Chapter 1
Metaheuristics: Intelligent Problem Solving

Marco Caserta and Stefan Voß

Abstract Metaheuristics support managers in decision making with robust tools providing high quality solutions to important problems in business, engineering, economics and science in reasonable time horizons. While finding exact solutions in these applications still poses a real challenge despite the impact of recent advances in computer technology and the great interactions between computer science, management science/operations research and mathematics, (meta-) heuristics still seem to be the methods of choice in many (not to say most) applications. In this chapter we give some insight into the state of the art of metaheuristics. It focuses on the significant progress regarding the methods themselves as well as the advances regarding their interplay and hybridization with exact methods.

1.1 Introduction

The use of heuristics and metaheuristics to solve real world problems is widely accepted within the operations research community. It is well-known that the great majority of complex real world decision problems, when modeled as optimization problems, belong to the class of $\mathcal{NP}$-hard problems. This implies that exact approaches are doomed to fail when dealing with large scale instances, whether they arise from business, engineering, economics or science. Today, decision making processes are increasingly complex and more encompassing, in the sense that more decision variables are used to model complex situations and more input data and parameters are available to capture the complexity of the problems themselves.

Marco Caserta · Stefan Voß
Institute of Information Systems (Wirtschaftsinformatik), University of Hamburg, Von-Melle-Park 5, 20146 Hamburg, Germany
e-mail: {marco.caserta,stefan.voss}@uni-hamburg.de

V. Maniezzo et al., (eds.), *Matheuristics*, Annals of Information Systems 10,
DOI 10.1007/978-1-4419-1306-7_1,

The inherent complexity of real world optimization problems, though, should be interpreted in the light of what complexity analysis really means: on the one hand, the fact that a problem belongs to the class of $\mathcal{NP}$-hard problems implies that there is no knowledge of an algorithm capable of solving the problem itself to optimality in polynomial time with respect to its input size (and many believe that there will never be); on the other hand, it is worth remembering that complexity analysis provides a worst case scenario, i.e., it indicates that, in the worst case, with the growth of the input size, the algorithm will require more and more time/steps to provide a definite answer. However, it is also worth noting that practitioners have observed that it is possible to design *ad-hoc* algorithms which, while not guaranteeing optimal or near-optimal solutions for the whole set of possible instances of a given problem (and, in many cases, not even a simple, single answer can be guaranteed at all), they do provide near-optimal solutions "most" of the times. This is exactly the goal of a metaheuristic designer (analyst), namely, to design an algorithm for which, even though no guarantee about the worst case scenario can be offered, a certain degree of confidence about the performance of the algorithm "most of the times" can still be asserted.

Consequently, the real challenge of the metaheuristic expert is not only to objectively measure the algorithm in terms of solution quality and computational time over problems for which no optimal solution is known (which, in itself, can be a challenging task due to the lack of any benchmark) but also to use sound quantitative methods and techniques to assert the robustness of the algorithm over a wide spectrum of instance types, hence enhancing the usability of the algorithm itself by industry decision makers, e.g., as "optimization modules" within decision support systems.

A first trade-off emerges here: on the one hand, it is highly desirable to design "general purpose" metaheuristics, which do not require problem specific knowledge and can readily be applied to a wide spectrum of problem classes. This has been the line of research of the last decades, during which a number of general purpose metaheuristic paradigms have been proposed, e.g., simulated annealing, genetic algorithms, tabu search, ant colony, etc. The main argument in favor of such paradigms is exactly their general applicability upon a large set of problems, without requiring major re-design or any in-depth knowledge of the problem to be tackled. Consequently, general paradigms seem especially suited for practitioners, who are interested in getting a solution to the problem without investing a huge amount of time in understanding the mathematical properties of the model and in implementing tailor-made algorithms. However, most successful metaheuristics applied to specific problems are also tailored to the problem, or fine-tuned. On the other hand, it has been observed that general purpose metaheuristics are outperformed by hybrid algorithms, which usually are algorithms that combine mathematical programming techniques with metaheuristic-based ideas. Algorithms of this class are tailor-made and specifically designed to exploit the mathematical properties of the problem at hand. While such approaches are

able to provide enhanced performance, an obvious consequence is a reduction in the usability of the algorithm itself. In general, a tailor-made algorithm can be used only for a specific class of problems and, often, the underlying ideas cannot easily be extended towards operating on a different class of problems.

The discussion on metaheuristic based algorithms is further complicated by what has been observed in recent years: these algorithms in general seem to provide varying performance depending upon the sensitivity (skills, expertise, ingenuity, etc.) of the designer in algorithmic fine tuning. In order to maximize algorithmic performance, an instance specific fine tuning might be required. The variability in the results of a metaheuristic presents at least two major drawbacks: (i) On one hand, the issue of *reproducibility* of results arises: It is common knowledge that a proposed algorithm implemented by two different researchers can lead to altogether different results, depending upon factors such as implementation skills, special usage of data structures, and ability in parameter settings, among others. For this reason, it is hard to compare and thoroughly assess a metaheuristic and its performance. (ii) On the other hand, a problem of *performance maximization* is envisioned: One of the commonalities of virtually all proposed metaheuristic paradigms is that they are characterized by a considerable number of parameters, whose value(s) strongly affect the overall performance of the algorithm itself. It is common knowledge that the fine tuning of algorithmic parameters is not only problem-specific, but even instance-specific, i.e., even for a given problem, parameter values should be fine-tuned and adjusted according to instance specific information (e.g., instance size, distribution of its values, etc.).

The purpose of this paper is to provide a survey of the general field of metaheuristics. While we cannot be fully comprehensive in a single paper, in line with the above remarks, some of the issues addressed in this paper are:

- In light of the well-known *no-free-lunch-theorem* [128], which basically states that, on average, no algorithm outperforms all the others, one might wonder what strategy to pursue, i.e., whether the goal of a researcher in the field should be to develop a better general framework able to effectively solve a wider spectrum of problems or, conversely, to tackle each individual problem separately, by designing tailor-made algorithms that fully exploit the mathematical structure and properties of each problem. A recent line of research in the metaheuristic field is concerned with the design of *hybrid* algorithms, where the term hybrid can indicate either the combination of different metaheuristics or the intertwined usage of metaheuristic features with mathematical programming techniques. Consequently, a trade-off between *re-usability* and *performance* of an algorithm arises.
- As highlighted before, no heuristic can guarantee high quality solutions over all possible instances of a given problem class. However, it is at least desirable to present a *robust* behavior over a spectrum of instances belonging to the same problem class. One key factor that seems to have a strong impact on the algorithmic performance is the fine tuning of the algorithm itself. Since the behavior of a metaheuristic is affected by its parameters,

one might wonder how to select a good set of parameter values. An important topic in metaheuristic design is, therefore, the identification and development of techniques for the *fine tuning of algorithmic parameters.*

- Due to the stochastic behavior of some metaheuristics and the lack of commonly accepted and adopted techniques for the evaluation of algorithmic performance, given two algorithms designed to tackle the same class of problems, it is not always possible to "rank" such algorithms in terms of their performance. One direct consequence is that there is still no clear understanding about which features are really successful and under which circumstances. In other words, unless clear standards about metaheuristics are defined, it will be really hard to have a full grasp of, first, which algorithms are better than others and, second, what is the real contribution of each feature of the algorithm upon the overall performance.

The structure of this paper is as follows: We first present general concepts about heuristics and metaheuristics, seen from an operations research perspective. Next, we illustrate some findings from recent research about hybridization, seen as a combination of exact approaches and mathematical programming techniques with metaheuristic frameworks. We also include here some recent contributions about metaheuristic design, i.e., a collection of ideas and thoughts about what should influence which features of a metaheuristic paradigm to be included in the algorithm (e.g., fitness landscape evaluation). Next, we focus on the important issue of metaheuristics fine tuning and calibrations, by introducing some quantitative methods drawn from statistics that have been employed with this goal in mind. We also present some thoughts on the ongoing debate on metaheuristic assessment, *i.e.*, how an objective evaluation of the performance of a metaheuristic can be carried out in order to increase objectivity of the evaluation and reproducibility of the results. Next, we mention some optimization software libraries especially focused on the implementation of general heuristic and metaheuristic frameworks. The development of software libraries for metaheuristics is perceived as a key factor in the emerging of standards in the field. Finally, the last section presents some concluding remarks.

Earlier survey papers on metaheuristics include [19, 121, 122].[1] The general concepts have not become obsolete, and many changes are mainly based upon an update to most recent references. A handbook on metaheuristics is available describing a great variety of concepts by various authors in a comprehensive manner [59].

[1] Here we occasionally rely on [121] and [122] without explicitly quoting at appropriate places for not "disturbing" the readability.

1.2 Basic Concepts and Discussion

The basic concept of heuristic search as an aid to problem solving was first introduced by [93]. A *heuristic* is a technique (consisting of a rule or a set of rules) which seeks (and hopefully finds) *good* solutions at a reasonable computational cost. A heuristic is *approximate* in the sense that it provides (hopefully) a good solution for relatively little effort, but it does not guarantee optimality.

Heuristics provide simple means of indicating which among several alternatives seems to be best. That is, "heuristics are criteria, methods, or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal. They represent compromises between two requirements: the need to make such criteria simple and, at the same time, the desire to see them discriminate correctly between good and bad choices. A heuristic may be a *rule of thumb* that is used to guide one's action." [91]

Greedy heuristics are simple iterative approaches available for any kind of (e.g., combinatorial) optimization problem. A good characterization is their *myopic* behavior. A greedy heuristic starts with a given feasible or infeasible solution. In each iteration there is a number of alternative choices (*moves*) that can be made to transform the solution. From these alternatives which consist in fixing (or changing) one or more variables, a *greedy choice* is made, i.e., the best alternative according to a given measure is chosen until no such transformations are possible any longer.

Usually, a greedy *construction heuristic* starts with an incomplete solution and completes it in a stepwise fashion. Savings and dual algorithms follow the same iterative scheme: Dual heuristics change an infeasible low cost solution until reaching feasibility, savings algorithms start with a high cost solution and realize the highest savings as long as possible. Moreover, in all three cases, once an element is chosen this decision is (usually) not reversed throughout the algorithm, it is kept.

As each alternative has to be measured, in general we may define some sort of *heuristic measure* (providing, e.g., some priority values or some ranking information) which is iteratively followed until a complete solution is build. Usually this heuristic measure is applied in a greedy fashion.

For heuristics we usually have the distinction between finding initial feasible solutions and improving them. In that sense we first discuss local search before characterizing metaheuristics.

1.2.1 Local Search

The basic principle of local search is to successively alter solutions *locally*. Related transformations are defined by neighborhoods which for a given solution

include all solutions that can be reached by one move. That is, neighborhood search usually is assumed to proceed by moving iteratively from one solution to another one by performing some sort of operation. More formally, each solution of a problem has an associated set of neighbors called its *neighborhood*, i.e., solutions that can be obtained by a single operation called transformation or move. Most common ideas for transformations are, e.g., to add or drop some problem specific individual components. Other options are to exchange two components simultaneously, or to swap them. Furthermore, components may be shifted from a certain position into other positions. All components involved within a specific move are called its elements or attributes.

Moves must be evaluated by some *heuristic measure* to guide the search. Often one uses the implied change of the objective function value, which may provide reasonable information about the (local) advantage of moves. Following a greedy strategy, *steepest descent* (SD) corresponds to selecting and performing in each iteration the best move until the search stops at a local optimum. Obviously, savings algorithms correspond to SD.

As the solution quality of local optima may be unsatisfactory, we need mechanisms that guide the search to overcome local optimality. For example, a metaheuristic strategy called iterated local search is used to iterate/restart the local search process after a local optimum has been obtained, which requires some perturbation scheme to generate a new initial solution (e.g., performing some random moves). Of course, more structured ways to overcome local optimality may be advantageous.

A general survey on local search can be found in [1] and the references from [2]. A simple template is provided by [116].

Despite the first articles on this topic already being in the 1970s (cf. Lin and Kernighan [82]), a variable way of handling neighborhoods is still a topic within local search. Consider an arbitrary neighborhood structure N, which defines for any solution s a set of neighbor solutions $N_1(s)$ as a neighborhood of depth $d = 1$. In a straightforward way, a neighborhood $N_{d+1}(s)$ of depth $d + 1$ is defined as the set $N_d(s) \cup \{s' | \exists s'' \in N_d(s) : s' \in N_1(s'')\}$. In general, a large d might be unreasonable, as the neighborhood size may grow exponentially. However, depths of two or three may be appropriate. Furthermore, temporarily increasing the neighborhood depth has been found to be a reasonable mechanism to overcome *basins of attraction*, e.g., when a large number of neighbors with equal quality exist [12, 13].

Large scale neighborhoods and large scale neighborhood search have become an important topic (see, e.g., [5] for a survey), especially when efficient ways are at hand for exploring them. Related research can also be found under various names; see, e.g., [92] for the idea of ejection chains.

1.2.2 Metaheuristics

The formal definition of metaheuristics is based on a variety of definitions from different authors derived from [52]. Basically, a metaheuristic is a top-level strategy that guides an underlying heuristic solving a given problem. In that sense we distinguish between a *guiding process* and an *application process*. The guiding process decides upon possible (local) moves and forwards its decision to the application process which then executes the chosen move. In addition, it provides information for the guiding process (depending on the requirements of the respective metaheuristic) like the recomputed set of possible moves.

According to [58] "metaheuristics in their modern forms are based on a variety of interpretations of what constitutes *intelligent search*," where the term *intelligent search* has been made prominent by Pearl [91] (regarding heuristics in an artificial intelligence context; see also [118] regarding an operations research context). In that sense we may also consider the following definition: "A metaheuristic is an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search spaces using learning strategies to structure information in order to find efficiently near-optimal solutions." [88].

To summarize, the following definition seems to be most appropriate: *"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. The family of metaheuristics includes, but is not limited to, adaptive memory procedures, tabu search, ant systems, greedy randomized adaptive search, variable neighborhood search, evolutionary methods, genetic algorithms, scatter search, neural networks, simulated annealing, and their hybrids."* [124], p. ix.

1.2.2.1 Simulated Annealing

Simulated annealing (SA) extends basic local search by allowing moves to inferior solutions [80, 35]. A basic SA algorithm may be described as follows: Iteratively, a candidate move is randomly selected; it is accepted if it leads to a solution with an improved objective function value compared to the current solution. Otherwise, the move is accepted with a probability depending on the deterioration Δ of the objective function value. The acceptance probability is computed as $e^{-\Delta/T}$, using a temperature T as control parameter. Usually, T is reduced over time for diversification at an earlier stage of the search and to intensify later.

Various authors describe a robust implementation of this general SA approach; see, e.g., [79], [77]. An interesting variant of SA is to strategically reheat the process, i.e., to perform a non-monotonic acceptance function.

Threshold accepting [37] is a modification (or simplification) of SA accepting every move that leads to a new solution that is 'not much worse' than the older one (i.e., it deteriorates not more than a certain threshold, which reduces with a temperature).

1.2.2.2 Tabu Search

The basic paradigm of *tabu search* (TS) is to use information about the search history to guide local search approaches to overcome local optimality (see [58] for a survey on TS). In general, this is done by a dynamic transformation of the local neighborhood. Based on some sort of memory, certain moves may be forbidden, i.e., they are set tabu. As for SA, the search may lead to performing deteriorating moves when no improving moves exist or all improving moves of the current neighborhood are set tabu. At each iteration, a best admissible neighbor may be selected. A neighbor, respectively a corresponding move, is called admissible, if it is not tabu or if an aspiration criterion is fulfilled. An aspiration criterion is a rule to eventually override a possibly unreasonable tabu status of a move. For example, a move that leads to a neighbor with a better objective function value than encountered so far should be considered as admissible.

The most commonly used TS method is based on a *recency-based* memory that stores moves, or attributes characterizing respective moves, of the recent past (*static TS*). The basic idea of such approaches is to prohibit an appropriately defined inversion of performed moves for a given period by storing attributes of the solution in a tabu list and then preventing moves that require the use of attributes in such a list.

Strict TS embodies the idea of preventing cycling to formerly traversed solutions. The goal is to provide necessity and sufficiency with respect to the idea of not revisiting any solution. Accordingly, a move is classified as tabu iff it leads to a neighbor that has already been visited during the previous search. There are two primary mechanisms to accomplish the tabu criterion: First, we may exploit logical interdependencies between the sequence of moves performed throughout the search process, as realized by, e.g., the reverse elimination method (cf., e.g., [53, 120]). Second, we may store information about all solutions visited so far. This may be carried out either exactly or, for reasons of efficiency, approximately (e.g., by using hash codes).

Reactive TS aims at the automatic adaptation of the tabu list length of static TS [15] by increasing the tabu list length when the tabu memory indicates that the search is revisiting formerly traversed solutions. A possible specification can be described as follows: Starting with a tabu list length l of 1 it is increased every time a solution has been repeated. If there has been

no repetition for some iterations, we decrease it appropriately. To accomplish the detection of a repetition of a solution, one may apply a trajectory based memory using hash codes as for strict TS.

There is a large number of additional ingredients that may make TS work well. Examples include restricting the number of neighbor solutions to be evaluated (using candidate list strategies, e.g., [97]), logical tests as well as diversification mechanisms.

1.2.2.3 Evolutionary Algorithms

Evolutionary algorithms comprise a great variety of different concepts and paradigms including genetic algorithms (see, e.g., [73, 60]), evolutionary strategies (see, e.g., [72, 106]), evolutionary programs [48], scatter search (see, e.g., [51, 54]), and memetic algorithms [87]. For surveys and references on evolutionary algorithms see also [49, 9, 85, 99].

Genetic algorithms are a class of adaptive search procedures based on principles derived from the dynamics of natural population genetics. One of the most crucial ideas for a successful implementation of a genetic algorithm (GA) is the representation of an underlying problem by a suitable scheme. A GA starts, e.g., with a randomly created initial population of artificial creatures (strings), a set of solutions. These strings in whole and in part are the base set for all subsequent populations. Information is exchanged between the strings in order to find new solutions of the underlying problem. The mechanisms of a simple GA essentially consist of copying strings and exchanging partial strings. A simple GA uses three operators which are named according to the corresponding biological mechanisms: reproduction, crossover, and mutation. Performing an operator may depend on a *fitness function* or its value (fitness), respectively. As some sort of heuristic measure, this function defines a means of measurement for the profit or the quality of the coded solution for the underlying problem and often depends on the objective function of the given problem.

GAs are closely related to *evolutionary strategies*. Whereas the mutation operator in a GA was argued to serve to protect the search from premature loss of information [60], evolutionary strategies may incorporate some sort of local search procedure (such as SD) with self adapting parameters involved in the procedure. On a simplified scale many algorithms may be classified as evolutionary once they are reduced to the following frame (see [71]). First, an initial population of individuals is generated. Second, as long as a termination criterion does not hold, perform some sort of co-operation and self-adaptation. Self-adaptation refers to the fact that individuals (solutions) evolve independently while co-operation refers to an information exchange among individuals.

Scatter search ideas established a link between early ideas from various sources—evolutionary strategies, TS and GAs. As an evolutionary approach,

scatter search originated from strategies for creating composite decision rules and surrogate constraints (see [51]). Scatter search is designed to operate on a set of points, called reference points, that constitute good solutions obtained from previous solution efforts. The approach systematically generates linear combinations of the reference points to create new points, each of which is mapped into an associated point that yields integer values for discrete variables. Scatter search contrasts with other evolutionary procedures, such as GAs, by providing unifying principles for joining solutions based on generalized path constructions in Euclidean space and by utilizing strategic designs where other approaches resort to randomization. For a very comprehensive treatment of scatter search see [81].

1.2.2.4 Cross Entropy Method

Initially proposed by [104] for the estimation of rare events, the *cross entropy* method (CE) was extended to solve combinatorial optimization problems [28]. The key ingredient in the CE is the identification of a parametric probability distribution function to be used to generate feasible solutions. Given an initial probability distribution function ϕ^0, a converging sequence of ϕ^t is generated in such a way that each subsequent probability distribution function better captures prominent features found in high quality solutions.

At any given iteration t, ϕ^t is used to generate a population of a given cardinality. Each solution is then evaluated according to a specified merit function (or heuristic measure), e.g., the objective function value associated to each random variate, and the stochastic parameters are then updated in such a way that, in the next generation of the population, high quality solutions will have higher probabilities of being generated under the new model. The problem of updating the stochastic parameters can be solved by applying the *maximum likelihood estimator* method upon a set of "elite solutions" of the current population. In other words, given the top $\rho\%$ of the current population, the CE aims at identifying the value of the parameters of the probability distribution function that better "explains" these elite solutions. This corresponds to adjusting the model to better describe the portion of the feasible space in which good solutions have been found. The two-phase process of generation and update is repeated until convergence in probability is reached.

1.2.2.5 Ant Colony Optimization

The *ant colony optimization* (ACO) metaheuristic [32, 33] is a stochastic method based upon the definition of a construction graph and the use of a set of stochastic procedures called artificial ants. A number of frameworks for the update of stochastic parameters have been proposed. The *ant system*

is a dynamic optimization process reflecting the natural interaction between ants searching for food (see, e.g., [32, 33]). The ants' ways are influenced by two different kinds of search criteria. The first one is the local visibility of food, i.e., the attractiveness of food in each ant's neighborhood. Additionally, each ant's way through its food space is affected by the other ants' trails as indicators for possibly good directions. The intensity of trails itself is time-dependent: With time passing, parts of the trails are diminishing, meanwhile the intensity may increase by new and fresh trails. With the quantities of these trails changing dynamically, an autocatalytic optimization process is started forcing the ants' search into most promising regions. This process of interactive learning can easily be modeled for most kinds of optimization problems by using simultaneously and interactively processed search trajectories.

A comprehensive treatment of the ant system paradigm can be found in [33]. To achieve enhanced performance of the ant system it is useful to hybridize it at least with a local search component.

As pointed out by [133], there are a number of commonalities between the CE and the ACO method, especially with respect to the parameter updating rule mechanisms.

1.2.2.6 Corridor Method

The *corridor method* (CM) has been presented by [109] as a hybrid metaheuristic, linking together mathematical programming techniques with heuristic schemes. The basic idea of the CM relies on the use of an exact method over restricted portions of the solution space of a given problem. Given an optimization problem P, the basic ingredients of the method are a very large feasible space $\mathcal{X}$, and an exact method M that could easily solve problem P if the feasible space were not large. Since, in order to be of interest, problem P generally belongs to the class of $\mathcal{NP}$-hard problems, the direct application of method M to solve P becomes unpractical when dealing with real world instances, i.e., when $\mathcal{X}$ is large.

The basic concept of a *corridor* is introduced to delimit a portion of the solution space around the incumbent solution. The optimization method will then be applied within the neighborhood defined by the corridor with the aim of finding an improved solution. Consequently, the CM defines method-based neighborhoods, in which a neighborhood is built taking into account the method M used to explore it. Given a current feasible solution $\mathbf{x} \in \mathcal{X}$, the CM builds a neighborhood of $\mathbf{x}$, say $\mathcal{N}(\mathbf{x})$, which can effectively be explored by employing method M. Ideally, $\mathcal{N}(\mathbf{x})$ should be exponentially large and built in such a way that it could be explored in (pseudo) polynomial time using method M.

1.2.2.7 Pilot Method

Building on a simple greedy algorithm such as, e.g., a construction heuristic the *pilot method* [38, 39] is a metaheuristic not necessarily based on a local search in combination with an improvement procedure. It primarily *looks ahead* for each possible local choice (by computing a so-called "pilot" solution), memorizing the best result, and performing the respective move. (Very similar ideas have been investigated under the acronym *rollout method* [17].) One may apply this strategy by successively performing a greedy heuristic for all possible local steps (i.e., starting with all incomplete solutions resulting from adding some not yet included element at some position to the current incomplete solution). The look ahead mechanism of the pilot method is related to increased neighborhood depths as the pilot method exploits the evaluation of neighbors at larger depths to guide the neighbor selection at depth one.

In most applications, it is reasonable to restrict the pilot process to some *evaluation depth*. That is, the method is performed up to an incomplete solution (e.g., partial assignment) based on this evaluation depth and then completed by continuing with a conventional heuristic. For a recent study applying the pilot method to several combinatorial optimization problems obtaining very good results see [123]. Additional applications can be found, e.g., in [84, 22].

1.2.2.8 Other Methods

*A*daptive memory programming (AMP) coins a general approach (or even thinking) within heuristic search focusing on exploiting a collection of memory components [55, 114]. An AMP process iteratively constructs (new) solutions based on the exploitation of some memory, especially when combined with learning mechanisms supporting the collection and use of the memory. Based on the idea of initializing the memory and then iteratively generating new solutions (utilizing the given memory) while updating the memory based on the search, we may subsume various of the above described metaheuristics as AMP approaches. This also includes exploiting provisional solutions that are improved by a local search approach.

The performance as well as the efficiency of a heuristic scheme strongly depends on its ability to use AMP techniques providing flexible and variable strategies for types of problems (or special instances of a given problem type) where standard methods fail. Such AMP techniques could be, e.g., dynamic handling of operational restrictions, dynamic move selection formulas, and flexible function evaluations.

Consider, as an example, adaptive memory within TS concepts. Realizing AMP principles depends on which specific TS application is used. For example, the reverse elimination method observes logical interdependencies be-

tween moves and infers corresponding tabu restrictions, and therefore makes fuller use of AMP than simple static approaches do.

To discuss the use of AMP in intelligent agent systems, one may use the simple model of ant systems as an illustrative starting point. As ant systems are based on combining constructive criteria with information derived from the pheromone trails, this follows the AMP requirement for using flexible (dynamic) move selection rules (formulas). However, the basic ant system exhibits some structural inefficiencies when viewed from the perspective of general intelligent agent systems, as no distinction is made between successful and less successful agents, no time-dependent distinction is made, there is no explicit handling of restrictions providing protection against cycling and duplication. Furthermore, there are possible conflicts between the information held in the adaptive memory (*diverging trails*).

A natural way to solve large optimization problems is to decompose them into independent sub-problems that are solved with an appropriate procedure. However, such approaches may lead to solutions of moderate quality since the sub-problems might have been created in a somewhat arbitrary fashion. Of course, it is not easy to find an appropriate way to decompose a problem *a priori*. The basic idea of POPMUSIC is to locally optimize subparts of a solution, *a posteriori*, once a solution to the problem is available. These local optimizations are repeated until a local optimum is found. Therefore, POPMUSIC may be viewed as a local search working with a special, large neighborhood. While POPMUSIC has been acronymed by [112] other metaheuristics may be incorporated into the same framework, too (e.g. [107]). Similarly, in the *variable neighborhood search* (VNS) [68] the neighborhood is altered during the search in such a way that different, e.g. increasingly distant, neighborhoods of a given solution are explored. Such method can be enhanced via *decomposition*, as in the *variable neighborhood decomposition search* (VNDS) (see, e.g., [69]).

For large optimization problems, it is often possible to see the solutions as composed of parts (or chunks [129], cf. the term vocabulary building). Considering as an example the vehicle routing problem, a part may be a tour (or even a customer). Suppose that a solution can be represented as a set of parts. Moreover, some parts are more in relation with some other parts so that a corresponding heuristic measure can be defined between two parts. The central idea of POPMUSIC is to select a so-called *seed part* and a set P of parts that are mostly related with the seed part to form a sub-problem.

Then it is possible to state a local search optimization frame that consists of trying to improve all sub-problems that can be defined, until the solution does not contain a sub-problem that can be improved. In the POPMUSIC frame of [112], the set of parts P corresponds precisely to seed parts that have been used to define sub-problems that have been unsuccessfully optimized. Once P contains all the parts of the complete solution, then all sub-problems have been examined without success and the process stops.

Basically, the technique is a gradient method that starts from a given initial solution and stops in a local optimum relative to a large neighborhood structure. To summarize, both, POPMUSIC as well as AMP may serve as a general frame encompassing various other approaches.

1.2.3 Miscellaneous

Target analysis may be viewed as a general learning approach. Given a problem, we first explore a set of sample instances and an extensive effort is made to obtain a solution which is optimal or close to optimality. The best solutions obtained provide *targets* to be sought within the next part of the approach. For instance, a TS algorithm may be used to bias the search trajectory toward already known solutions (or as close to them as possible). This may give some information on how to choose parameters for other problem instances.

A different acronym in this context is *path relinking* (PR) which provides a useful means of intensification and diversification. Here new solutions are generated by exploring search trajectories that combine elite solutions, i.e., solutions that have proved to be better than others throughout the search. For references on target analysis and PR see, e.g., [58].

Considering local search based on *data perturbation*, the acronym *noising method* may be related to the following approach, too. Given an initial feasible solution, the method performs some data perturbation [111] in order to change the values taken by the objective function of a problem to be solved. On the perturbed data a local search may be performed (e.g., following a SD approach). The amount of data perturbation (the noise added) is successively reduced until it reaches zero. The noising method is applied, e.g., in [23] for the clique partitioning problem.

The key issue in designing *parallel algorithms* is to decompose the execution of the various ingredients of a procedure into processes executable by parallel processors. Opposite to ant systems or GAs, metaheuristics like TS or SA, at first glance, have an intrinsic sequential nature due to the idea of performing the neighborhood search from one solution to the next. However, some effort has been undertaken to define templates for parallel local search (see, e.g., [119, 117, 26, 116]). A comprehensive treatment with successful applications is provided in [6]. The discussion of parallel metaheuristics has also led to interesting hybrids such as the combination of a population of individual processes, agents, in a cooperative and competitive nature (see, e.g., the discussion of *memetic algorithms* in [87]) with TS.

Neural networks may be considered as metaheuristics, although we have not considered them here; see, e.g., [108] for a comprehensive survey on these techniques for combinatorial optimization. On the contrary, one may use metaheuristics to speed up the learning process regarding artificial neural networks; see [7] for a comprehensive consideration.

Require: $\mathbf{x}^k$ incumbent solution; Ω^k current set of heuristic parameters
Ensure: $\mathbf{x}^{k+1}$ next solution
1. $\mathcal{N}(\mathbf{x}^k) \leftarrow$ `neighborhood_definition`$(\mathbf{x}^k, \Omega^k)$
2. $\mathbf{x}^{k+1} \leftarrow$ `neighborhood_exploration`$(\mathcal{N}(\mathbf{x}^k))$
3. $\Omega^{k+1} \leftarrow$ `parameters_update()`

Fig. 1.1 General Metaheuristic Iteration.

Stochastic local search (SLS) is pretty much all we know about local search but enhanced by randomizing choices. That is, an SLS algorithm is a local search algorithm making use of randomized choices in generating or selecting candidate solutions for given instances of optimization problems. Randomness may be used for search initialization as well as the computation of search steps. A comprehensive treatment of SLS is given in [75].

Furthermore, recent efforts on problems with multiple objectives and corresponding metaheuristic approaches can be found in [78, 41]. See, e.g., [105] for some ideas regarding GAs and fuzzy multi-objective optimization.

1.3 A Taxonomy

In this section, we present a taxonomy of metaheuristics along a single dimension of analysis. The driving factor is the way in which the neighborhood is defined with respect to each metaheuristic approach. Alternative classifications have been proposed, e.g., in [19, 64].

From a general perspective, each metaheuristic paradigm can be seen as made up by three major ingredients, which are repeatedly used at each iteration until specified stopping criteria are reached. A generalization of a single iteration of a metaheuristic scheme is given in Figure 1.1.

As illustrated in Step 1 of Figure 1.1, a common ingredient of each metaheuristic paradigm is the existence of a rule aimed at iteratively guiding the search trajectory, i.e., a set of rules to define a neighborhood. In turn, such neighborhood demarcates which solutions can be reached starting from the incumbent solution. In line with this observation, a possible dimension along which a taxonomy of metaheuristics can be created is given by the way in which neighborhoods are built. A classification of metaheuristics along this dimension leads to the definition of at least two broad classes:

- *model-based heuristics*: as in [133], with this term we refer to metaheuristic schemes where new solutions are generated by using a model. Consequently, the neighborhood is implicitly defined by a set of parameters, and iteratively updated during the search process;

- *method-based heuristics*: as in [109], with this term we make reference to heuristic paradigms in which new solutions are sought in a neighborhood whose structure is dictated by the method used to explore the neighborhood itself. Consequently, the neighborhood is implicitly defined by the predetermined method employed.

By observing the underlying philosophy of these two broad classes, a clear dichotomy arises: on the one hand, model-based heuristics tackle the original optimization problem by defining and iteratively updating a model aimed at identifying prominent features in good solutions and at replicating these features in future solutions. Consequently, what determines whether a point belongs to the neighborhood is the set of parameters that defines the model and the 'probability' of generating such point under the current model. On the other hand, method-based heuristics are driven by the technique used to solve a "reduced" version of the original problem, i.e., a problem in which only a subset of the original solution space is considered. Consequently, in method-based heuristics what dictates the structure and shape of the neighborhood is the optimization method employed to explore the neighborhood itself, whether it be a classical mathematical programming technique, e.g., branch and bound, dynamic programming, etc., or a simple enumeration-based technique.

Model-based heuristics are generally based upon the identification of a set of parameters, defining a model that, in turn, well captures some features of the search space. This type of heuristics heavily relies on a set of update schemes, used to progressively modify the model itself in such a way that, after each update, the possibility of obtaining higher quality solutions under the new model is increased. Consequently, in Step 1 of the general `Metaheuristic_Iteration()`, all the solutions that "comply" with the requirements enforced by the model upon the search space are included in the current neighborhood. A special role is played by Step 3 of the same algorithm, in which the parameters of the model are updated via the application of learning mechanisms. In this phase, modifications are applied to the model and/or its parameters to reflect insight collected and generated during the search phase.

A well-known paradigm that can be interpreted under the philosophy of the model-based method is the CE, where a stochastic model is continually updated to reflect the findings of the last iteration of the search process. Other metaheuristics that can be seen as model-based are ACO (as well as other methods belonging to the *Swarm Intelligence* field), where a construction graph and stochastic procedures called *ants* are employed; semi-greedy heuristics [70], including the greedy randomized adaptive search procedure *GRASP* [43], where the greedy function that guides the selection of the best candidates defines a stochastic model; and GAs, where adaptive search procedures based upon genetics are put into place. The GA paradigm heavily relies on a model, defined by a set of operators, that determines which solutions will be included in the next generation, i.e., in the current neighborhood.

More generally, evolutionary algorithms could similarly be included into the category of model-based metaheuristics.

On the other side of the spectrum we find metaheuristic paradigms driven by a method, rather than by a model. The basic ingredient of such an approach is the existence of a search method, whether it be an exact method or a heuristic method, that is used to explore the neighborhood itself. Consequently, the size and cardinality of the neighborhood depend upon the ability of the method itself to explore the portion of the search space included in the neighborhood. Within the class of method-based metaheuristics we can introduce a further level of classification, according to the nature of the method employed to explore the neighborhood. Broadly speaking, method-based heuristics could be divided into two categories, those for which classical mathematical programming techniques are employed to explore the neighborhood and those for which enumeration-based techniques are used to conduct the exploration of the basin of solutions.

A cardinal concept for the method-based heuristics is connected to the introduction of a "distance" metric. A distance is used to draw the boundaries of the neighborhood around the incumbent solution, in such a way that only points whose distance from the incumbent solution is within a threshold are included in the neighborhood itself. Consequently, the neighborhood is explicitly defined by the notion of distance adopted. On the other hand, the definition of the threshold distance is strongly connected with the capabilities of the method used to explore the neighborhood. In other words, the cardinality of the neighborhood is chosen in such a way that, on the one hand, it will be large enough to have a reasonable chance of containing a solution better than the incumbent one and, on the other hand, small enough to be explored by employing the method at hand in a reasonable amount of computational time.

Historically, the first metaheuristics developed might be regarded as belonging to this class, e.g., SA or TS. Let us consider, *e.g.*, the TS metaheuristic. Given an incumbent solution $\mathbf{x}^i$, a distance function $d(\mathbf{x}_1, \mathbf{x}_2)$ and a threshold value δ, only solutions for which $d(\mathbf{x}^i, \mathbf{x}) \leq \delta$ will be included into the current neighborhood. Once the neighborhood definition phase is terminated, a method capable of exploring such neighborhood in a reasonable amount of computational time is employed to find a possibly better solution. Consequently, while the neighborhood is explicitly defined by the value of δ, it is possible to say that such neighborhood is implicitly defined by the method used, since the value of δ depends upon the capabilities of the method itself.

It is worth noting that these metaheuristics can, and in general, do use a set of parameters to refine the definition of the neighborhood. For example, let us once more consider the case of the TS metaheuristic. The neighborhood is first defined according to a "distance" from the incumbent solution and, then, refined via the application of, *e.g.*, the tabu list, with the effect of eliminating some of the solutions from the current neighborhood. However, it should be

evident that the major ingredient used in determining the neighborhood is still related to the concept of distance from the incumbent solution.

Other metaheuristics that fit into this category are VNS [68] and the *pilot method*. In VNS, the neighborhood is altered during the search in such a way that increasingly distant neighborhoods of a given solution are explored. However, a "method", e.g., the local search routine, must be applied to perform the search over the neighborhood. Thus, the way in which neighborhoods are built is influenced by the method used to explore the portion of the search space at hand. Similarly, in the pilot method, the core idea is that the neighborhood is defined by a look-ahead mechanism. Consequently, there is a method (even a simple local search) that determines the shape, or the deepness, of the neighborhood itself.

More recently, metaheuristics employing classical mathematical programming techniques to explore the neighborhood have been proposed. Let us consider the case of the CM. After defining a corridor around the incumbent solution, an optimization method is then applied within the neighborhood defined by the corridor with the aim of finding an improved solution. Consequently, in Step 1 of the general metaheuristic iteration of Figure 1.1, only solutions within the corridor, or within a predefined distance from the incumbent, will be included in the neighborhood. As previously mentioned, an optional feature of the `neighborhood_definition()` phase is the application of a set of criteria to refine the neighborhood, e.g., a tabu list, aspiration criteria, etc. Step 2 of the algorithm relies on the use of either an enumeration-based technique, or a classical mathematical programming technique (branch and bound, dynamic programming, etc.) to explore the neighborhood. Finally, Step 3 of the algorithm, `parameters_update()` can include, e.g., the dynamic update of distance and corridor parameters, depending upon the current status of the search.

Beside the corridor method, other concepts that fall into this category are, e.g., constraint programming [103], in which the corridor is defined by the subsequent application of constraints and conditions to be satisfied by the solution, and local branching [47], in which the neighborhood is defined by introducing linear inequalities, called local branching cuts. Once a restricted neighborhood is so defined, an exact technique, i.e., linear programming, is used in the spirit of the branch and bound framework.

A further example which might be interpreted as a method-based technique is the POPMUSIC framework [112], where one wants to solve, preferably to optimality, smaller portions of the solution space, based upon an available feasible solution.

More recently, the *relaxation induced neighborhood search* method (RINS) has been introduced [27]. The RINS defines a neighborhood exploiting information contained in the linear programming (LP) relaxation and can naturally be seen as a method-based framework for mixed integer programs (MIP). The central idea of the method is related to the exploitation of a "relaxed" solution to define a core problem, smaller than the original MIP. The core

problem is identified at any given node of the branch and cut tree, by first fixing all variables that have the same values both in the incumbent (feasible) solution and the relaxed solution. Next, a branch-and-cut framework is used to solve to optimality the reduced problem on the remaining variables, called sub-MIP. The process is iteratively applied at any node of the global tree, since the LP relaxation induced solution is different and, therefore, gives raise to different sub-MIPs.

1.4 Hybrids with Exact Methods

In recent years, a lot of attention has been devoted to the integration, or hybridization, of metaheuristics with exact methods (see, e.g., [95, 96] for a survey and a taxonomy about hybrid approaches in combinatorial optimization, respectively.) In this section, we use the term *hybrid* in a somehow restrictive way, since we classify as *hybrid approaches* only those approaches that combine the use of exact techniques with metaheuristic frameworks. Consequently, algorithms that combine together different metaheuristics are not included in this analysis although they could be and are also termed hybrid.

This exposition also relates to the term *Matheuristics*, which describes works that also are along these lines, e.g., exploiting mathematical programming techniques in (meta)heuristic frameworks or on granting to mathematical programming approaches the cross-problem robustness and constrained-CPU-time effectiveness which characterize metaheuristics. Discriminating landmark is some form of exploitation of the mathematical formulation of the problems of interest [67].

Generally speaking, hybrid algorithms present a so-called "master-slave" structure of a guiding process and an application process. Either (i) the metaheuristic acts at a higher level and controls the calls to the exact approach, or (ii) the exact technique acts as the master and calls and controls the use of the metaheuristic scheme.

Hybrid algorithms of type (i) are such that the *definition* of the neighborhood follows the logic of a metaheuristic, while the *exploration* of the neighborhood itself is left to the exact approach. From this perspective, the metaheuristic acts as the master by defining the size and boundaries of the neighborhood and by controlling repeated calls to the exact method, which, in turn, acts as an application process, by exploring each neighborhood in an exact fashion. Algorithms that fall into this category are, e.g., those inspired by the CM, in which large scale neighborhoods are searched exhaustively through an exact method applied on a sub-portion of the search space. The call to the exact method is managed by a scheme that heuristically defines a corridor around an incumbent solution (cf. the previous section).

A similar philosophy is shared by the *large scale neighborhood search* (see, e.g., [5] for a survey), in which exponentially large neighborhoods are searched to optimality by means of, e.g., ad-hoc enumeration schemes, dynamic programming schemes, etc.

Along the same line, the RINS as well as local branching could be seen as algorithms of type (i), at least in *spirit*. For example, even though the RINS is casted into a branch and cut approach, and, therefore, the guiding process is an exact approach, the logic of the method is centered upon metaheuristic-type features, such as neighborhood definition, diversification and intensification. It is worth noting, though, that in these two approaches, no real metaheuristic is ever deployed, since they entirely rely on the branch and bound framework. However, they can still be seen as hybrid approaches because of the embedded metaheuristic philosophy that drives the search process.

On the other hand, we also have hybrid approaches of type (ii), in which the metaheuristic scheme is embedded into the solver. Modern branch and cut solvers exploit the potentials of (meta)heuristics to quickly get good quality solutions, especially at early stages of the tree exploration. Related bounds are then employed to prune branches of the tree and, consequently, contribute to speed up the search process and to reduce the overall computational effort. Since it has been observed that in some important practical cases MIP solvers spend a large amount of computational time before finding the first feasible solution, [46] introduced a heuristic scheme, further improved in [3, 16], called the *feasibility pump*, aimed at quickly finding good quality initial solutions. Such initial solutions are obtained via a sequence of roundings, based upon continuous relaxation solutions, that converge to a feasible MIP solution. Clearly, such heuristic-type schemes can also be used to quickly find initial solutions to be fed to type (i) hybrid algorithms, such as the CM, the RINS as well as local branching (see also [56, 57, 40] for heuristic methods for MIP feasible solution generation).

In a fashion similar to hybrid approaches of type (ii), some researchers have also employed metaheuristic schemes for column generation and cut generation within branch and price and branch and cut frameworks, respectively (see, e.g., [44] and [94]). In addition, one may investigate hybrids of branch and bound and metaheuristics, e.g., for deciding upon branching variables or search paths to be followed within a branch and bound tree (see, e.g., [130] for an application of reactive TS). Here we may also use the term *cooperative solver*.

A key question that arises when designing a hybrid algorithm concerns which components should be "hybridized" to create an effective algorithm. While providing an all-encompassing rule for hybridization does not seem to be a feasible approach, from the analysis of the state of the art of hybrid algorithms some interesting guidelines emerge.

A method-based approach is centered upon the exploitation of an effective "method" to solve the problem at hand. Consequently, the starting point lies

in the identification of the most effective(s) method(s) with respect to the optimization problem. For example, the design of a CM inspired algorithm requires previous knowledge about which method could effectively tackle the problem if this were of reduced size. Thus, the identification of the method to be used constitutes the central point in the design of the algorithm itself.

A basic ingredient of a method-based algorithm concerns the heuristic rule used to draw the boundaries of the neighborhood upon which the method will be applied, which is the *design of the neighborhood* itself in terms of how large the neighborhood should be. Size and boundaries of such neighborhood depend on the "power" of the method used, i.e., on its ability to explore large portions of the solution space in a reasonable amount of computational time. While determining the appropriate dimension of the neighborhood for the method at hand is an issue of algorithm fine tuning (as presented in Section 1.6), some general considerations are related to the fitness landscape analysis as well as the computational complexity of the method itself. Roughly speaking, given an optimization method and its worst case computational complexity in terms of size of the input, it is possible to determine the maximum size of the neighborhood that guarantees running times below a desired threshold. On the other hand, since computational complexity analysis mainly deals with worst case scenarios, it seems beneficial to employ fitness landscape analysis techniques (e.g., connectivity measures) to draw tightest complexity bounds that translate directly into larger neighborhoods.

Another guideline is provided by the *intensification-diversification trade-off*. By reviewing hybrid algorithms proposed in the literature, many times it is possible to identify a predominant focus, in the sense that some algorithms put a higher emphasis on diversification of solutions, while others emphasize the intensification of the search in promising regions. For example, as illustrated in [47], the application of valid inequalities in the spirit of local branching fosters the intensification of the search in a given neighborhood, hence allowing to find good quality solutions early on in the search process. In a similar fashion, the CM seems to put more emphasis on intensifying the search within a promising region, without defining specific restarting mechanisms to achieve diversification. On the other hand, a method such as the RINS, based upon a solution of the LP relaxation of the MIP, puts more emphasis on diversification, since at each node of the search tree a different LP induced solution is used and, consequently, different solutions feasible with respect to the MIP will be produced.

Finally, an interesting line of research aimed at grasping a clearer understanding of why some search techniques are successful on a given problem class is related to the *fitness landscape analysis*. As mentioned in [18, 115], a central measure of landscape structure is the fitness-distance correlation, which captures the correlation between objective function value and the length of a path to an optimal solution within the fitness landscape. Such a measure is used to explain why local search techniques perform well in tackling certain problems. However, the link between problem difficulty and

fitness landscape is, as of today, not completely understood (see also the idea of target analysis mentioned above).

Fitness landscape analysis can be used with at least two goals in mind:

- on the one hand, as brought out in [126], this kind of analysis helps to understand what makes a problem hard or, conversely, well suited, for a specific search technique. Information such as fitness-distance correlation, ruggedness, nodes connectivity, and drifting can be exploited to design an effective metaheuristic scheme as well as to identify which components should be hybridized;
- on the other hand, as illustrated in [24], fitness landscape analysis can help to identify which formulation of the same problem will be more suitable with respect to an available algorithm. For example, [24] were able to "predict" the behavior of a VNS algorithm upon two different formulations of the Golomb Ruler problem and, consequently, to select the formulation that better fitted with the potentials of their algorithm.

While this field of study seems promising in grasping a better understanding of the "hows" and "whys" of metaheuristics, an important issue of generalization of results has already been noticed. As mentioned in [126], the results obtained so far have mainly been used *a posteriori*, to justify the use of a given algorithm and its features. However, it is unclear how this kind of analysis can be extended to develop improved algorithms, since there is no clear understanding about the general validity of the findings of the proposed models. However, it is worth noting that, from the methodological perspective, the contribution of the fitness landscape analysis is far from negligible, since its focus is well oriented toward interpreting and partially explaining successes and failures of metaheuristic-based algorithms.

1.5 General Frames: A Pool-Template

An important avenue of metaheuristics research refers to general frames (e.g., to explain the behavior and the relationship between various methods) as well as the development of software systems incorporating metaheuristics (eventually in combination with other methods). Besides other aspects, this takes into consideration that in metaheuristics it has very often been appropriate to incorporate a certain means of diversification versus intensification to lead the search into new regions of the search space. This requires a meaningful mechanism to detect situations when the search might be trapped in a certain area of the solution space. Therefore, within intelligent search the exploration of memory plays a most important role.

In [64] a *pool template* (PT) is proposed as can be seen in Figure 1.2. The following notation is used. A pool of $p \geq 1$ solutions is denoted by P. Its input and output transfer is managed by two functions which are called *IF*

```
1.  Initialize P by an external procedure
WHILE termination=FALSE DO BEGIN
    2.  S := OF(P)
    3.  IF s > 1 THEN S' := SCM(S) ELSE S' := S
    4.  S'' := IM(S')
    5.  P := IF(S'')
END
6.  Apply a post-optimizing procedure to P
```

Fig. 1.2 Pool Template.

and OF, respectively. S is a set of solutions with cardinality $s \geq 1$. A solution combination method (procedure SCM) constructs a solution from a given set S, and IM is an improvement method.

Depending on the method used, in Step 1 either a pool is completely (or partially) built by a (randomized) diversification generator or filled with a single solution which has been provided, e.g., by a simple greedy approach. Note that a crucial parameter that deserves careful elaboration is the cardinality p of the pool. The main loop, executed until a termination criterion holds, consists of Steps 2–5. Step 2 is the call of the output function which selects a set of solutions, S, from the pool. Depending on the kind of method represented in the PT, these solutions may be assembled (Step 3) to a (set of) working solution(s) S' which is the starting point for the improvement phase of Step 4. The outcome of the improvement phase, S'', is then evaluated by means of the input function which possibly feeds the new solution into the pool. Note that a post-optimizing procedure in Step 6 is for facultative use. It may be a straightforward greedy improvement procedure if used for single-solution heuristics or a pool method on its own. As an example we quote a *sequential* pool method, the TS with PR in [11]. Here a PR phase is added *after* the pool has been initialized by a TS. A *parallel* pool method on the other hand uses a pool of solutions *while* it is constructed by the guiding process (e.g., a GA or scatter search).

Several heuristic and metaheuristic paradigms, whether they are obviously pool-oriented or not, can be summarized under the common PT frame. We provide the following examples:

a) Local Search/SD: PT with $p = s = 1$.
b) SA: $p = 2, s = 1$ incorporating its probabilistic acceptance criterion in IM. (It should be noted that $p = 2$ and $s = 1$ seems to be unusual at first glance. For SA we always have a current solution in the pool for which one or more neighbors are evaluated and eventually a neighbor is found which replaces the current solution. Furthermore, at all iterations throughout the search the so far best solution is stored, too (even if no real interaction between those two stored solutions takes place). The same is also valid for