54. Stochastic Local Search Algorithms: An Overview

Holger H. Hoos, Thomas Stützle

In this chapter, we give an overview of the main concepts underlying the stochastic local search (SLS) framework and outline some of the most relevant SLS techniques. We also discuss some major recent research directions in the area of stochastic local search. The remainder of this chapter is structured as follows. In Sect. 54.1, we situate the notion of SLS within the broader context of fundamental search paradigms and briefly review the definition of an SLS algorithm. In Sect. 54.2, we summarize the main issues and trends in the design of greedy constructive and iterative improvement algorithms, while in Sects. 54.3–54.5, we provide a concise overview of some of the most widely used simple, hybrid, and population-based SLS methods. Finally, in Sect. 54.6, we discuss some recent topics of interest, such as the systematic design of SLS algorithms and methods for the automatic configuration of SLS algorithms.

Stochastic local search (SLS) algorithms are the method of choice for solving computationally hard decision and optimization problems from a wide range of areas, including computing science, operations research, engineering, chemistry, biology and physics. SLS comprises a spectrum of techniques ranging from simple constructive and iterative improvement procedures to more complex methods, such as simulated annealing (SA), iterated local search or evolutionary algorithms (EAs). As evident from the term *stochastic* local search, randomization can, and often does, play a prominent role in these methods. Randomized choices may be used in the generation of initial solutions or in the decision which of several possible search steps to perform next – sometimes merely to break ties between equivalent alternatives, and sometimes to heuristically and probabilistically select from large and diverse sets of possible candidates. Judicious use of randomization can arguably simplify algorithm design and help achieve robust algorithm behavior.

The concept of an *SLS algorithm* has been defined formally [54.1] and not only provides a unifying framework for many different types of algorithms, including the previously mentioned constructive and iterative improvement procedures, but also provides a wide range of more complex search methods commonly known as metaheuristics.

Greedy constructive and iterative improvement procedures are important SLS algorithms, since they typically serve as building blocks for more complex SLS algorithms, whose performance critically depends on the design choices and fine tuning of these underlying components. Greedy constructive algorithms and iterative improvement procedures terminate naturally when a complete solution has been generated or a local optimum of a given evaluation function is reached, respectively. One possible way to obtain better solutions is to restart these basic SLS procedures from randomly chosen initial search positions. However, this approach has shown to be relatively ineffective in practice for reasonably sized problem instances (and it breaks down for large instances [54.2]).

To overcome these limitations, over the last decades, a large number of more sophisticated, general-purpose SLS methods [54.1] have been introduced; these are often called metaheuristics [54.3], since they are based on higher level schemes for controlling one or more subsidiary heuristic search procedures. We divide these general-purpose SLS methods into three broad classes: simple, hybrid and population-based SLS methods. Simple SLS methods typically use one neighborhood relation during the search and either modify the acceptance criterion for search steps, allowing to occasionally accept worsening steps, or modify the evaluation function that is used during the local search process. Examples of simple SLS methods include SA [54.4, 5] and (simple) tabu search [54.6–9]. A number of SLS methods combine different types of search steps – for example, construction steps and perturbative local search steps – or introduce occasional larger modifications into current candidate solutions, to provide appropriate starting points for subsequent iterative improvement search. Examples of such hybrid SLS methods include greedy randomized adaptive search procedures (GRASPs) [54.10] and iterated local search [54.11]. Finally, several SLS methods maintain and manipulate at each iteration a set, or population, of candidate solutions, which provides a natural way of increasing search diversification. Examples of such population-based SLS methods include EAs [54.12–15], scatter search [54.16, 17] and ant colony optimization [54.18–20].

Our classification into simple, hybrid and population-based SLS methods is not the only possible one, and certain SLS algorithms could be seen as belonging to more than one category. For example, many population-based SLS methods are also hybrid, as they use different search operators or combine the manipulation of the population of candidate solutions with iterative improvement on members of the population to achieve increased performance. In fact, there is an increasing trend to design and apply SLS algorithms that are not merely based on a single, well-established general-purpose SLS method, but rather combine flexibly elements of different SLS methods or incorporate mechanisms taken from systematic search algorithms, such as branch and bound or dynamic programming. The conceptual framework of SLS naturally accommodates this development, and the composition of more complex SLS algorithms from conceptually simpler components is explicitly supported, for example, by the concept of generalized local search machines [54.1]. In this context, methodological issues concerning the engineering of SLS algorithms [54.21, 22] are increasingly gaining importance. Similarly, the exploitation of automatic algorithm configuration techniques and, more generally, the programming by optimization paradigm [54.23] enable the systematic development of high-performance SLS algorithms.

54.1 The Nature and Concept of SLS

Computational approaches for the solution of hard, combinatorial problems can all be viewed as performing some form of search. Essentially, search algorithms generate and evaluate candidate solutions for the problem instance at hand. For combinatorial decision problems, the evaluation of a candidate solution requires to check whether the candidate solution is a feasible solution satisfying all given constraints; for combinatorial optimization problems, it involves computing the value of the given objective function. For NP-complete decision problems and NP-equivalent optimization problems, even the most efficient algorithms known to date require running time exponential in the instance size in the worst case, while candidate solutions can be evaluated in polynomial time.

A candidate solution for an instance of a combinatorial problem is generally composed of *solution components*. Consider, for example, the well-known traveling salesperson problem (TSP). In the TSP, one is given a weighted, fully connected graph $G = (V, E, w)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of $|V| = n$ vertices, $E \subset V \times V$ is the set of edges that fully connects the graph, and $w : E \mapsto \mathbb{R}_0^+$ is a function that assigns to each edge $e \in E$ a nonnegative weight $w(e)$. The objective is to find a minimum-weight Hamiltonian cycle in G. A candidate solution for a TSP instance can be represented by a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ of the vertex indices, and the objective function w is given as

$$w(\pi) := w(v_{\pi(n)}, v_{\pi(1)}) + \sum_{i=1}^{n-1} w(v_{\pi(i)}, v_{\pi(i+1)}) . \quad (54.1)$$

In the TSP, a (complete) candidate solution, commonly also called a tour, can be seen as consisting of n out of the $n \cdot (n-1)$ possible edges, and each edge represents a solution component.

Any given tour can be modified by removing two edges and introducing two unique new edges such that another valid tour is obtained. This modification is an example of a *perturbation* of a complete candidate solution, and we refer to search algorithms that make systematic use of such solution modifications as *perturbative search methods*. In practice, such perturbative search methods iteratively modify a current candidate solution according to some rule, and this process ends when a given termination criterion is met.

Perturbative search methods start from some complete candidate solution. The task of generating such candidate solutions is commonly accomplished by *constructive search methods* or *construction heuristics*. Constructive search methods iteratively extend an initially *empty* candidate solution by one or several solution components until a complete candidate solution is obtained. Constructive search methods can thus be seen as operating in a search space of *partial candidate solutions*. An example of a constructive search method is the nearest neighbor heuristic for the TSP. An initial vertex is chosen randomly, and at each construction step, the nearest neighbor heuristic follows a minimal weight edge to one of the vertices that have not yet been visited. These steps are iterated until all vertices have been visited, and the tour is completed by returning to the initial vertex.

Generally speaking, local search algorithms start at some initial search position and iteratively move, based on local information, from the current position to neighboring positions in the search space. Both perturbative and constructive search methods match this general description. While in the literature, the term local search is mostly used for perturbative search methods, it also applies to constructive search methods: A partial solution corresponds to a position in the search space of partial candidate solutions, and the neighbors of a partial solutions are obtained by extending it with one or more solution components. In fact, there are a number of well-known generic SLS methods, such as GRASP, iterated greedy and ant colony optimization, that are based on constructive local search.

Many local search algorithms use randomized decisions, for example, for generating initial solutions or when determining search steps. We therefore refer to such methods as stochastic local search (SLS) *algorithms*. The following components need to be specified to define an SLS algorithm (for a formal definition, we refer to Chap. 1 of [54.1]).

- *Search space* – comprises the set of *candidate solutions* (or *search positions*) for the given problem instance.
- *Solution set* – consists of the search positions that are considered to be solutions of the given problem instance. In the case of decision problems, the solution set comprises all feasible candidate solutions; in the case of optimization problems, the solution set typically comprises all optimal feasible candidate solutions.
- *Neighborhood relation* – specifies the direct neighbors of each candidate solution s, i. e., the search positions that can be reached from s in a single search step of the SLS algorithm.
- *Memory states* – hold additional information about the search beyond the search position. If an algorithm is memoryless, the memory may consist of a single, constant state.
- *Initialization function* – specifies the search initialization in the form of a probability distribution over initial search positions and memory states.
- *Step function* – determines the computation of search steps by mapping each search position and memory state to a probability distribution over its neighboring search positions and memory states.
- *Termination predicate* – used to decide search termination based on the current search position and memory state.

The formal definition of an SLS algorithm specifies the initialization function, the step function, and the termination predicate as probability distributions, which the algorithm samples at each step during any given run. In practice, however, the initialization function, the step function, and the termination predicate will be specified by procedures, and the corresponding probability distributions are only implicitly defined. Note that the definition of an SLS algorithm is general enough to include deterministic local search algorithms. In fact, formally we can describe deterministic local search algorithms as special cases of SLS algorithms – deterministic decisions can be modeled using degenerate probability distributions (Dirac delta).

The working principle of an SLS algorithm is then as follows. The search process starts from some initial search state that is generated by the initialization function. While some termination criterion is not satisfied, search steps are performed according to the step function. In the case of optimization problems, the SLS algorithm keeps track of the best solution found so far, which is then returned upon termination of the algorithm. In the case of decision problems, the SLS algorithm typically stops as soon as a (feasible) solution is found or another termination criterion is satisfied.

In all but the simplest cases, the search process is guided by an *evaluation function*, which measures the quality of candidate solutions. The efficacy of this guidance depends on the properties of the evaluation function and the way in which it is integrated into the search process. Evaluation functions are generally problem specific. For many optimization problems, the objective function given by the problem definition is used; however, different evaluation functions can sometimes provide better guidance, for example, in the sense of approximation guarantees [54.24]. In decision problems, an appropriate evaluation function has to be defined by the algorithm designer. Often, the objective function used for optimization variants of the decision problem can provide useful guidance. For example, for the satisfiability problem in propositional logic (SAT), the objective function of MAX-SAT, which, in a nutshell, counts the number of constraint violations, provides effective guidance. Some SLS methods, such as dynamic local search (briefly discussed in Sect. 54.3), modify the evaluation function during the search process.

The general concept of SLS algorithms, as introduced above and discussed in depth by *Hoos* and *Stützle* [54.1], provides a unified view of constructive and perturbative local search techniques that range from rather simplistic greedy constructive heuristics and iterative improvement algorithms to rather complex hybrid and population-based SLS methods. Population-based algorithms, which manipulate sets of candidate solutions at each iteration, fall under the definition of an SLS algorithm by considering search positions consisting of sets of candidate solutions. In this case, the step function also operates on sets of candidate solutions for the given problem instance. For example, in the case of typical EAs, recombination, mutation, and selection can all be modeled as operations on sets of candidate solutions, which are formally parts of a single-step function used for mapping one generation to the next.

It is instructive to contrast the concept of an SLS algorithm with that of a metaheuristic. Metaheuristics have been described as heuristics that are *superimposed on another heuristic* [54.6], a [54.25]:

master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality,

as [54.20]:

a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems,

and as [54.26]:

a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms.

However, the term metaheuristic [54.26]:

is also used to refer to a problem-specific implementation of a heuristic optimization algorithm according to the guidelines expressed in such a framework.

As is evident from these characterizations, there is no formal definition of the term metaheuristic, and its precise meaning has evolved over time. The term metaheuristic is commonly used to refer to the high-level guidance strategies that in many occasions are used to extend underlying greedy constructive or perturbative search procedures. Hence, the scope of the term metaheuristic differs from that of an SLS algorithm; it comprises what can be similarly loosely characterized as general-purpose SLS methods, but extends naturally to higher-level search strategies involving paradigms other than SLS, such as systematic search methods based on backtracking.

Conversely, the term metaheuristic is usually not applied to simple SLS procedures (such as random sampling, random walk and iterative improvement), nor to problem-specific SLS algorithms with provable properties. Therefore, there are SLS algorithms based on metaheuristics (such as ant colony optimization, iterated local search or EAs for various problems), SLS algorithms that are not metaheuristics (such as 2-opt for the TSP or conflict-directed random walk for SAT) and metaheuristics that are not based on SLS (such as various branch and bound methods and hybrids between systematic and local search).

Because the notion of an SLS algorithm explicitly refers to aspects that are not related to the high-level guidance of the search process, such as the choice of a neighborhood relation, evaluation function and termination predicate, research on SLS also covers the design, implementation and analysis of these more problem-specific components.

54.2 Greedy Construction Heuristics and Iterative Improvement

The main SLS techniques underlying more complex SLS methods (or metaheuristics) comprise (greedy) constructive search and iterative improvement algorithms. In the following, we discuss the main principles and choices underlying these methods.

Constructive search procedures (or construction heuristics) typically evaluate at each construction step the quality of the available solution components based on a heuristic function. *Greedy construction heuristics* choose to add at each step a solution component with best heuristic value, breaking ties either randomly or by means of a secondary heuristic function. For several polynomially solvable problems, such as the minimum spanning tree problem, greedy construction heuristics (for example, Kruskal's algorithm) are guaranteed to produce optimal solutions [54.27]; unfortunately, for NP-hard problems, this is generally not the case, due to the myopic decisions taken during solution construction.

A useful distinction can be made between static and adaptive construction heuristics. In *static* construction heuristics, the heuristic values associated with solution components are precomputed before the actual construction process is executed and remain unchanged throughout. In *adaptive* construction heuristics, the heuristic values are recomputed at each construction step to take into account the impact of the current partial solution. Adaptive construction heuristics tend to be more accurate and result in better quality candidate solutions than static heuristics, but they are also computationally more expensive.

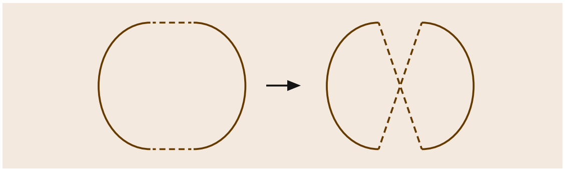
Fig. 54.1 A 2-exchange move for the symmetric TSP. Note that the pair of edges to be introduced is uniquely determined to ensure that the neighbor is again a tour

Construction heuristics are often used to provide good initial candidate solutions for perturbative local search algorithms. One of the most basic SLS methods is to iteratively improve a candidate solution for a given problem instance. Such an *iterative improvement* algorithm starts from some initial search position and iteratively replaces the current candidate solution s by an improving neighboring candidate solution s'. The local search is terminated once no improving neighbor is available, that is, $\forall s' \in N(s) : g(s) \leq g(s')$, where $g(\cdot)$ is the evaluation function to be minimized, and $N(s)$ denotes the set of neighbors of s. In the literature, iterated improvement algorithms are also referred to as *iterated descent* or (in the case of maximization problems) *hill-climbing procedures*.

Neighborhoods are problem specific, and it is generally difficult to predict a priori which of several possible neighborhoods results in best performance. However, for most problems, standard neighborhoods exist. Under the k-exchange neighborhood, two candidate solutions are neighbors if they differ by at most k solution components. An example is the 2-exchange neighborhood for the TSP, where two tours are neighbors if they differ by a pair of edges. Figure 54.1 illustrates a move in this neighborhood. In a k-exchange neighborhood, each candidate solution has $\mathcal{O}(n^k)$ direct neighbors, where n is the number of solution components in each candidate solution. Thus, the neighborhood size is exponential in k, as is the time to identify improving neighbors. While using larger neighborhoods typically makes it possible to reach better solutions, finding those solutions also takes more time. In other words, there is a tradeoff between the quality of the local optima reachable by an iterative improvement algorithm and its run time. In practice, neighborhoods that involve a quadratic or cubic time-complexity may already result in prohibitive computation times for large problem instances.

The overall time-complexity of searching a given neighborhood is determined by its size and the cost of evaluating each neighbor. The power of local search crucially relies on the fact that *caching and incremental updating techniques* can significantly reduce the cost of evaluating neighbors compared to computing the respective evaluation function values from scratch. For example, the quality of a 2-exchange neighbor of a tour for a TSP instance with n vertices can be computed from the quality of the current tour by subtracting and adding two edge weights (that is, two numbers) each; computing the weight of such a tour from scratch, on the other hand, requires n additions. Sometimes, to render the computation of the incremental updates as efficient as possible, additional data structures need to be implemented, but the net effect is often a very large reduction in computational effort.

A second important technique for reducing the time-complexity of evaluating a given neighborhood is based on the idea of excluding from consideration neighbors that are unlikely or provably unable to lead to improvements. These *neighborhoods pruning techniques* play a crucial role in many high-performance SLS algorithms. Examples of such pruning techniques are the fixed radius searches and nearest neighbors lists used for the TSP [54.28–30], the use of so-called *don't look bits* [54.28], as well as reduced neighborhoods for the job-shop scheduling problem [54.31] and pre-tests for search steps, as done for the single machine total weighted tardiness problem [54.32].

The speed and performance of iterative improvement algorithms also depends on the mechanism used to determine search steps, the so-called *pivoting rule* [54.33]. *Iterative best improvement* chooses at each step a neighboring candidate solution that mostly improves the evaluation function value. Any ties that occur can be broken either randomly, according to the order in which the neighborhood is searched, or based on a secondary criterion (as in [54.34]). In order to find a most improving neighbors, iterative best improvement needs to examine the entire neighborhood in each step. *Iterative first improvement*, in contrast, examines the neighborhood in some given order and moves to the first improving neighboring candidate solution found during this neighborhood scan. Iterative first improvement applies improving search steps earlier than iterative best improvement, but the amount of improvement achieved in each step tends to be smaller; therefore, it usually requires more improvement steps to reach a local optimum. If a candidate solution is a local optimum, first- and best-improvement algorithms detect this only by inspecting the entire neighborhoods of that solution; don't look bits [54.28, 29] offer a particularly useful mechanism for reducing the time required by this final check, the so-called check-out time.

Interestingly, the local optimum found by iterative first improvement depends on the order in which the neighborhood is examined. This property can be exploited by using a random order for scanning the neighborhood, and repeated runs of random-order first improvement algorithms can identify very different local optima, even if each run is started from the same initial position [54.1, Sect. 2.1]. Thus, the search process in random-order first improvement is more diversified than in the first improvement algorithms that scan local neighborhoods in fixed order.

The notion of local optimality is defined with respect to a specific neighborhood. Thus, changing the neighborhood during the local search process may provide an effective means for escaping from poor quality local optima, and offers the opportunity to benefit from the advantages of large neighborhoods without incurring the computational burden associated with using them exclusively. In the context of iterative improvement algorithms, this idea forms the basis of *variable neighborhood descent* (VND), a variant of a general-purpose SLS method known as variable neighborhood search (VNS) [54.35, 36]. VND uses a sequence of neighborhoods $N_1, N_2, \ldots, N_k$; this sequence is typically ordered according to increasing neighborhood size or increasing time complexity of searching the neighborhoods. VND starts by using the first neighborhood, N_1, until a local optimum is reached. Every time the exploration of a neighborhood N_i does not identify an improving local search step, that is, a local optimum w.r.t. neighborhood N_i is found, VND switches to the next neighborhood, N_{i+1} in the given sequence. Whenever an improving move has been made in a neighborhood N_i, VND switches back to N_1 and continues using the subsequent neighborhoods, N_2 etc., from there. The search is terminated when a local optimum w.r.t. N_k has been reached. The central idea of this scheme is to use small neighborhoods whenever possible, since they allow for the most efficient local search process. The VND scheme typically results in a significant reduction of computation time when compared to an iterative improvement algorithm that uses the largest neighborhood only. VND typically finds high-quality local optima, because upon termination, the resulting candidate solution is locally optimal with respect to all k neighborhoods examined.

Finally, recent years have seen an explosion in the development of iterative improvement methods that exploit *very large scale neighborhood*, whose size is typically exponential in the size of the given problem instance [54.37]. In fact, there are two main approaches to searching these neighborhoods. The first is to perform a heuristic search in the neighborhood, since a exact search would be computationally too demanding. This idea forms the basis of variable-depth search algorithms, where the number of solution components that are modified in each step is not determined a priori. Interestingly, the two best-known variable-depth search algorithms, the *Kernighan–Lin* algorithm for graph partitioning [54.38] and the *Lin–Kernighan* algorithm for the TSP [54.39], have been devised about in the early 1970s, a fact that illustrates the lasting interest in these types of methods. The more recent concept of ejection chains [54.40] is related to variable-depth search. Another interesting approach is to devise neighborhoods with a special structure that allows them to be searched either in polynomial time or at least very efficiently in practice [54.37, 41–43]. This is the central idea behind many recent developments in very large scale neighborhoods, which include techniques such as Dynasearch [54.32, 44] and cyclic exchange neighborhoods [54.45, 46]. As a result of these research efforts, current state-of-the-art methods for a variety of combinatorial problems such as the TSP [54.47] or the single machine total weighted tardiness problem [54.48] rely on iterative improvement algorithms based on very large scale neighborhoods.

54.3 *Simple* SLS Methods

Iterative improvement algorithms accept only improving neighbors as new current candidate solutions, and they terminate when encountering a local optimum. To allow the search process to progress beyond local optima, many SLS methods permit moves to worsening neighbors. We refer to the methods discussed in the following as *simple* SLS methods, because they essentially only use one type of search steps, in a single, fixed neighborhood relation.

54.3.1 Randomized Iterative Improvement

The key idea behind randomized iterative improvement (RII) is to occasionally perform moves to random neighboring candidate solutions irrespective of their evaluation function value. The simplest way of implementing this idea is to apply, with a given probability w_p, a so-called *uninformed random walk* step, which chooses a neighbor of the current candidate solution uniformly at random, while with probability $1 - w_p$, an improvement step is performed. Often, the improvement step will correspond to one iteration of a best improvement procedure. The parameter w_p is referred to as *walk probability* or, simply, *noise parameter*. RII algorithms have the property that they can perform arbitrarily long sequences of random walk steps; the length of these sequences (i. e., the number of consecutive random walk steps) follows a geometric distribution with parameter w_p. This allows effective escapes from local optima and renders RII probabilistically approximately complete [54.1, Sect. 4.1]. A main advantage of RII is ease of implementation – often, only a few additional lines of code are required to extend an iterative improvement procedure to an RII procedure – and its behavior is effectively controlled by a single parameter.

RII algorithms have been shown to perform quite well in a number of applications. For example, in the 1990s, minor variations of RII, in which random walk steps are determined based on the status of constraint violations rather than chosen uniformly at random, have been *state of the art* for solving the SAT [54.49, 50] and other constraint satisfaction problems [54.51]. Due to their simplicity, RII algorithms also facilitate theoretical analyses, including characterization of performance in dependence of parameter settings [54.52].

54.3.2 Probabilistic Iterative Improvement

Instead of accepting worsening search steps regardless of the amount of deterioration in evaluation function value they caused (as is the case for random walk steps), it may be preferable to have the probability of acceptance depend on the change of the evaluation function value incurred. This is the key idea underlying probabilistic iterative improvement (PII). Unlike RII, each step of PII involves two phases: first, a neighboring candidate solution $s' \in N(s)$ is selected uniformly at random (proposal mechanism); then, a probabilistic decision is made whether to accept s' as the new search position (acceptance test). For minimization problems, the acceptance probability is often based on the *Metropolis condition* and defined as

$$p_{\text{accept}}(T, s, s') := \begin{cases} 1 & \text{if } g(s') < g(s) \\ \exp\left(\dfrac{g(s) - g(s')}{T}\right) & \text{otherwise}, \end{cases} \tag{54.2}$$

where $p_{\text{accept}}(T, s, s')$ is the acceptance probability, g is the evaluation function to be minimized, and T is a parameter that influences the probability of accepting a worsening search step. PII is closely related to simulated annealing (SA), discussed next; in fact, when using the acceptance mechanism given above, PII is equivalent to constant-temperature SA. In light of this connection, parameter T is also called *temperature*. For various applications, such PII procedures have been shown to perform quite well, provided that T is chosen carefully [54.53, 54]. It is worth noting that in the limit for $T = 0$, PII effectively turns into an iterative improvement procedure (i. e., never accepts worsening steps), while for $T = \infty$, it performs a uniform random walk.

54.3.3 Simulated Annealing

Simulated annealing (SA) [54.4, 5] is similar to PII, except that the parameter T is modified at run time. Following the analogy of the physical annealing of solid materials (e.g., metals and glass), which inspired SA, the temperature T is initially set to some high value and then gradually decreased. At the beginning of the search process, high temperature values result in relatively high probabilities of accepting worsening candidate solutions. As the temperature is decreased, the search process becomes increasingly greedy; for very low settings of the temperature, almost only improving neighbors or neighbors with evaluation function value equal to the current candidate solution are accepted.

Standard SA algorithms iterate over the same two stage process as PII, typically using uniform sampling (with or without replacement) from the neighborhood as a proposal mechanism and a parameterized acceptance test based on the Metropolis condition (54.2) [54.4, 5]. The modification of temperature T is managed by a so-called *annealing (or cooling) schedule*, which is a function that determines the temperature value at each search step. One of the most common choices is a geometric cooling schedule, defined by an initial temperature, T_0, a parameter α between 0 and 1, and a value k, called the temperature length, which defines the number of candidate solutions that are proposed at each fixed value of the temperature; every k steps, the temperature is updated as $T := \alpha \cdot T$. Important parameters of SA are often determined based on characteristics of the problem instance to be solved. For example, the initial temperature may be based on statistics derived from an initial, short random walk, the temperature length may be set to a multiple of the neighborhood size, and the search process may be terminated when the frequency with which proposed search steps are accepted falls below a given threshold.

SA is one of the oldest and most studied SLS methods. It has been applied to a very broad range of computational problems, and many types of annealing schedules, proposal mechanisms, and acceptance tests have been investigated. SA has also been subject to a substantial amount of theoretical analysis, which has yielded various convergence results. For more details on SA, we refer to [54.55, 56].

54.3.4 Tabu Search

Tabu search (TS) differs significantly from the previously discussed SLS methods, in that it makes a direct and systematic use of memory to direct the search process [54.25]. In its most basic form, which is also called *simple tabu search*, TS expands an iterative improvement procedure with a short-term memory to prevent the local search process from returning to recently visited search positions. Instead of memorizing complete candidate solutions and forbidding these explicitly, TS usually associates a tabu status with specific solution components. In the latter case, TS stores for each solution component the time (i.e., the iteration number) at which it was last modified. Each solution component is then considered as potentially tabu if the difference between the stored iteration number and the current iteration number is larger than the value of a parameter called *tabu tenure* (or tabu list length). The tabu status of a local search step is then determined based on specific tabu criteria, which are a function of the tabu status of solution components that are affected by it. One effect is that once a search step has been performed, it is tabu in that it cannot be reversed for a certain number of iterations.

Seen from a neighborhood perspective, TS dynamically restricts the set of neighbors permissible at each local search step by excluding neighbors that are currently tabu. Since the tabu mechanism through prohibition of solution components is quite restrictive, many simple TS algorithms use an *aspiration criterion*, which overrides the tabu status of neighbors if specific conditions are satisfied; for example, if a local search step leads to a new best solution, aspiration allows it to be accepted regardless of its tabu status.

As an example, consider a simple TS algorithm for the TSP, based on the 2-exchange neighborhood. Edges that are removed (or introduced) by a 2-exchange step may then not be reintroduced into (or removed from) the current tour for tt search steps, where tt is the tabu tenure.

For several problems, even simple TS algorithms have been shown to perform quite well. However, the performance of TS strongly depends on the tabu tenure setting. To avoid the difficulty of finding fixed settings suitable for a given problem, mechanisms such as reactive tabu search [54.57] have been devised to adapt the tabu tenure at run time. Simple TS algorithms can be improved in many different ways. In particular, various mechanisms have been developed that make use of intermediate-term and long-term memory to further enhance the performance of simple TS. For a detailed description of such techniques, which aim either at intensifying the search in specific areas of the search space or at diversifying the search to explore unvisited search space regions, we refer to the book by *Glover* and *Laguna* [54.25].

54.3.5 Dynamic Local Search

In contrast to the *simple* SLS methods discussed so far, dynamic local search (DLS) does not accept worsening search steps, but rather modifies the evaluation function during the search in order to escape from local optima. These modifications of the evaluation function g are commonly triggered whenever the underlying local search algorithm, typically an iterative improvement procedure, has reached a locally optimal solution with respect to g', the current evaluation function. Next, the evaluation function is modified and the subsidiary local search algorithm is run until a local optimum (with respect to the new g') is encountered. These local search phases and evaluation function updates are iterated until some termination criterion is met (see Algorithm 54.1).

Algorithm 54.1 High-level outline of dynamic local search

```
**Dynamic local search (DLS):**
determine initial candidate solution $s$
initialize penalties
**while** termination criterion is not satisfied **do**
  compute modified evaluation function $g'$
      from $g$ and penalties
  perform subsidiary local search on $s$ using $g'$
  update penalties based on $s$
**end while**
```

The modified evaluation function g' is typically computed as the sum of the original evaluation function and penalties associated with each solution component, that is

$$g'(s) := g(s) + \sum_{i \in SC(s)} \text{penalty}(i) \,, \tag{54.3}$$

where g is the original evaluation function, $SC(s)$ is the set of solution components of candidate solution s, and penalty(i) is the penalty of solution component i. Initially, all penalties are set to zero. Variants of DLS differ in the details of their penalty update mechanism (e.g., additive vs. multiplicative updates, occasional reduction of penalties) and the choice of the solution components whose penalties are adjusted. For example, *guided local search* [54.58, 59] uses the following mechanism for choosing the solution components whose penalties are increased: First, a utility value $u(i) := g_i(s)/(1 + \text{penalty}(i))$ is computed for each solution component i, where $g_i(s)$ measures the impact of i on the evaluation function; then, the penalties of solution components with maximal utility are increased.

DLS algorithms are sometimes referred to as a soft form of tabu search, since solution components are not strictly forbidden, but the effect of the penalties resembles a soft prohibition. There are also conceptual links to Lagrangian methods [54.60, 61]. DLS algorithms have been shown to reach state-of-the-art performance for SAT [54.62] and for the maximum clique problem [54.63].

54.4 Hybrid SLS Methods

The performance of basic SLS techniques can often be improved by combining them with each other. In fact, even RII can be seen as a combination of iterative improvement and random walk, using the same neighborhood. Several other SLS methods combine different types of search steps, and in the following, we briefly discuss some prominent examples.

54.4.1 Greedy Randomized Adaptive Search Procedures

As mentioned previously, construction heuristic can be easily and effectively combined with perturbative local search procedures. While greedy construction heuristics generally generate only one or very few different candidate solutions, randomization of the constriction process makes it possible to generate many different high-quality solutions. The idea underlying GRASP [54.10, 64] is to combine randomized greedy construction with a subsequent perturbative local search phase, whose goal is to improve the candidate solutions produced by the construction heuristic. The two phases of solution construction and perturbative local search are repeated until a termination criterion, e.g., maximum computation time, is met. The term *adaptive* in GRASP refers to the fact that the hybrid search process typically uses an adaptive construction heuristic. Randomization in GRASP is realized based on the concept of a *restricted candidate list*, which contains the best-scoring solution components according to the given heuristic function. In the simplest and most common GRASP variants, elements are chosen uniformly at random from this restricted candidate list during the construction process. For a detailed description, various extensions, and an overview of applications of GRASP, we refer to [54.64].

54.4.2 Iterated Greedy Algorithms

A disadvantage of GRASP is that new candidate solutions are constructed from scratch and independently of previously found solutions. Iterated greedy (IG) algorithms iteratively apply greedy construction heuristics to generate a chain of high-quality candidate solutions. The central idea is to alternate between solution construction and destruction phases, and thus to combine at least two different types of search steps. IG algorithms first build an initial, complete candidate solution s. Then, they iterate over the following phases, until a termination criterion is met:

1. Starting from the current candidate solution, s, a destruction phase is executed, during which some solution components are removed from s, resulting in a partial candidate solution s'. The solution components that are removed in this phase may be chosen at random or, for example, based on their impact on the evaluation function.
2. Starting from s', a construction heuristic is used to generate another candidate solution, s''. This construction heuristic may differ from the one used to generate the initial candidate solution.
3. Based on an acceptance criterion, a decision is made whether to continue the search from s or s''. Additionally, it is often useful to further improve complete candidate solutions by means of a subsidiary perturbative local search procedure (see Algorithm 54.2 for a high-level outline of IG).

Algorithm 54.2 High-level outline of an iterated greedy (IG) algorithm

```
Iterated greedy (IG):
construct initial candidate solution $s$
perform subsidiary local search on $s$
while termination criterion is not satisfied do
    apply destruction to $s$, resulting in $s'$
    apply constructive heuristic starting from $s'$,
    resulting in $s''$
    perform subsidiary local search on $s''$ (optional)
    based on acceptance criterion, keep $s$ or
      accept $s := s''$
end while
```

The principle underlying IG methods has been rediscovered several times, and consequently, can be found under various names, including *ruin-and-recreate* [54.65], *iterative flattening* [54.66], and *iterative construction heuristic* [54.67]; it has also been used in the context of SA [54.68]. IG algorithms, especially when combined with perturbative local search methods, have reached state-of-the-art performance for a number of problems, including several variants of flowshop scheduling [54.69, 70].

54.4.3 Iterated Local Search

Iterated local search (ILS) generates a sequence of solutions by alternating applications of a perturbation mechanism and of a subsidiary local search algorithm. Consequently, ILS can be seen as a hybrid between the search methods underlying the local search and perturbation phases.

An ILS algorithm is specified by four main components. The first is the mechanism used for generating an initial solution, for example, a greedy constructive heuristic. The second is a subsidiary (perturbative) local search procedure; typically, this is an iterative improvement algorithm, but often, other simple SLS methods are used. The third component is a perturbation procedure that introduces a modification to a given candidate solution. These perturbations should be complementary to the modifications introduced by the subsidiary local search procedure; in particular, the effect of the perturbation procedure should not be easily reversible by the local search procedure. The fourth component is an acceptance criterion, which is used to decide whether to accept the outcome of the latest perturbation and local search phase.

ILS starts by generating an initial candidate solution, to which then subsidiary local search is applied. It then iterates over the following phases, until a termination criterion is met:

1. Perturbation is applied to the current candidate solution s, to obtain an intermediate candidate solution s'.
2. Subsidiary local search is applied to s'.
3. Based on the acceptance criterion, a decision is made whether to continue the search from s or s' (see Algorithm 54.3 for a high-level outline of ILS).

Often, the subsidiary search is based on iterative improvement and ends in a local optimum; ILS can therefore be seen as performing a biased random walk in the space of local optima produced by the given subsidiary local search procedure. The acceptance criterion (together with the strength of the perturbation mechanism) then determines the degree of search intensification: if only improving candidate solutions are accepted, ILS performs a randomized first-improvement search in the space of local optima; if any new local optimum is accepted, ILS performs a random walk in the space of local optima.

Algorithm 54.3 High-level outline of iterated local search

```
**Iterated local search (ILS):**
generate initial candidate solution $s$
perform subsidiary local search on $s$
**while** termination criterion is not satisfied **do**
    apply perturbation to $s$, resulting in $s'$
    perform subsidiary local search on $s'$
    based on acceptance criterion, keep $s$
      or accept $s := s'$
**end while**
```

An attractive feature of ILS is that basic versions can be quickly and easily implemented, especially if a simple SLS algorithm or an iterative improvement procedure is already available. Using some additional refinements, ILS methods define the current state of the art for solving many combinatorial problems, including the TSP [54.71]. Similar to IG, ILS is based on an idea that has been rediscovered several times and is known under various names, including *large-step Markov chains* [54.29] and *chained local optimization* [54.72]. There is also a close conceptual connection with several variants of variable neighborhood search (VNS) [54.35]; in fact, the so-called basic VNS and skewed VNS algorithms can be seen as variants of ILS that adapt the perturbation strength at run time. For more details on iterated local search, we refer to [54.73].

54.5 Population-Based SLS Methods

The use of a population of candidate solutions offers a convenient way to increase diversification in SLS. For example, population-based extensions of ILS algorithms have been proposed with this aim in mind [54.74, 75]. A further potential benefit comes from the inherent parallelizability of the most population-based SLS methods, although the parallelization thus achieved is not necessarily more effective than the simple and generic approach of performing multiple independent runs of an SLS algorithm in parallel (see also [54.1], Sect. 4.4). As previously remarked, population-based methods can be cast into the SLS framework described in Sect. 54.1 by defining search positions to consist of sets of candidate solutions and by using neighborhood relations, initialization, and step functions that operate on such populations.

Unfortunately, the benefits derived from the use of populations come at the cost of increased complexity, in terms of implementation effort, and parameters that need to be set appropriately. In what follows, we describe two of the most prominent population-based methods, one based on a constructive search paradigm (ant colony optimization), and the other based on a perturbative search paradigm (evolutionary algorithms).

54.5.1 Ant Colony Optimization

Ant colony optimization (ACO) algorithms have originally been inspired by the trail-following behavior of real ant species, which allows them to find shortest paths [54.76, 77]. This biological phenomenon gave rise to a surprisingly effective algorithm for combinatorial optimization [54.18, 19]. In ACO, the artificial ants perform a randomized constructive search that is biased by (artificial) pheromone trails and heuristic information derived from the given problem instance. The pheromone trails are numerical values associated with solution components that are adapted at run time to reflect experience gleaned from the search process so far.

During solution construction, at each step every ant chooses a solution component, probabilistically preferring those with high-pheromone trail and heuristic information values. For illustration, consider the TSP – the first problem to which ACO has been applied [54.18]. Each edge (i,j) has an associated pheromone value τ_{ij} and a heuristic value η_{ij}, which for the TSP is typically defined as $1/w(i,j)$, that is, the inverse of the edge weight. In ant system [54.19], the first ACO algorithm for the TSP, an ant located at vertex i would add vertex j to its current partial tour s' with probability

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{l \in N(i)} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}} , \tag{54.4}$$

where $N(i)$ is the feasible neighborhood of vertex i, i. e., the set of all vertices that have not yet been visited in s', and α and β are parameters that control the relative importance of pheromone trails and heuristic information, respectively. Note that the tour construction procedure implemented by the artificial ants is a randomized version of the nearest neighbor construction heuristic. In fact, randomizing a greedy construction heuristic based on pheromone trails associated with the decisions to be made would generally be a good initial step toward an effective ACO algorithm for a combinatorial problem.

Once every ant has constructed a complete candidate solution, it is typically highly advantageous to apply an iterative improvement procedure or a simple SLS algorithm [54.20, 78]. Next, the pheromone trail values are updated by means of two counteracting mechanisms. The first models pheromone evaporation and decreases some or all pheromone trail values by a constant factor. The second models pheromone deposit and increases the pheromone trail levels of solution components that have been used by one or more ants. The amount of pheromone deposited typically depends on the quality of the respective solutions. In the best performing ACO algorithms, only some of the ants with the highest quality solutions are allowed to deposit pheromone. The overall result of the pheromone update is an increased probability of choosing solution components in subsequent solution constructions that have previously been found to occur in high-quality solutions. ACO algorithms then cycle through these phases of solution construction, application of local search, and pheromone update until some termination criterion is met (see Algorithm 54.4 for a high-level outline of ACO).

Algorithm 54.4 High-level outline of ant colony optimization

```
Ant colony optimization (ACO):
initialize pheromone trails
while termination criterion is not satisfied do
    generate population sp of candidate solutions
        using subsidiary randomized
        constructive search
    perform subsidiary local search on sp
    update pheromone trails
end while
```

Many different variants of ACO algorithms have been studied. Along with many additional details on ACO, these are described in the book by *Dorigo* and *Stützle* [54.20]; for more recent surveys, we refer the reader to [54.79, 80]. The ACO metaheuristic [54.81, 82] provides a general framework for these variants and a generic view of how to apply ACO algorithms. ACO is also one of the most successful algorithmic techniques within the broader field of swarm intelligence [54.83].

54.5.2 Evolutionary Algorithms

Evolutionary algorithms (EAs) are a prominent class of population-based SLS methods that are loosely inspired by concepts from biological evolution. Unlike ACO algorithms, EAs work with a population of complete candidate solutions. The initial set of candidate solutions is typically created randomly, but greedy construction heuristics may also be used to seed the population. This population then undergoes an artificial evolution, where at each iteration, the population of candidate solutions is modified by means of mutation, recombination and selection.

Mutation operators typically introduce small, random perturbations into individual candidate solutions. The strength of these perturbations is usually controlled by a parameter called *mutation rate*; alternatively, a specific, fixed perturbation, akin to a random walk step in RII, may be performed. *Recombination operators* generate one or more new candidate solutions by combining information from two or more parent candidate solutions. The most common type of recombination is *crossover*, inspired by the homonymous mechanism in biological evolution; it generates offspring by assembling partial candidate solutions from linear representations of two parents. In addition to mutation and recombination, *selection mechanisms* are used to determine the candidate solutions that will undergo mutation and recombination, as well as those that will form the population used in the next iteration of the evolutionary process. Selection is based on the *fitness*, i. e., evaluation function values, of the candidate solutions, such that better candidate solutions have a higher probability to be selected.

Details of the mutation, recombination and selection mechanisms all have a strong impact on the performance of an EA. Generally, the use of problem specific knowledge within these mechanisms leads to better performance. In fact, much research in EAs has been devoted to the design of effective mutation and recombination operators; a good example for this is the TSP [54.84, 85]. To achieve cutting-edge performance in an EA, it is often useful to improve at least the best candidate solutions in a given population by means of a perturbative local search method, such as iterative improvement. The resulting class of hybrid algorithms, which are also known as *memetic algorithms* (MA) [54.86], are enjoying increasing popularity as a broadly applicable method for solving solving combinatorial problems (see Algorithm 54.5 for a high-level outline of an MA).

Algorithm 54.5 High-level outline of a memetic algorithm

```
**Memetic algorithm (MA):**
initialize population $p$
perform subsidiary local search on each
  candidate solution in $p$
**while** termination criterion is not satisfied **do**
    generate set $pr$ of candidate solutions
      through recombination
    perform subsidiary local search on each
      candidate solution of $pr$
    generate set $pm$ of candidate solutions
      from $p \cup pr$ through mutation
    perform subsidiary local search on each
      candidate solution of $pm$
    select new population $p$ from candidate
      solutions in $p \cup pr \cup pm$
**end while**
```

Several other techniques are conceptually related to evolutionary algorithms but have different roots. *Scatter search* and *path relinking* are SLS methods whose origins can be traced back to the mid-1970s [54.16]. Scatter search can be seen as a memetic algorithm that uses special types of recombination and selection mechanisms. Path relinking corresponds to a specific form of interpolation between two (or possibly more) candidate solutions and is thus conceptually related to recombination operators. Both methods have recently become increasingly popular; details can be found in [54.17, 87].

54.6 Recent Research Directions

In this section, we concisely discuss three research directions that we regard as particularly timely and promising: combinations of SLS and systematic search techniques, SLS algorithm engineering, and automated configuration and design of SLS algorithms. For other topics of interests, such as SLS algorithms for multiobjective [54.88–90], stochastic [54.91] or dynamic problems [54.92, 93], we refer to the literature for more details.

54.6.1 Combination of SLS Algorithms with Systematic Search Techniques

Systematic search and SLS are traditionally seen as two distinct approaches for solving challenging combinatorial problems. Interestingly, the particular advantages and disadvantages of each of these approaches render them rather complementary. Therefore, it is hardly surprising that over the last few years, there has been increased interest in the exploration and development of hybrid algorithms that combine ideas from both paradigms. For example, related to the area of mathematical programming, the term *Matheuristics* has recently been coined to refer to methods that combine elements from mathematical programming techniques (which are primarily based on systematic search) and (meta)heuristic search algorithms [54.94].

Hybrids between SLS and systematic search fall into two main classes. The first of these consists of approaches where the systematic search algorithm plays the role of the master process, and an SLS procedure is used to solve subproblems that arise during the systematic search process. Probably, the simplest, yet potentially effective method is to use an SLS algorithm to provide an initial high-quality (primal) bound on the optimal solution of the problem, which is then used by the systematic search algorithm for pruning parts of the search tree. Several more elaborate schemes have been devised, e.g., in the context of column generation and separation routines in integer programming [54.95]. Other approaches introduce the spirit of local search into integer programming solvers; examples of these include *local branching* [54.96] and *relaxation-induced neighborhood search* [54.97]. We refer to [54.95] for a recent overview of such combinations.

The second class of hybrid approaches is based on the idea of using systematic search procedures to deal with specific tasks arising while running an SLS algorithm. Very-large neighborhood search [54.37], as discussed in Sect. 54.2, is probably one of the best-known examples. Elements of tree search methods can also be exploited within constructive search algorithms, as exemplified by the use of branch and bound techniques in ACO algorithms [54.98, 99]. Other examples include *tour merging* [54.100] and the usage of information derived from integer programming formulations of optimization problems in heuristic methods [54.101]. We refer to [54.102] for a survey of this general approach. A taxonomy of the possible combinations of exact and local search algorithms has been introduced by *Jourdan* et al. [54.103].

Despite an increasing number of efforts on combining systematic search methods and SLS methods, as reviewed in [54.94], much work remains to be done in this direction, especially considering that the two underlying fundamental search paradigms are developed primarily in rather disjoint communities. We believe that much can be gained by overcoming the traditional view of these two approaches as being competing with each other in favour of focusing on synergies due to their complementarity.

54.6.2 SLS Algorithm Engineering

Despite the impressive successes in SLS research and applications – SLS algorithms are now firmly established as the method of choice for tackling a broad range of combinatorial problems – there are still significant shortcomings. Perhaps most prominently, there is a lack of guidelines and best practices regarding the design and development of effective SLS algorithms. Current practice is to implement one specific SLS method, based on one or more construction heuristics or iterative improvement procedures. However, general-purpose SLS methods are not fully defined recipes: they leave many design choices open, and typically only specific combinations of these choices will result in an effective algorithms for a given problem. Even worse, the underlying basic construction and iterative improvement procedures have a tremendous influence on the final performance of the SLS algorithms built on them, and this influence is frequently neglected.

We firmly believe that a more methodological approach needs to be taken toward the design and implementation of SLS algorithms. The research direction dedicated to developing such an approach is called *stochastic local search algorithm engineering* or, for short, *SLS engineering*; it is conceptually related to algorithm engineering [54.104] and software engineering [54.105], where similar methodological issues are tackled in a different context. Algorithm engineering is rather closely related to SLS engineering; it has been conceived as an extension to the traditionally more theoretically oriented research on algorithms. Algorithm engineering, according to [54.104], deals with the iterative process of designing, analyzing, implementing, tuning and experimentally evaluating algorithms. SLS engineering shares this motivation; however, the algorithms that are dealt with in the context of SLS approaches have substantially more complex and unpredictable behavior than those typically considered in algorithm engineering. There are several reasons for this: SLS algorithms are usually used for solving NP-hard problems, they allow for many more degrees of freedom in the choice of algorithm components, and their stochasticity makes analysis more complex.

From a high-level perspective, an initial approach to a successful *SLS engineering process* would proceed in a bottom-up fashion. Starting from knowledge about the problem, it would build SLS algorithms by iteratively adding complexity to simple, basic algorithms. More concretely, a tentative first attempt at such a process could be as follows:

1. Study existing knowledge on the problem to be solved and its characteristics;
2. Implement basic and advanced constructive and iterative improvement procedures;
3. Starting from these, add complexity (for example, by moving to simple SLS methods);
4. Improve performance by gradually adding concepts from more complex SLS techniques (for example, perturbations, prohibition mechanisms, populations);
5. Further configure and fine-tune parameters and design choices;
6. If found to be useful: iterate over steps 4–5.

Obviously, such a process would not necessarily strictly follow this outline, but insights gained at later stages could prompt revisiting earlier design decisions. Several high-performance SLS algorithms have already been developed following roughly the process outlined above (see [54.106] for an explicit example).

The SLS engineering process can be supported in various ways. Algorithm development, implementation and testing is facilitated by the use of programming frameworks like Paradiseo [54.107, 108] and EasyLocal++ [54.109, 110], dedicated languages and systems like COMET [54.111], libraries of data types (such as LEDA [54.112]), and statistical tools, such as the comprehensive, open-source R environment [54.113]. We expect that software environments specifically designed for the automated empirical analysis and design of algorithms, such as HAL [54.114, 115], will be especially useful in this context. Tools for the automatic configuration and tuning of algorithms, discussed further in the next section are also of considerable importance.

Furthermore, we see an improved understanding of the relationship between problem and instance features on the one side, and the properties and the behavior of SLS methods on the other side as key enabling factors for advanced SLS engineering approaches. The potential insights to be gained are not only of practical value to SLS engineering but also of considerable scientific interest. Progress in this direction is facilitated by advanced search space analysis techniques, statistical methods and machine learning approaches (see, e.g., *Merz* and *Freisleben* [54.116], *Xu* et al. [54.117] and *Watson* et al. [54.118]). Another promising avenue for future research involves the integration of theoretical insights into the design process, for example, by restricting design alternatives or parameter choices.

It is important to note that research toward SLS engineering adopts a *component-wise view of SLS methods*. For example, iterated local search (ILS) uses perturbations to diversify the search as well as acceptance tests (components: perturbations, acceptance tests), while evolutionary algorithms prominently involve the use of a population of solutions (component: population of solutions). Each of these components can be instantiated in different ways, and various combinations are possible. An effective SLS engineering process should provide guidance to the algorithm designer regarding the choice and configuration of these components. It would naturally and incrementally lead to combinations of algorithmic components taken from different SLS methods (or other paradigms, such as mathematical programming – [54.94]), if these contribute to desirable performance characteristics of the algorithm under design. Such an engineering process would therefore rather naturally produce hybrid algorithms that are effective for solving the given computational problem.

Finally, SLS engineering highlights more the importance of decisions concerning the underlying basic SLS techniques (such as construction heuristics, neighborhoods, efficient data structures, etc.) than the general-purpose SLS methods (or metaheuristics) used in a given algorithm design scenario. In fact, in our experience, such fundamental choices together with: (i) the level of expertise of the SLS algorithm developer and implementer, (ii) the time invested in designing and configuring the SLS algorithm, (iii) the creative use of insights into algorithm behavior and interaction with problem characteristics play a considerably more important role in the design of effective SLS algorithms than the focus on specific features prescribed by so-called metaheuristics.

54.6.3 Automatic Configuration of SLS Algorithms

The performance of algorithms for virtually any computationally challenging problem (and in particular, for any NP-hard problem) depends strongly on appropriate settings of algorithm parameters. In many cases, there are tens of such parameter; for example, the well-known commercial CPLEX solver for integer programming problems has more than 130 user-specifiable parameters that influence its search behavior. Likewise, the behavior of most SLS algorithms is controlled by parameters, and many design choices can be exposed in the form of parameters. This gives rise to algorithms with many categorical and numerical parameters. Categorical parameters are used to make choices from a discrete set of design variants, such as search strategies, neighborhoods or perturbation mechanisms. Numerical parameters often arise as subordinate parameters that directly control the behavior of a search strategy (e.g., temperature in SA and tabu tenure in simple tabu search). The goal in automated algorithm configuration is to find settings of these parameters that achieve optimized performance w.r.t. a performance metric of interest (for example, solution quality or computation time).

Automated algorithm configuration methods are an active area of research and have been demonstrated to achieve very substantial performance gains on many widely studied and challenging problems [54.119]. So-called *offline configuration methods*, which determine performance-optimizing parameter settings on a representative set of benchmark instances during a training phase before algorithm deployment, have arguably been studied most intensely been studied. These include procedures that are limited to tuning numerical parameters, such as CALIBRA [54.120], experimental design-based approaches [54.69, 121], SPO [54.122] and SPO$^+$ [54.123]. Methods that can handle categorical as well as numerical parameters are considerably more versatile; these include racing procedures [54.124, 125], model-free configuration procedures [54.126–128], and recent sequential model-based techniques [54.129, 130].

In *online configuration*, algorithm parameters are modified while attempting to solve a given problem instance. There are some inherent advantages of online configuration methods w.r.t. offline methods, especially when targeting very heterogeneous instances, where appropriate algorithm parameters may depend strongly on the problem instance to be solved. Some of these methods fall into the realm of reactive search methods [54.131]; others have been studied in the area of evolutionary computation (for an overview, we refer to [54.132]). Unfortunately, most online configuration methods presently available deal with very few parameters primarily responsible for algorithm performance (often only one) and rely on specific insight into the working principles of the given algorithm.

There are also various approaches for determining configurations of a given algorithm that result in good performance on a given problem instance. These *per-instance configuration methods* typically make use of computationally cheap instance features, which provide the basis for selecting the configuration to be used for solving a given instance [54.133–135].

Finally, we believe that there is significant promise in approaches for automating large parts of the design process of performance-optimized SLS algorithms as, for example, outlined in recent work on computer-aided algorithm design for generalized local search machines [54.136] and the *programming by optimization* (PbO) software design paradigm [54.23].

References

54.1 H.H. Hoos, T. Stützle: Stochastic Local Search—Foundations and Applications (Morgan Kaufmann, San Francisco 2004)

54.2 G.R. Schreiber, O.C. Martin: Cut size statistics of graph bisection heuristics, SIAM J. Optim. **10**(1), 231–251 (1999)

54.3 M. Gendreau, J.-Y. Potvin (Eds.): *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, Vol. 146 (Springer, New York 2010)

54.4 S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi: Optimization by simulated annealing, Science **220**, 671–680 (1983)

54.5 V. Cerný: A thermodynamical approach to the traveling salesman problem, J. Optim. Theory Appl. **45**(1), 41–51 (1985)

54.6 F. Glover: Future paths for integer programming and links to artificial intelligence, Comput. Oper. Res. **13**(5), 533–549 (1986)

54.7 F. Glover: Tabu search – Part I, ORSA J. Comput. **1**(3), 190–206 (1989)

54.8 F. Glover: Tabu search – Part II, ORSA J. Comput. **2**(1), 4–32 (1990)

54.9 P. Hansen, B. Jaumard: Algorithms for the maximum satisfiability problem, Computing **44**, 279–303 (1990)

54.10 T.A. Feo, M.G.C. Resende: A probabilistic heuristic for a computationally difficult set covering problem, Oper. Res. Lett. **8**(2), 67–71 (1989)

54.11 H.R. Lourenço, O. Martin, T. Stützle: Iterated local search. In: *Handbook of Metaheuristics*, ed. by F. Glover, G. Kochenberger (Kluwer, Norwell 2002) pp. 321–353

54.12 J.H. Holland: *Adaption in Natural and Artificial Systems* (The University of Michigan, Ann Arbor 1975)

54.13 D.E. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley, Reading 1989)

54.14 I. Rechenberg: *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Information* (Fromman, Freiburg, Germany 1973)

54.15 H.-P. Schwefel: *Numerical Optimization of Computer Models* (Wiley, Chichester 1981)

54.16 F. Glover: Heuristics for integer programming using surrogate constraints, Decis. Sci. **8**, 156–164 (1977)

54.17 F. Glover, M. Laguna, R. Martí: Scatter search and path relinking: Advances and applications. In: *Handbook of Metaheuristics*, ed. by F. Glover, G. Kochenberger (Kluwer, Norwell 2002) pp. 1–35

54.18 M. Dorigo, V. Maniezzo, A. Colorni: Positive feedback as a search strategy. Techn. Rep. 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991

54.19 M. Dorigo, V. Maniezzo, A. Colorni: Ant System: Optimization by a colony of cooperating agents, IEEE Trans. Syst. Man. Cybern. B **26**(1), 29–41 (1996)

54.20 M. Dorigo, T. Stützle: *Ant Colony Optimization* (MIT, Cambridge 2004)

54.21 T. Stützle, M. Birattari, H.H. Hoos: Engineering stochastic local search algorithms – designing, implementing and analyzing effective heuristics, Lect. Notes Comput. Sci. **4638**, 1–221 (2007)

54.22 T. Stützle, M. Birattari, H.H. Hoos: Engineering stochastic local search algorithms – designing, implementing and analyzing effective heuristics, Lect. Notes Comput. Sci. **5217**, 1–155 (2009)

54.23 H.H. Hoos: Programming by optimization, Commun. ACM **55**, 70–80 (2012)

54.24 S. Khanna, R. Motwani, M. Sudan, U. Vazirani: On syntactic versus computational views of approximability, Proc. 35th Annu. IEEE Symp. Found. Comput. Sci. (IEEE Computer Society, Los Alamitos 1994) pp. 819–830

54.25 F. Glover, M. Laguna: *Tabu Search* (Kluwer, Boston 1997)

54.26 K. Sörensen, F. Glover: Metaheuristics. In: *Encyclopedia of Operations Research and Management Science*, ed. by S.I. Gass, M.C. Fu (Springer, Berlin 2013) pp. 960–970

54.27 C.H. Papadimitriou, K. Steiglitz: *Combinatorial Optimization – Algorithms and Complexity* (Prentice Hall, Englewood Cliffs 1982)

54.28 J.L. Bentley: Fast algorithms for geometric traveling salesman problems, ORSA J. Comput. **4**(4), 387–411 (1992)

54.29 O.C. Martin, S.W. Otto, E.W. Felten: Large-step Markov chains for the traveling salesman problem, Complex Syst. **5**(3), 299–326 (1991)

54.30 D.S. Johnson, L.A. McGeoch: The traveling salesman problem: A case study in local optimization. In: *Local Search in Combinatorial Optimization*, ed. by E.H.L. Aarts, J.K. Lenstra (Wiley, Chichester 1997) pp. 215–310

54.31 A.S. Jain, B. Rangaswamy, S. Meeran: New and "stronger" job-shop neighbourhoods: A focus on the method of Nowicki and Smutnicki, J. Heuristics **6**(4), 457–480 (2000)

54.32 R.K. Congram, C.N. Potts, S. van de Velde: An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem, INFORMS J. Comput. **14**(1), 52–67 (2002)

54.33 M. Yannakakis: The analysis of local search problems and their heuristics, Lect. Notes Comput. Sci. **415**, 298–310 (1990)

54.34 R. Battiti, M. Protasi: Reactive search, a history-based heuristic for MAX-SAT, ACM J. Exp. Algorithmics **2**, 2 (1997)

54.35 P. Hansen, N. Mladenović: Variable neighborhood search: Principles and applications, Eur. J. Oper. Res. **130**(3), 449–467 (2001)

54.36 P. Hansen, N. Mladenović: Variable neighborhood search. In: *Handbook of Metaheuristics*, ed. by F. Glover, G. Kochenberger (Kluwer, Norwell 2002) pp. 145–184

54.37 R.K. Ahuja, O. Ergun, J.B. Orlin, A.P. Punnen: A survey of very large-scale neighborhood search techniques, Discrete Appl. Math. **123**(1–3), 75–102 (2002)

54.38 B.W. Kernighan, S. Lin: An efficient heuristic procedure for partitioning graphs, Bell Syst. Technol. J. **49**, 213–219 (1970)

54.39 S. Lin, B.W. Kernighan: An effective heuristic algorithm for the traveling salesman problem, Oper. Res. **21**(2), 498–516 (1973)

54.40 F. Glover: Ejection chain, reference structures and alternating path methods for traveling salesman problems, Discrete. Appl. Math. **65**(1–3), 223–253 (1996)

54.41 R.K. Ahuja, O. Ergun, J.B. Orlin, A.P. Punnen: Very large-scale neighborhood search. In: *Handbook*

of Approximation Algorithms and Metaheuristics, Computer and Information Science Series, ed. by T.F. Gonzalez (Chapman Hall/CRC, Boca Raton 2007) pp. 1–12
54.42 I. Dumitrescu: Constrained Path and Cycle Problems, Ph.D. Thesis (University of Melbourne, Department of Mathematics and Statistics 2002)
54.43 M. Chiarandini, I. Dumitrescu, T. Stützle: Very large-scale neighborhood search: Overview and case studies on coloring problems. In: *Hybrid Metaheuristics – An Emergent Approach to Optimization*, Studies in Computational Intelligence, Vol. 117, ed. by C. Blum, M.J. Blesa Aguilera, A. Roli, M. Sampels (Springer, Berlin 2008) pp. 117–150
54.44 C.N. Potts, S. van de Velde: Dynasearch: Iterative local improvement by dynamic programming; Part I, the traveling salesman problem. Techn. Rep. LPOM-9511, Faculty of Mechanical Engineering, University of Twente, Enschede, The Netherlands, 1995
54.45 P.M. Thompson, J.B. Orlin: The theory of cycle transfers, Working Paper OR 200-89, Operations Research Center, MIT, Cambridge 1989
54.46 P.M. Thompson, H.N. Psaraftis: Cyclic transfer algorithm for multivehicle routing and scheduling problems, Oper. Res. **41**, 935–946 (1993)
54.47 K. Helsgaun: An effective implementation of the Lin–Kernighan traveling salesman heuristic, Eur. J. Oper. Res. **126**(1), 106–130 (2000)
54.48 A. Grosso, F. Della Croce, R. Tadei: An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem, Oper. Res. Lett. **32**(1), 68–72 (2004)
54.49 B. Selman, H. Kautz: Domain-independent extensions to GSAT: Solving large structured satisfiability problems, Proc. 13th Int. Jt. Conf. Artif. Intell., ed. by R. Bajcsy (Morgan Kaufmann, San Francisco 1993) pp. 290–295
54.50 B. Selman, H. Kautz, B. Cohen: Noise strategies for improving local search, Proc. 12th Natl. Conf. Artif. Intell., AAAI/The MIT (1994) pp. 337–343
54.51 O. Steinmann, A. Strohmaier, T. Stützle: Tabu search vs. random walk, Lect. Notes Artif. Intell. **1303**, 337–348 (1997)
54.52 O.J. Mengshoel: Understanding the role of noise in stochastic local search: Analysis and experiments, Artif. Intell. **172**(8/9), 955–990 (2008)
54.53 D.T. Connolly: An improved annealing scheme for the QAP, Eur. J. Oper. Res. **46**(1), 93–100 (1990)
54.54 M. Fielding: Simulated annealing with an optimal fixed temperature, SIAM J. Optim. **11**(2), 289–307 (2000)
54.55 E.H.L. Aarts, J.H.M. Korst, P.J.M. van Laarhoven: Simulated annealing. In: *Local Search in Combinatorial Optimization*, ed. by E.H.L. Aarts, J.K. Lenstra (Wiley, Chichester 1997) pp. 91–120
54.56 A.G. Nikolaev, S.H. Jacobsen: Simulated annealing. In: *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, Vol. 146, ed. by M. Gendreau, J.-Y. Potvin (Springer, New York 2010) pp. 1–40 2 edition, chapter 8
54.57 R. Battiti, G. Tecchiolli: Simulated annealing and tabu search in the long run: A comparison on QAP tasks, Comput. Math. Appl. **28**(6), 1–8 (1994)
54.58 C. Voudouris: Guided Local Search for Combinatorial Optimization Problems, Ph.D. Thesis (University of Essex, Department of Computer Science, Colchester 1997)
54.59 C. Voudouris, E. Tsang: Guided local search and its application to the travelling salesman problem, Eur. J. Oper. Res. **113**(2), 469–499 (1999)
54.60 Y. Shang, B.W. Wah: A discrete Lagrangian-based global-search method for solving satisfiability problems, J. Glob. Optim. **12**(1), 61–100 (1998)
54.61 D. Schuurmans, F. Southey, R.C. Holte: The exponentiated subgradient algorithm for heuristic boolean programming, Proc. 17th Int. Jt. Conf. Artif. Intell., ed. by B. Nebel (Morgan Kaufmann, San Francisco 2001) pp. 334–341
54.62 F. Hutter, D.A.D. Tompkins, H.H. Hoos: Scaling and probabilistic smoothing: Efficient dynamic local search for SAT, Lect. Notes Comput. Sci. **2470**, 233–248 (2002)
54.63 W.J. Pullan, H.H. Hoos: Dynamic local search for the maximum clique problem, J. Artif. Intell. Res. **25**, 159–185 (2006)
54.64 M.G.C. Resende, C.C. Ribeiro: Greedy randomized adaptive search procedures: Advances and applications. In: *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, Vol. 146, ed. by M. Gendreau, J.-Y. Potvin (Springer, New York 2010) pp. 281–317
54.65 G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, G. Dueck: Record breaking optimization results using the ruin and recreate principle, J. Comput. Phys. **159**(2), 139–171 (2000)
54.66 A. Cesta, A. Oddi, S.F. Smith: Iterative flattening: A scalable method for solving multi-capacity scheduling problems, Proc. 17th Natl. Conf. Artif. Intell., AAAI/The MIT (2000) pp. 742–747
54.67 A.J. Richmond, J.E. Beasley: An iterative construction heuristic for the ore selection problem, J. Heuristics **10**, 153–167 (2004)
54.68 L.W. Jacobs, M.J. Brusco: A local search heuristic for large set-covering problems, Nav. Res. Logist. **42**(7), 1129–1140 (1995)
54.69 R. Ruiz, T. Stützle: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, Eur. J. Oper. Res. **177**(3), 2033–2049 (2007)
54.70 R. Ruiz, T. Stützle: An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives, Eur. J. Oper. Res. **187**(3), 1143–1159 (2008)
54.71 D.S. Johnson, L.A. McGeoch: Experimental analysis of heuristics for the STSP. In: *The Travel-*

ing Salesman Problem and its Variations, ed. by G. Gutin, A. Punnen (Kluwer, Dordrecht, The Netherlands 2002) pp. 369–443

54.72 D. Applegate, W. Cook, A. Rohe: Chained Lin-Kernighan for large traveling salesman problems, INFORMS J. Comput. **15**(1), 82–92 (2003)

54.73 H.R. Lourenço, O. Martin, T. Stützle: Iterated local search: Framework and applications. In: *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, Vol. 146, ed. by M. Gendreau, J.-Y. Potvin (Springer, New York 2010) pp. 363–397

54.74 T. Stützle: Iterated local search for the quadratic assignment problem, Eur. J. Oper. Res. **174**(3), 1519–1539 (2006)

54.75 I. Hong, A.B. Kahng, B.R. Moon: Improved large-step Markov chain variants for the symmetric TSP, J. Heuristics **3**(1), 63–81 (1997)

54.76 S. Goss, S. Aron, J.L. Deneubourg, J.M. Pasteels: Self-organized shortcuts in the Argentine ant, Naturwissenschaften **76**, 579–581 (1989)

54.77 J.-L. Deneubourg, S. Aron, S. Goss, J.-M. Pasteels: The self-organizing exploratory pattern of the Argentine ant, J. Insect Behav. **3**, 159–168 (1990)

54.78 T. Stützle, H.H. Hoos: *MAX–MIN* ant system, Future Gener. Comput. Syst. **16**(8), 889–914 (2000)

54.79 M. Dorigo, M. Birattari, T. Stützle: Ant colony optimization: Artificial ants as a computational intelligence technique, IEEE Comput. Intell. Mag. **1**(4), 28–39 (2006)

54.80 M. Dorigo, T. Stützle: Ant colony optimization: Overview and recent advances. In: *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, Vol. 146, ed. by M. Gendreau, J.-Y. Potvin (Springer, New York 2010) pp. 227–263

54.81 M. Dorigo, G. Di Caro: The ant colony optimization meta-heuristic. In: *New Ideas in Optimization*, ed. by D. Corne, M. Dorigo, F. Glover (McGraw Hill, London 1999) pp. 11–32

54.82 M. Dorigo, G. Di Caro, L.M. Gambardella: Ant algorithms for discrete optimization, Artif. Life **5**(2), 137–172 (1999)

54.83 E. Bonabeau, M. Dorigo, G. Theraulaz: *Swarm Intelligence: From Natural to Artificial Systems* (Oxford Univ. Press, New York 1999)

54.84 J.-Y. Potvin: Genetic algorithms for the traveling salesman problem, Ann. Oper. Res. **63**, 339–370 (1996)

54.85 P. Merz, B. Freisleben: Memetic algorithms for the traveling salesman problem, Complex Syst. **13**(4), 297–345 (2001)

54.86 P. Moscato: Memetic algorithms: A short introduction. In: *New Ideas in Optimization*, ed. by D. Corne, M. Dorigo, F. Glover (McGraw Hill, London 1999) pp. 219–234

54.87 M. Laguna, R. Martí: *Scatter Search: Methodology and Implementations in C*, Vol. 24 (Kluwer, Boston 2003)

54.88 M. Ehrgott, X. Gandibleux: Approximative solution methods for combinatorial multicriteria optimization, TOP **12**(1), 1–88 (2004)

54.89 M. Ehrgott, X. Gandibleux: Hybrid metaheuristics for multi-objective combinatorial optimization. In: *Hybrid Metaheuristics: An emergent approach for optimization*, ed. by C. Blum, M.J. Blesa, A. Roli, M. Sampels (Springer, Berlin, Germany 2008) pp. 221–259

54.90 L. Paquete, T. Stützle: Stochastic local search algorithms for multiobjective combinatorial optimization: A review. In: *Handbook of Approximation Algorithms and Metaheuristics*, Computer and Information Science Series, ed. by T.F. Gonzalez (Chapman Hall/CRC, Boca Raton 2007) pp. 1–15

54.91 L. Bianchi, M. Dorigo, L.M. Gambardella, W.J. Gutjahr: A survey on metaheuristics for stochastic combinatorial optimization, Nat. Comput. **8**(2), 239–287 (2009)

54.92 D. Ouelhadj, S. Petrovic: A survey of dynamic scheduling in manufacturing systems, J. Sched. **12**(4), 417–431 (2009)

54.93 V. Pillac, M. Gendreau, C. Guéret, A. L. Medaglia: A review of dynamic vehicle routing problems. Techn. Rep. CIRRELT-2011-62, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, Montréal, Canada, October 2011

54.94 V. Maniezzo, T. Stützle, S. Voß (Eds.): *Matheuristics – Hybridizing Metaheuristics and Mathematical Programming*, Annals of Information Systems, Vol. 10 (Springer, New York 2010)

54.95 J. Puchinger, G.R. Raidl, S. Pirkwieser: MetaBoosting: Enhancing integer programming techniques by metaheuristics. In: *Matheuristics – Hybridizing Metaheuristics and Mathematical Programming*, Annals of Information Systems, Vol. 10, ed. by V. Maniezzo, T. Stützle, S. Voß (Springer, New York 2010) pp. 71–102

54.96 M. Fischetti, A. Lodi: Local branching, Math. Program. **98**(1/3), 23–47 (2003)

54.97 E. Danna, E. Rothberg, C. Le Pape: Exploring relaxation induced neighborhoods to improve mip solutions, Math Program. **102**(1), 71–90 (2005)

54.98 V. Maniezzo: Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem, INFORMS J. Comput. **11**(4), 358–369 (1999)

54.99 C. Blum: Beam-ACO for simple assembly line balancing, INFORMS J. Comput. **20**(4), 618–627 (2008)

54.100 W. Cook, P. Seymour: Tour merging via branch-decomposition, INFORMS J. Comput. **15(3)**, 233–248 (2003)

54.101 M.A. Boschetti, V. Maniezzo: Benders decomposition, Lagrangean relaxation and metaheuristic design, J. Heuristics **15**(3), 283–312 (2009)

54.102 I. Dumitrescu, T. Stützle: Usage of exact algorithms to enhance stochastic local search algorithms. In: *Matheuristics – Hybridizing Meta-*

heuristics and Mathematical Programming, Annals of Information Systems, Vol. 10, ed. by V. Maniezzo, T. Stützle, S. Voß (Springer, New York 2010) pp. 103–134

54.103 L. Jourdan, M. Basseur, E.-G. Talbi: Hybridizing exact methods and metaheuristics: A taxonomy, Eur. J. Oper. Res. **199**(3), 620–629 (2009)

54.104 C. Demetrescu, I. Finocchi, G.F. Italiano: Algorithm engineering, Bulletin EATCS **79**, 48–63 (2003)

54.105 I. Sommerville (Ed.): *Software Engineering*, 7th edn. (Addison Wesley, Boston 2004)

54.106 P. Balaprakash, M. Birattari, T. Stützle: Engineering stochastic local search algorithms: A case study in estimation-based local search for the probabilistic traveling salesman problem. In: *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, Studies in Computational Intelligence, Vol. 153, ed. by C. Cotta, J. van Hemert (Springer, Berlin 2008) pp. 55–69

54.107 S. Cahon, N. Melab, E.-G. Talbi: ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics, J. Heuristics **10**(3), 357–380 (2004)

54.108 Paradiseo: A Software Framework for Metaheuristics, http://paradiseo.gforge.inria.fr

54.109 L. Di Gaspero, A. Schaerf: Writing local search algorithms using EASYLOCAL++. In: *Optimization Software Class Libraries*, ed. by S. Voß, D.L. Woodruff (Kluwer, Boston, 2002) pp. 155–175

54.110 Atlassian Bitbucket: https://bitbucket.org/satt/easylocal-3

54.111 P. Van Hentenryck, L. Michel: *Constraint-Based Local Search* (MIT, Cambridge 2005)

54.112 K. Mehlhorn, S. Näher: *LEDA: A Platform for Combinatorial and Geometric Computing* (Cambridge Univ. Press, Cambridge 1999)

54.113 The R Project for Statistical Computing, http://www.r-project.org

54.114 C.W. Nell, C. Fawcett, H.H. Hoos, K. Leyton-Brown: HAL: A framework for the automated design and analysis of high-performance algorithms, Lect. Notes Comput. Sci. **6683**, 600–615 (2011)

54.115 HAL: The High-performance Algorithm Laboratory, http://hal.cs.ubc.ca/

54.116 P. Merz, B. Freisleben: Fitness landscapes and memetic algorithm design. In: *New Ideas in Optimization*, ed. by D. Corne, M. Dorigo, F. Glover (McGraw Hill, London 1999) pp. 244–260

54.117 L. Xu, H. Hoos, K. Leyton-Brown: Hierarchical hardness models for SAT, Lect. Notes Comput. Sci. **4741**, 696–711 (2007)

54.118 J.-P. Watson, L.D. Whitley, A.E. Howe: Linking search space structure, run-time dynamics, and problem difficulty: A step towards demystifying tabu search, J. Artif. Intell. Res. **24**, 221–261 (2005)

54.119 H.H. Hoos: Automated algorithm configuration and parameter tuning. In: *Autonomous Search*, ed. by Y. Hamadi, E. Monfroy, F. Saubion (Springer, Berlin 2012) pp. 37–71

54.120 B. Adenso-Díaz, M. Laguna: Fine-tuning of algorithms using fractional experimental designs and local search, Oper. Res. **54**(1), 99–114 (2006)

54.121 S.P. Coy, B.L. Golden, G.C. Runger, E.A. Wasil: Using experimental design to find effective parameter settings for heuristics, J. Heuristics **7**(1), 77–97 (2001)

54.122 T. Bartz-Beielstein: *Experimental Research in Evolutionary Computation – The New Experimentalism* (Springer, Berlin 2006)

54.123 F. Hutter, H.H. Hoos, K. Leyton-Brown, K.P. Murphy: An experimental investigation of model-based parameter optimisation: SPO and beyond, Genet. Evol. Comput. Conf., GECCO 2009, ed. by F. Rothlauf (ACM, New York 2009) pp. 271–278

54.124 M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp: A racing algorithm for configuring metaheuristics, Proc. Genet. Evol. Comput. Conf. (GECCO-2002), ed. by W.B. Langdon, E. Cantú-Paz, K.E. 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.K. Burke, N. Jonoska (Morgan Kaufmann, San Francisco 2002) pp. 11–18

54.125 M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle: F-Race and iterated F-Race: An overview. In: *Experimental Methods for the Analysis of Optimization Algorithms*, ed. by T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (Springer, Berlin, Germany 2010) pp. 311–336

54.126 F. Hutter, H.H. Hoos, T. Stützle: Automatic algorithm configuration based on local search, Proc. 22nd Conf. Artif. Intell. (AAAI), ed. by R.C. Holte, A. Howe (AAAI / The MIT, Menlo Park 2007) pp. 1152–1157

54.127 C. Ansótegui, M. Sellmann, K. Tierney: A gender-based genetic algorithm for the automatic configuration of algorithms, Proc. 15th Int. Conf. Princ. Pract. Constraint Program. (CP 2009) (2009) pp. 142–157

54.128 F. Hutter, H.H. Hoos, K. Leyton-Brown, T. Stützle: Param ILS: An automatic algorithm configuration framework, J. Artif. Intell. Res. **36**, 267–306 (2009)

54.129 F. Hutter, H.H. Hoos, K. Leyton-Brown: Sequential model-based optimization for general algorithm configuration, Lect. Notes Comput. Sci. **6683**, 507–523 (2011)

54.130 F. Hutter, H.H. Hoos, K. Leyton-Brown: Parallel algorithm configuration, Lect. Notes Comput. Sci. **7219**, 55–70 (2011)

54.131 R. Battiti, M. Brunato, F. Mascia: *Reactive Search and Intelligent Optimization*, Operations Research/Computer Science Interfaces Series, Vol. 45 (Springer, New York 2008)

54.132 A.E. Eiben, Z. Michalewicz, M. Schoenauer, J.E. Smith: Parameter control in evolutionary algorithms. In: *Parameter Setting in Evolutionary Algorithms*, ed. by F. Lobo, C.F. Lima, Z. Michalewicz (Springer, Berlin, Germany 2007) pp. 19–46

54.133 F. Hutter, Y. Hamadi, H.H. Hoos, K. Leyton-Brown: Performance prediction and automated tuning of randomized and parametric algorithms, Lect. Notes Comput. Sci. **4204**, 213–228 (2006)

54.134 L. Xu, H.H. Hoos, K. Leyton-Brown: Hydra: Automatically configuring algorithms for portfolio-based selection, Proc. 24th AAAI Conf. Artif. Intell. (AAAI-10) (2010) pp. 210–216

54.135 S. Kadioglu, Y. Malitsky, M. Sellmann, K. Tierney: ISAC – Instance-specific algorithm configuration, Proc. 19th Eur. Conf. Artif. Intell. (ECAI 2010) (2010) pp. 751–756

54.136 H.H. Hoos: Computer-aided algorithm design using generalised local search machines and related design patterns. Techn. Rep. TR-2009-26, University of British Columbia, Department of Computer Science, 2009