover, many

[24] We restrict the discussion to finite domains.

real-world problems have additional constraints (called *side constraints*) which might make them unsuitable for usual neighborhood exploration performed by metaheuristics, since they usually just sample the neighborhood or enumerate its solutions. For instance, time window constraints often reduce the number of feasible solutions in a neighborhood, which might make a local search inefficient. Thus, domain filtering techniques can effectively support neighborhood exploration. In fact, for such a kind of neighborhoods, both sampling and enumeration are usually inefficient. More examples can be found in [129, 130, 54].

The third approach preserves the search space exploration based on a systematic search (such as tree search), but sacrifices the exhaustive nature of the search [65, 84, 85, 113]. The hybridization is usually achieved by integrating concepts developed for metaheuristics (e.g., probabilistic choices, aspiration criteria, heuristic construction) into tree search methods. A typical application of this integration is the use of a probabilistic backtracking, instead of a – deterministic – chronological backtracking. For instance, an extreme case is the random choice of a backtracking move. The list of possible backtracking moves can also be sorted by means of a dynamic heuristic or a *sampling* of the leaves of the search tree. This sampling can be performed by a metaheuristic: the result of each possible backtracking move is chosen as the starting point for producing a complete solution by a metaheuristic (more than one solution can be generated from each partial solution). Then, the quality of these complete solutions is used as a score for – probabilistically – selecting a backtracking move. Another prominent example is the introduction of randomization in systematic techniques, as described in [76]. Many examples of this approach can be found in [54, 96, 144, 30, 133, 29].

6 Conclusions

In this work we have presented and compared nowadays most important metaheuristic methods. In Sections 3 and 4 we have outlined the basic metaheuristics as they are described in the literature. In Section 5 we then proposed a conceptual comparison of the different metaheuristics based on the way they implement the two main concepts for guiding the search process: Intensification and diversification. This comparison is founded on the *I&D frame*, where algorithmic components can be characterized by the criteria they depend upon (objective function, guiding functions and randomization) and their effect on the search process. Although metaheuristics are different in the sense that some of them are population-based (EC, ACO), and others are trajectory methods (SA, TS, ILS, VNS, GRASP), and although they are based on different philosophies, the mechanisms to efficiently explore a search space are all based on intensification and diversification. Nevertheless, it is possible to identify "sub-tasks" in the search process where some metaheuristics perform better than others. This has to be examined more closely in the future in order to be able to produce hybrid metaheuristics performing considerably better than their "pure" parents. In fact we can find this phenomenon in many facets of life, not just in the world of algorithms. Mixing and hybridizing is often better than purity.

Acknowledgments

We would like to thank Marco Dorigo, Joshua Knowles, Andrea Lodi, Michela Milano, Michael Sampels, and Thomas Stützle for suggestions and useful discussions.

Christian Blum acknowledges support by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential program of the CEC, contract HPRN-CT-1999-00106. Andrea Roli acknowledges support by the CEC through a "Marie Curie Training Site" fellowship, contract HPMT-CT-2000-00032. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] E. H. L. Aarts, J. H. M. Korst, and P. J. M. van Laarhoven. Simulated annealing. In Emile H. L. Aarts and Jan Karel Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 91–120.

Wiley-Interscience, Chichester, England, 1997.

[2] E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, UK, 1997.

[3] R. M. Aiex, S. Binato, and M. G. C. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 2003. To appear.

[4] V. Bachelet and E.G. Talbi. Cosearch: a co-evolutionary metaheuritics. In *Proceedings of Congress on Evolutionary Computation – CEC'2000*, pages 1550–1557, 2000.

[5] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[6] T. Bäck, D. B. Fogel, and Z. Machalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing Ltd, Bristol, UK, 1997.

[7] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report No. CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, 1994.

[8] S. Baluja and R. Caruana. Removing the Genetics from the Standard Genetic Algorithm. In A. Prieditis and S. Russel, editors, *The International Conference on Machine Learning 1995*, pages 38–46, San Mateo, California, 1995. Morgan Kaufmann Publishers.

[9] Y. Bar–Yam. *Dynamics of Complex Systems*. Studies in nonlinearity. Addison–Wesley, 1997.

[10] R. Battiti. Reactive search: Toward self-tuning heuristics. In V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, editors, *Modern Heuristic Search Methods*, pages 61–83. John Wiley & Sons, Chichester, UK, 1996.

[11] R. Battiti and M. Protasi. Reactive Search, a history-base heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics*, 2:Article 2, 1997.

[12] R. Battiti and G. Tecchiolli. The Reactive Tabu Search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[13] 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*. Kluwer Academic Publishers, 2001.

[14] C. Blum. ACO Applied to Group Shop Scheduling: A Case Study on Intensification and Diversification. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Proceedings of ANTS 2002 – Third International Workshop on Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 14–27. Springer Verlag, Berlin, Germany, 2002.

[15] C. Blum. Ant Colony Optimization For The Edge-Weighted k-Cardinality Tree Problem. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 27–34, New York, 2002. Morgan Kaufmann Publishers.

[16] C. Blum, A. Roli, and M. Dorigo. HC–ACO: The hyper-cube framework for Ant Colony Optimization. In *Proceedings of MIC'2001 – Meta–heuristics International Conference*, volume 2, pages 399–403, Porto, Portugal, 2001.

[17] 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.

[18] V. Campos, F. Glover, M. Laguna, and R. Martí. An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem. *Journal of Global Optimization*, 21:397–414, 2001.

[19] Y. Caseau and F. Laburthe. Effective Forget-and-Extend Heuristics for Scheduling Problems. In *Proceedings of CP-AI-OR'02 – Fourth Int. Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems*, Ferrara (Italy), 1999. Also available at: www.deis.unibo.it/Events/Deis/Workshops/Proceedings.html.

[20] V. Cerny. A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[21] 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.

[22] C. A. Coello Coello. An Updated Survey of GA-Based Multiobjective Optimization Techniques. *ACM Computing Surveys*, 32(2):109–143, 2000.

[23] D. T. Connolly. An improved annealing scheme for the QAP. *European Journal of Operational Research*, 46:93–100, 1990.

[24] T. G. Crainic and M. Toulouse. Introduction to the special issue on Parallel Meta-Heuristics. *Journal of Heuristics*, 8(3):247–249, 2002.

[25] 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.

[26] B. De Backer, V. Furnon, and P. Shaw. Solving Vehicle Routing Problems Using Constraint Programming and Metaheuristics. *Journal of Heuristics*, 6:501–523, 2000.

[27] J. S. de Bonet, C. L. Isbell Jr., and P. Viola. MIMIC: Finding optima by estimating probability densities. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems (NIPS'97)*, pages 424–431. MIT Press, Cambridge, MA, 1997.

[28] K. A. DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, MI, 1975. Dissertation Abstracts International 36(10), 5140B, University Microfilms Number 76-9381.

[29] 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*, 2003. To appear.

[30] M. Dell'Amico and A. Lodi. On the Integration of Metaheuristic Strategies in Constraint Programming. In C. Rego and B. Alidaee, editors, *Adaptive Memory and Evolution: Tabu Search and Scatter Search*. Kluwer Academic Publishers, Boston, MA, 2002.

[31] 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.

[32] 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 *Proceedings of EvoStim'01*, Lecture Notes in Computer Science, pages 441–452. Springer, April 2001.

[33] 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.

[34] 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.

[35] R. L. Devaney. *An introduction to chaotic dynamical systems*. Addison–Wesley, second edition, 1989.

[36] G. Di Caro and M. Dorigo. AntNet: Distributed Stigmergetic Control for Communication Networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[37] M. Dorigo. *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, DEI, Politecnico di Milano, Italy, 1992. pp. 140.

[38] M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999.

[39] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Art. Life*, 5(2):137–172, 1999.

[40] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997.

[41] M. Dorigo, V. Maniezzo, and A. Colorni. Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 26(1):29–41, 1996.

[42] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications and advances. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 251–285. Kluwer Academic Publishers, Norwell, MA, 2002.

[43] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Boston, MA, 2003. To appear.

[44] G. Dueck. New Optimization Heuristics. *Journal of Computational Physics*, 104:86–92, 1993.

[45] 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.

[46] A. E. Eiben, P.-E. Raué, and Z. Ruttkay. Genetic algorithms with multi-parent recombination. In Y. Davidor, H.-P. Schwefel, and R. Manner, editors, *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, volume 866 of *Lecture Notes in Computer Science*, pages 78–87, Berlin, 1994. Springer.

[47] A. E. Eiben and Z. Ruttkay. Constraint satisfaction problems. In T. Bäck, D. Fogel, and M. Michalewicz, editors, *Handbook of Evolutionary Computation*. Institute of Physics Publishing Ltd, Bristol, UK, 1997.

[48] A. E. Eiben and C. A. Schippers. On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35:1–16, 1998.

[49] W. Feller. *An Introduction to Probability Theory and its Applications*. John Whiley, 1968.

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

[51] 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.

[52] A. Fink and S. Voß. Generic metaheuristics application to industrial engineering problems. *Computers & Industrial Engineering*, 37:281–284, 1999.

[53] 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.

[54] 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.

[55] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, January 1994.

[56] 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.

[57] L. J. Fogel. Toward inductive inference automata. In *Proceedings of the International Federation for Information Processing Congress*, pages 395–399, Munich, 1962.

[58] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966.

[59] 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, 1999.

[60] B. Freisleben and P. Merz. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In *International Conference on Evolutionary Computation*, pages 616–621, 1996.

[61] E. C. Freuder, R. Dechter, M. L. Ginsberg, B. Selman, and E. P. K. Tsang. Systematic Versus Stochastic Constraint Satisfaction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI 1995*, volume 2, pages 2027–2032. Morgan Kaufmann, 1995.

[62] 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.

[63] M. R. Garey and D. S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness*. W.H. Freeman, 1979.

[64] M. Gendreau, G. Laporte, and J.-Y Potvin. Metaheuristics for the Vehicle Routing Problem. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, volume 9 of *SIAM Series on Discrete Mathematics and Applications*, pages 129–154. 2001.

[65] M. L. Ginsberg. Dynamic Backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[66] F. Glover. Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, 8:156–166, 1977.

[67] F. Glover. Future paths for integer programming and links to artificial intelligence. *Comp. Oper. Res.*, 13:533–549, 1986.

[68] F. Glover. Tabu Search Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[69] F. Glover. Scatter search and path relinking. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, Advanced topics in computer science series. McGraw-Hill, 1999.

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

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

[72] F. Glover, M. Laguna, and R. Martí. Scatter Search and Path Relinking: Advances and Applications. 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.

[73] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, Reading, MA, 1989.

[74] D. E. Goldberg, K. Deb, and B. Korb. Don't worry, be messy. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, La Jolla, CA, 1991. Morgan Kaufmann.

[75] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In J. J. Grefenstette, editor, *Genetic Algorithms and their Applications*, pages 41–49. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[76] C. P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-Tayled phenomena in Satisfiability and Constraint Satisfaction Prpblems. *Journal of Automated Reasoning*, 24:67–100, 2000.

[77] J. J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 42–60. Morgan Kaufmann Publishers, 1987.

[78] 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.

[79] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy, 1986.

[80] P. Hansen and N. Mladenović. Variable Neighborhood Search for the *p*-Median. *Location Science*, 5:207–226, 1997.

[81] 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.

[82] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[83] G. Harik. Linkage learning via probabilistic modeling in the ECGA. Technical Report No. 99010, IlliGAL, University of Illinois, 1999.

[84] W. D. Harvey. *Nonsystematic Backtracking Search*. PhD thesis, CIRL, University of Oregon, 1995.

[85] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In Chris S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI 1995*, volume 1, pages 607–615, Montréal, Québec, Canada, 1995. Morgan Kaufmann, 1995.

[86] A. Hertz and D. Kobler. A framework for the description of evolutionary algorithms. *European Journal of Operational Research*, 126:1–12, 2000.

[87] T. Hogg and A. Huberman. Better than the best: The power of cooperation. In *SFI 1992 Lectures in Complex Systems*, pages 163–184. Addison-Wesley, 1993.

[88] T. Hogg and C.P. Williams. Solving the really hard problems with cooperative search. In *Proceedings of AAAI93*, pages 213–235. AAAI Press, 1993.

[89] J. H. Holland. *Adaption in natural and artificial systems*. The University of Michigan Press, Ann Harbor, MI, 1975.

[90] W. Hordijk. A measure of landscapes. *Evolutionary Computation*, 4(4):335–360, 1996.

[91] 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.

[92] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: a case study. In E.H. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, 1997.

[93] T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD thesis, Univ. of New Mexico, Albuquerque, NM, 1995.

[94] T. Jones. One operator, one landscape. Santa Fe Institute Technical Report 95-02-025, Santa Fe Institute, 1995.

[95] D. E. Joslin and D. P. Clements. "Squeaky Wheel" Optimization. *Journal of Artificial Intelligence Research*, 10:353–373, 1999.

[96] N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139:21–45, 2002.

[97] S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.

[98] 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, 1999.

[99] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, 13 May 1983*, 220(4598):671–680, 1983.

[100] M. Laguna, H. Lourenço, and R. Martí. Assigning Proctors to Exams with Scatter Search. In M. Laguna and J. L. González-Velarde, editors, *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pages 215–227. Kluwer Academic Publishers, Boston, MA, 2000.

[101] M. Laguna and R. Martí. GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization. *INFORMS Journal on Computing*, 11(1):44–52, 1999.

[102] M. Laguna, R. Martí, and V. Campos. Intensification and Diversification with Elite Tabu Search Solutions for the Linear Ordering Problem. *Computers and Operations Research*, 26:1217–1230, 1999.

[103] P. Larrañaga and J. A. Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, Boston, MA, 2002.

[104] 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, Porto – Portugal, 2001.

[105] 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.

[106] M. Lundy and A. Mees. Convergence of an annealing algorithm. *Mathematical Programming*, 34(1):111–124, 1986.

[107] O. Martin and S. W. Otto. Combining Simulated Annealing with Local Search Heuristics. *Annals of Operations Research*, 63:57–75, 1996.

[108] 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.

[109] 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.

[110] http://www.metaheuristics.net/, 2000. Visited in January 2003.

[111] N. Meuleau and M. Dorigo. Ant Colony Optimization and Stochastic Gradient Descent. *Artificial Life*, 8(2):103–121, 2002.

[112] 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.

[113] M. Milano and A. Roli. On the relation between complete and incomplete search: an informal discussion. In *Proceedings of CP-AI-OR'02 – Fourth Int. Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 237–250, Le Croisic, France, 2002.

[114] 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.

[115] M. Mitchell. *An introduction to genetic algorithms*. MIT press, Cambridge, MA, 1998.

[116] N. Mladenović and D. Urošević. Variable Neighborhood Search for the k-Cardinality Tree. In *Proceedings of MIC'2001 – Meta–heuristics International Conference*, volume 2, pages 743–747, Porto – Portugal, 2001.

[117] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Toward memetic algorithms. Tech. Rep. Caltech Concurrent Computation Program 826, California Institute of Technology,Pasadena, California, USA, 1989.

[118] P. Moscato. Memetic algorithms: A short introduction. In F. Glover D. Corne and M. Dorigo, editors, *New Ideas in Optimization*. McGraw-Hill, 1999.

[119] H. Mühlenbein. Evolution in time and space – the parallel genetic algorithm. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*. Morgan Kaufmann, San Mateo, USA, 1991.

[120] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the 4th Conference on Parallel Problem Solving from Nature – PPSN IV*, volume 1411 of *Lecture Notes in Computer Science*, pages 178–187, Berlin, 1996. Springer.

[121] H. Mühlenbein and H.-M. Voigt. Gene Pool Recombination in Genetic Algorithms. In I. H. Osman and J. P. Kelly, editors, *Proc. of the Metaheuristics Conference*, Norwell, USA, 1995. Kluwer Academic Publishers.

[122] G. L. Nemhauser and A. L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, 1988.

[123] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job-shop problem. *Management Science*, 42(2):797–813, 1996.

[124] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

[125] I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:513–623, 1996.

[126] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization - Algorithms and Complexity*. Dover Publications, Inc., New York, 1982.

[127] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, pages 525–532, Orlando, FL, 13-17 1999. Morgan Kaufmann Publishers, San Fransisco, CA.

[128] M. Pelikan, D. E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. Technical Report No. 99018, IlliGAL, University of Illinois, 1999.

[129] G. Pesant and M. Gendreau. A view of local search in Constraint Programming. In *Principles and Practice of Constraint Programming - CP'96*, volume 1118 of *Lecture Notes in Computer Science*, pages 353–366. Springer-Verlag, 1996.

[130] G. Pesant and M. Gendreau. A Constraint Programming Framework for Local Search Methods. *Journal of Heuristics*, 5:255–279, 1999.

[131] 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.

[132] 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.

[133] S. Prestwich. Combining the Scalability of Local Search with the Pruning Techniques of Systematic Search. *Annals of Operations Research*, 115:51–72, 2002.

[134] N. J. Radcliffe. Forma Analysis and Random Respectful Recombination. In *Proceedings of the Fourth International Conference on Genetic Algorithms, ICGA 1991*, pages 222–229. Morgan Kaufmann Publishers, San Mateo, California, 1991.

[135] V. J. Rayward-Smith. A unified approach to tabu search, simulated annealing and genetic algorithms. In V. J. Rayward-Smith, editor, *Applications of Modern Heuristics*. Alfred Waller Limited, Publishers, 1994.

[136] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[137] C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publishing, Oxford, England, 1993.

[138] C. R. Reeves. Landscapes, operators and heuristic search. *Annals of Operations Research*, 86:473–490, 1999.

[139] C. R. Reeves and J. E. Rowe. *Genetic Algorithms: Principles and Perspectives. A Guide to GA Theory*. Kluwer Academic Publishers, Boston (USA), 2002.

[140] C. Rego. Relaxed Tours and Path Ejections for the Traveling Salesman Problem. *European Journal of Operational Research*, 106:522–538, 1998.

[141] C. Rego. Node-ejection chains for the vehicle routing problem: Sequential and parallel algorithms. *Parallel Computing*, 27(3):201–222, 2001.

[142] M. G. C. Resende and C. C. Ribeiro. A GRASP for graph planarization. *Networks*, 29:173–189, 1997.

[143] 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.

[144] A. Schaerf. Combining local search and look-ahead for scheduling and constraint satisfaction problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence, IJCAI 1997*, pages 1254–1259. Morgan Kaufmann Publishers, San Mateo, CA, 1997.

[145] A. Schaerf, M. Cadoli, and M. Lenzerini. LOCAL++: a C++ framework for local search algorithms. *Software Practice & Experience*, 30(3):233–256, 2000.

[146] 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*. Springer, 1998.

[147] 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.

[148] 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.

[149] W. M. Spears, K. A. De Jong, T. Bäck, D. B. Fogel, and H. de Garis. An overview of evolutionary computation. In Pavel B. Brazdil, editor, *Proceedings of the European Conference on Machine Learning (ECML-93)*, volume 667, pages 442–459, Vienna, Austria, 1993. Springer Verlag.

[150] P. F. Stadler. Towards a theory of landscapes. In R. Lopéz-Peña, R. Capovilla, R. García-Pelayo, H. Waelbroeck, and F. Zertuche, editors, *Complex Systems and Binary Networks*, volume 461 of *Lecture Notes in Physics*, pages 77–163. Springer Verlag, Berlin, New York, 1995. Also available as SFI preprint 95-03-030.

[151] P. F. Stadler. Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20:1–45, 1996. Also available as SFI preprint 95-07-067.

[152] T. Stützle. Iterated local search for the quadratic assignment problem. Technical report aida-99-03, FG Intellektik, TU Darmstadt, 1999.

[153] T. Stützle. *Local Search Algorithms for Combinatorial Problems - Analysis, Algorithms and New Applications*. DISKI - Dissertationen zur Künstliken Intelligenz. infix, Sankt Augustin, Germany, 1999.

[154] T. Stützle and H. H. Hoos. $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[155] G. Syswerda. Simulated Crossover in Genetic Algorithms. In L.D. Whitley, editor, *Proceedings of the second workshop on Foundations of Genetic Algorithms*, pages 239–255, San Mateo, California, 1993. Morgan Kaufmann Publishers.

[156] http://www.tabusearch.net. Visited in January 2003.

[157] E. Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991.

[158] E-G. Talbi. A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics*, 8(5):541–564, 2002.

[159] 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.

[160] M. Toulouse, K. Thulasiraman, and F. Glover. Multi-level cooperative search: A new paradigm for combinatorial optimization and application to graph partitioning. In *Proceedings of the fifth International Euro-Par Conference on Parallel Processing*, Lecture Notes in Computer Science, pages 533–542, 1999.

[161] C. H. M. van Kemenade. Explicit filtering of building blocks for genetic algorithms. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the 4th Conference on Parallel Problem Solving from Nature – PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 494–503, Berlin, 1996. Springer.

[162] 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.

[163] M. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. Complex Adaptive Systems. MIT Press, 1999.

[164] 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.

[165] S. Voß and D. Woodruff, editors. *Optimization Software Class Libraries*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[166] C. Voudouris. *Guided Local Search for Combinatorial Optimization Problems*. PhD thesis, Department of Computer Science, University of Essex, 1997. pp. 166.

[167] C. Voudouris and E. Tsang. Guided Local Search. *European Journal of Operational Research*, 113(2):469–499, 1999.

[168] 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.

[169] R. A. Watson, G. S. Hornby, and J. B. Pollack. Modeling building-block interdependency. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, University of Wisconsin, Madison, Wisconsin, USA, 1998. Stanford University Bookstore.

[170] 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.

[171] M. Yagiura and T. Ibaraki. On metaheuristic algorithms for combinatorial optimization problems. *Systems and Computers in Japan*, 32(3):33–55, 2001.

[172] M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-based search for combinatorial optimization. Technical Report TR/IRIDIA/2001-15, IRIDIA, Université Libre de Bruxelles, Belgium, 2001.