Chapter 3
Local Search

Marco A. Montes de Oca, Carlos Cotta, and Ferrante Neri

3.1 Basic Concepts

At an abstract level, memetic algorithms can be seen as a broad class of population-based stochastic local search (SLS) methods, where a main theme is "exploiting *all available knowledge* about a problem," see also Moscato and Cotta [618], page 105. The most wide-spread implementation of this theme is probably that of improving some or all individuals in the population by some local search method. This combination of a population-based, global search and a single-solution local search is a very appealing one. The global search capacity of the evolutionary part of a memetic algorithm takes care of exploration, trying to identify the most promising search space regions; the local search part scrutinizes the surroundings of some initial solution, exploiting it in this way. This idea is not only an appealing one, it is also practically a very successful one. In fact, for a vast majority of combinatorial optimization problems and, as it is also becoming more clear in recent research, also for many continuous optimization problems this combination leads to some of best performing heuristic optimization algorithms.

The role of the local search is fundamental and the selection of its search rule and its harmonization within the global search schemes make the global algorithmic success of memetic frameworks. The local search can be integrated within the

Marco A. Montes de Oca
IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
e-mail: mmontes@ulb.ac.be

Carlos Cotta
Department of Lenguajes y Ciencias de la Computación, University of Malaga, Spain
e-mail: ccottap@lcc.uma.es

Ferrante Neri
Department of Mathematical Information Technology, P.O. Box 35 (Agora), 40014, University of Jyväskylä, Finland
e-mail: ferrante.neri@jyu.fi

F. Neri et al. (Eds.): Handbook of Memetic Algorithms, SCI 379, pp. 29–41.
springerlink.com

evolutionary cycle mainly in two ways. The first is the so called "life-time learning", i.e., the application of the local search to a candidate solution. In this case the metaphor is the cultural development of the individuals which is then transmitted to the other solutions over the subsequent generations. The second way is the application of the local search during the solution generation, i.e., the generation of a perfect child. This class of memetic implementations aims at selecting the most convenient offspring amongst the potential offspring solutions. This aim can be achieved, for example, by applying a local search to select the most convenient cutting point in a genetic algorithm crossover, or by using some kind of complete technique for this purpose [164, 165] – see Chapter 12.

In this chapter, we focus on the local search techniques and give a concise overview of local improvement methods both for combinatorial and for continuous optimization problems. Rather than giving a detailed account of all intricacies of local search algorithms, we focus on the main concepts and review some of the main local search variants. We start by first introducing some basic notions about neighborhoods and local optima. Next, we discuss the characterization of the local search and illustrate in details, some examples of local search algorithms in combinatorial and continuous domains. Without loss of generality, we assume here that we deal with minimization problems. Obviously, everything discussed here can easily be adapted to the maximization case.

3.2 Neighborhoods and Local Optima

The notion of neighborhood is essential to understand local search. Intuitively, a solution s' is termed a *neighbor* of s if the former can be reached from the latter in a single step (by the application of a so-called move operator). The neighborhood $\mathcal{N}(s)$ of a solution s is the set of all its neighbors. Notice firstly that neighborhood relationships are often –but not always– symmetrical. Secondly, the existence of a move operator allows not having to define neighborhoods explicitly (by enumerating the neighbors) but just implicitly (by referring to the potential transitions attainable upon application of the move operator). Moves can be typically regarded as modifications of some parts of a solution. Under an appropriate distance measure between solutions, these moves can thus be seen as "local", hence the name of the search paradigm. This said, the notion of closeness between neighboring solutions is not straightforward; as a matter of fact, neighborhood relationships can be quite complex – see [624].

Local search algorithms in combinatorial and continuous spaces have some intrinsic differences due to the differences in the types of the underlying search spaces. Let us denote the search space by S in the following. While combinatorial search spaces are finite for finite size problems, continuous search spaces are infinite and not enumerable. From these differences some other differences also result in the notions of local optima and the way how one is searching for improved candidate solutions. Intuitively, a local optimum is a solution for which in its local neighborhood no better solution exists.

In combinatorial problems, the number of candidate solutions in the neighborhood of a current candidate solution s is enumerable and a local optimum can be defined as a a candidate solution s^l for which it holds that $\forall s \in \mathcal{N}(s^l)$ we have $f(s) \leqslant f(s^l)$, where $f : s \mapsto R$ is the objective function. It is also easy to identify whether a candidate solution is a local optimum, since one simply needs to enumerate all neighboring candidate solutions and check whether they are better or not than the current candidate solution. If none such exists, the solution is a local optimum. If the number of neighbors is polynomial in the instance size and the objective function is computable in polynomial time, which is the typical case for many neighborhood definitions and optimization problems, then this check can be also done in polynomial time.

For continuous optimization problems, the decision space is in principle a dense set and is thus composed of an infinite amount of points. This makes the enumeration impossible for the search of the optimum. It must be remarked that in the representation within a (finite) machine, dense sets cannot be represented with an infinite set and the sets are technically finite and the distance between each pairs of points is at least equal to the machine precision. An extensive description of this topic is given in Chapter 8, where continuous optimization problems are discussed. We can formally define a *local optimum* in a continuous space as a point $s^\circ \in S$, such that

$$f(s^\circ) \leqslant f(s)\,, \tag{3.1}$$

for all points $s \in S$ that satisfy the relation $0 \leqslant ||s^\circ - s|| \leqslant \varepsilon$. The set of points encircled in the region limited by the magnitude of ε is the *neighborhood* of the local optimum s°. Note that any global optimum is also a local optimum; however, the opposite is not necessarily true.

According to an alternative representation, in continuous optimization, a *global optimum* is a solution $s^\star \in S$, such that

$$s^\star = \underset{s \in S}{\operatorname{argmin}} f(s)\,, \tag{3.2}$$

where $S \subseteq \mathbb{R}^n$ is the feasible search space, and $f : S \mapsto \mathbb{R}$ is the cost function to minimize. When $S = \mathbb{R}^n$ the problem of finding the optimum of f is called *unconstrained*, otherwise it is said to be *constrained*.

3.3 Classifications of Local Search

At a general level, for both combinatorial and continuous domains, local search algorithms can be classified from various perspectives:

1. According to the nature of the search logic:
   - **Stochastic:** the generation of the trial solution occurs in a randomized way
   - **Deterministic:** the generation of the trial solution is deterministic

2. According to the amount of solutions involved
    - **Single-solution**: the algorithm processes and perturbs only one solution
    - **Multiple-solution**: the algorithm processes more that one solution which are usually employed for interacting and jointly generate trial solutions
3. According to the pivot rule:
    - **Steepest Descent**: the algorithm generates a set of solutions and selects the most promising only after having explored all the possibilities
    - **Greedy**: the algorithm performs the replacement as soon as detects a solution outperforming the current best and starts over the exploration

On the basis of these classifications two important considerations must be carried out. First, every local search algorithm and, more generally, every optimization algorithm, can be seen as a logical procedure composed of two sets of operations: 1) generation of trial solutions 2) selection of trial solutions. In this light the first two classifications above characterize some crucial aspects of the trial solution generation while the latter regards the selection phase. Second, the classifications should not be considered in a binary way but more as properties of the procedure phases. For example, an algorithm is not either fully stochastic or fully deterministic but is likely to have a certain degree of stochastic logic and determinism. For two given algorithms a meaningful statement is to establish which among those two is "more" deterministic. To give a more concrete example, the simplex proposed in [653] cannot be considered a stochastic algorithm but due to the randomized re-sampling of the solutions is more stochastic than the Powell algorithm proposed in [728]. In a similar way, the classification according to the pivot rule must be interpreted as a property of the selection phase. An algorithm which explores/enumerates the entire neighborhood of a trial solution is possible as well as an algorithm which centers the search in the new current best solution. On the other hand, several intermediate possibilities can also be implemented and lead to efficient algorithms.

In this light, also the concepts of local and global searches should be seen as a progressive property of the algorithm. Intuitively, a local search is thought of as an algorithmic structure converging to the closest local optimum while the global search should have the potential of detecting the global optimum. De facto this definition is not rigorous and does not correspond to the actual behavior of the algorithms. For example, a hill-descender with a fully deterministic candidate solution generation can potentially converge to the global optimum of a highly multi-modal fitness landscape if a proper initial radius is set. Dually, an evolutionary algorithm with a population composed of a few individuals can converge to a suboptimal solution in the neighborhood of the best individual of the initial population. In this sense, an algorithm is not simply either global or local but can be characterized by a certain degree of local/global search features.

3.4 Local Search in Combinatorial Domains

Combinatorial spaces constitute a formidable and challenging application domain for local search techniques. Unlike continuous spaces in which neighborhood relationships are naturally defined in terms of Euclidean distance or any other suitable metric on $\mathbb{R}^n$, the definition of neighborhood in discrete domains is a major step in the resolution of the problem under consideration. Following the terminology used in forma analysis [750], we can consider situations in which the representation of solutions is orthogonal –i.e., solutions are represented as a collection of variables v_i, each of them taking values from a domain $\mathscr{D}_i$ and any combination of values for different variables being feasible– and situations in which the representation is non-orthogonal. In the former case the (feasible) search space S is

$$S = \prod_{i=1}^{n} \mathscr{D}_i, \tag{3.3}$$

i.e., the Cartesian product of variable domains. It is then possible to define neighborhood relationships on the basis of modifications of single variables. An illustrative example is that of binary strings, the typical representation in traditional genetic algorithms. Solutions thus belong to $\mathbb{B}^n$, and we can define a collection of nested neighborhoods $\mathscr{N}_i(s) = \{s' \mid H(s,s') = i\}$, where $H(s,s')$ is the Hamming distance between s and s'. This example is easily generalizable to integer representations, and can be further extended by considering the L_1 norm (Manhattan distance) or alike in case there was some locality between numerically close variable values.

The case of non-orthogonal representations is more complex since not every combination of variable values is feasible in them. This means that moves must be done solution-wise rather than variable-wise. A popular example is that of permutations: solutions are in this case arrangements of a collection of n objects (the integers $1, \cdots, n$ typically), and no single variable can be modified in isolation. There are numerous possibilities for perturbing permutations though. For example:

- *swap*: two elements are selected and interchanged.
- *insert*: an element is selected, removed from its location and inserted at another place.
- *invert*: a subsequence of adjacent elements is selected and their ordering is reversed.

just to name a few (please check chapter 7 of [764] for an overview of perturbation strategies for permutations in the context of the Traveling Salesman Problem). The situation can obviously be more complex in cases where other more sophisticated structures are used as representation of solutions, e.g., trees [19, 152, 723].

The neighborhoods sketched above are central in the functioning of single solution metaheuristics. These algorithms process one solution and after subsequent modifications returns a solution which is supposed to be similar to the starting solution but characterized by a higher performance. Within the context of MAs, these local searches process one solution within the evolutionary framework and improve during their "life-time".

Algorithm 2. A Local Search Algorithm

```
1  Procedure Local Search (s);
2  begin
3      InitializeMemory(M);
4      repeat
5          N ← PickNeighborhoodStructure(M);
6          s' ← PickNeighbor(N, s);
7          Select(s, s', M);
8          UpdateMemory(s, M);
9      until TerminationCriterion(M);
10     return s;
11 end
```

Algorithm 2 provides a rather general outline of a single-solution metaheuristic. This algorithm receives an initial solution and iteratively picks a neighbor and decides whether to accept this neighbor as the new current solution or not. This process can be modulated by a memory structure that the algorithm may use in order to decide which neighborhood should be used to select the neighbor, whether to accept the latter as new current solution or not, and even to support some high-level strategy for intensifying or diversifying the search.

One of the most distinctive features of local search strategies in combinatorial domains is the possibility of performing some kind of incremental evaluation of neighbors, i.e., computing $f(s')$ as $f(s') = f(s) + \Delta f(s,s')$, where $\Delta f(s,s')$ is a term that depends on the perturbation exerted on s to obtain s' and can be typically computed in a simple and efficient way. This often means that the cost of exploring the neighborhood of a solution is not much higher than a few full evaluations, thus allowing the practical use of intensive local search procedures. The following subsections give some examples of single-solution local search algorithms.

3.4.1 Hill Climbing

The simplest way to perform the local search in a combinatorial space is obtained by perturbing a trial solution and replacing it with the perturbed solution when the newly generate candidate solution outperforms the solution before perturbation. This procedure is termed Hill Climbing (HC) –or hill descending in a minimization context– and can be characterized in terms of the pseudocode depicted in Algorithm 2 as a memoryless, single-neighborhood local search procedure.

There can be several variants of the algorithm depending on, for instance, the pivot rule as mentioned in Sect. 3.3. Thus, we have steepest-ascent HC (an algorithm that explores the full neighborhood $\mathcal{N}(s)$ of the current solution s , picks the best neighbor, accepts it if it is better than the incumbent) and random HC (an algorithm that picks a single random neighbor $s' \in \mathcal{N}(x)$ and accepts it if is better than s). In the first case, the algorithm terminates upon finding a local optimum (i.e., a solution s which is better than any other $s' \in \mathcal{N}(x)$) or exhausting its computational

budget; in the second case only the computational limit applies, unless the algorithm keeps track of the neighbors generated and is capable of avoiding duplicates and/or detecting when the neighborhood is fully explored.

In some cases the neighborhood is very large and a full exploration is not possible (at least using the simple schema of HC – local branching procedures for exploring very large neighborhoods are possible though [263]). Such situations are usually dealt with by considering a random sampling of a certain size of the neighborhood, or by resorting to a simpler random HC. Another important issue is the presence of plateaus, i.e., when the best neighbor is neither better nor worse than the current solution. In that case the algorithm may opt for terminating, or may try to navigate through the plateau by accepting this best neighbor even if it is not strictly better than the incumbent. This is done for example in GSAT [804], a powerful solver for the MAX-SAT problem. Some strategy for avoiding cycling (i.e., oscillating search between a couple of solutions) may be required in this case. these are typically used in tabu search – see Sect. 3.4.3.

3.4.2 Simulated Annealing

While simple, the selection strategy depicted before for HC is unable to cope with rugged search spaces in which local optima are manifold, at least as an independent search technique. The search will terminate in a local optimum and it will have to be restarted from a different initial solution in order to locate other local optima (and eventually the global optimum). Such a restarting strategy is a simple way of endowing HC with global optimization capabilities, yet in some sense can be seen as a brute-force approach. More sophisticated strategies are however possible in order to escape from local optima. In particular, uphill moves (i.e., moves to worse solutions) must be at some point accepted. This is precisely the case of Simulated Annealing (SA) [122, 468]. SA is a single solution metaheuristic which performs the search of new solutions according to the logic described for HC but, in addition to performing the replacement when $f(s') \leqslant f(s)$, a worsening in the performance is accepted with a certain probability. This probability depends exponentially on the run-time, i.e., it is high at the beginning of the (local) optimization process and low at its end. The acceptance of only improvements may result into the fact that the algorithm gets stuck on a suboptimal solution. This condition is clearly undesired because if the algorithm does not succeed at improving its candidate solution for a large number of function calls, the algorithmic budget is wasted and the final solution is likely to have a poor performance. Thus, the main idea behind SA is to avoid such situation by refreshing the solution. The acceptance of a slightly worse solution should prevent the algorithm from getting stuck in some suboptimal solutions. To be precise, the neighboring configuration is accepted with probability P given by

$$P = \begin{cases} 1, & \text{if } \Delta f > 0 \\ e^{-\frac{\Delta f}{T}}, & \text{otherwise} \end{cases} \tag{3.4}$$

where T is a time-varying parameter termed *temperature*. This parameter modulates the acceptance probability, since the larger the temperature, the worse an acceptable neighbor can be. This parameter is decreased from its initial value T_0 to a final value $T_k < T_0$ via a process termed cooling schedule. There exist many cooling schedules in the literature, such as arithmetic, geometric or hyperbolic cooling [867]. In addition, there exist adaptive cooling strategies that take into account the evolution of the search, performing cooling and reheating as required, e.g., [243].

3.4.3 Tabu Search

Tabu Search (TS) is memory-based local search metaheuristic [309, 310, 317] with global optimization capabilities. It can be regarded a sophisticated extension of basic HC in which the best neighboring solution is chosen as the next configuration, even if it is worse than the current one. Notice that in case the neighborhood relationship is symmetric, it might happen that the best neighbor of solution s was s' and vice versa. In that case a memory-less search would simply cycle between these two solutions. TS avoids this by keeping a so-called *tabu list* of movements: a neighboring solution is accepted only if the corresponding move is not tabu. The actual meaning of a move being *tabu* may vary depending on the problem and designer's choice. Thus, it may be possible that it is tabu to restore a modified variable to a previous value, or it may be tabu to modify that variable at all (an analogous reasoning can be done when the moves are done solution-wise –e.g., as in permutations– rather than variable-wise). The tabu status of a move is not permanent: it only lasts for a number of search steps, whose value is termed *tabu tenure*. The tabu tenure can be fixed, or may vary during the search (even randomly within certain limits). The latter is useful to hinder long cycles in the search. In addition, it is common to consider an aspiration criterion that allows overriding the tabu status of a move, e.g., a move is accepted if it improves the best known solution even if it is tabu.

Algorithm 3 depicts the basic structure of a TS algorithm. It must be noted that a full-fledged TS algorithm is often endowed with additional strategies for intensifying and diversifying the search (e.g., frequency-based strategies that promote or penalize attribute values that occur frequently), and may also incorporate multiple neighborhoods among which the algorithm oscillates strategically.

3.5 Local Search in Continuous Domains

In this section, we turn our attention to optimization in continuous domains. In contrast to combinatorial optimization, where the search space is finite, in continuous optimization the search space is, in theory, infinite. Of course, in practice, the search space is also discrete as one is constrained by the precision of the representation of floating point numbers in the host computer. Nevertheless, its cardinality is so huge that for practical purposes it can be regarded as continuous.

Finding an extremum in a continuous domains is a different story. It is a very important task but it's difficult in general. The main difference with respect to the local

Algorithm 3. Tabu Search

```
1  Procedure Tabu Search (s, 𝒩);
2  begin
3      InitializeTabuList(M);
4      s* ← s;                                   // best known solution
5      repeat
6          L ← GenerateNeighborList(𝒩, s);
7          s' ← PickBest(L);
8          if IsTabu(s', M) and not AspirationCriteria(s', s*) then
9              s' ← PickBestNonTabu(L)
10         endif
11         UpdateTabuList(s, s', M);
12         s ← s';
13         if f(s*) > f(s) then
14             s* ← s
15         endif
16     until TerminationCriterion(M);
17     return s*;
18 end
```

search in discrete domain is the concept of gradient. More specifically, in continuous optimization, a local search can make use of gradient information in order to quickly descend the basin of attraction and thus support a global optimization framework which performs the selection of the search directions by means of the only fitness comparisons. For example, a trivial MA composed of an evolutionary framework and a hill-climber combines two alternative search logics: the fitness comparison of distant solutions generated within the decision space offered by the evolutionary framework and the gradient based search logic offered by the hill-climber. A proper combination of global and local search guarantees the success of a MA.

This section focuses on local search for continuous optimization problems. After some basic concepts and a classification of LS techniques for continuous domains, four popular algorithms are briefly described and a short survey on other techniques is given.

3.5.1 Classification of Local Search Techniques for Continuous Domains

The main criterion to classify local search methods for continuous domains is based on the order of the derivatives used for exploring the search space. Based on this criterion, optimization methods can be classified as follows:

Zeroth-order methods. Methods that belong to this class are called *direct search* methods. According to Trosset's definition [892], direct search methods are those that work with ordinal relations between objective function values. They do not

use the actual values to model, directly or indireclty, higher order properties of the objective function.

First-order methods. This class of methods rely on direct access to the objective function and to the gradient vector at any point. Methods referred to as "derivative-free", belong to this category if the zeroth-order information is used to approximate higher order properties of the objective function.

Second-order methods. These methods use objective function values, (numerical approximations of) gradient vectors, and (numerical approximations of) Hessian matrices.

A second classification criterion is whether a method is stochastic or deterministic. Stochastic methods make randomized choices during their execution, while deterministic methods do not. These randomized choices encompass solution generation and/or solution selection [393]. In memetic algorithms, both types of techniques are used; however, deterministic methods are more commonly used. A brief description of some of these techniques is presented in the following section.

3.5.2 Commonly Used Local Search Techniques in Memetic Algorithms for Continuous Domains

In this section, we describe some of the most commonly used local search techniques in the literature of memetic algorithms for continuous optimization. Additionally, each of the algorithms that are described in detail belong to different strategies for local optimization, namely, downhill, gradient, quasi-Newton, and trust-region strategies.

3.5.2.1 Downhill Strategy: Simplex Method

This method, which was proposed by Nelder and Mead [653], is used for minimizing an n-dimensional objective function f. It is based on a *simplex*, which is composed of a set of $n+1$ points $\mathscr{P} = \{P_0, P_1, \ldots, P_n\}$ in the search space. At each iteration of the algorithm, the simplex can modified by at least one of three operations: *reflection*, *contraction*, and *expansion*. We denote P_h and P_l as the points with the highest and lowest objective function values, respectively. $\bar{P}$ is the centroid of the points P_i such that $f(P_i) < f(P_h)\ \forall P_i \in \mathscr{P}$. The reflection of P_h, denoted by $P^\star$, is defined as

$$P^\star = (1+\alpha)\bar{P} - \alpha P_h, \tag{3.5}$$

where $\alpha > 0$ is a parameter called *reflection coefficient*. If $f(P_l) < f(P^\star) < f(P_h)$, then P_h is replaced by $P^\star$, and the algorithm starts a new iteration. If $f(P^\star) < f(P_l)$, then $P^\star$ is expaned to $P^{\star\star}$ as follows

$$P^{\star\star} = \gamma P^\star + (1-\gamma)\bar{P}, \tag{3.6}$$

where $\gamma > 1$ is a parameter called *expansion coefficient*. If $f(P^{\star\star}) < f(P_l)$, then P_h is replaced by $P^{\star\star}$, and the algorithm starts a new iteration. However, if $f(P^{\star\star}) >$

$f(P_l)$, then it is said that the expansion operation failed and P_h is replaced by $P^\star$ before continuing to the next iteration.

If $f(P^\star) > f(P_i)\ \forall P_i \in \mathscr{P} \setminus \{P_h\}$, then P_h is replaced by $P^\star$ only if $f(P^\star) < f(P_h)$, otherwise P_h remains unchanged. After this operation, a new point $P^{\star\star}$ is generated as follows

$$P^{\star\star} = \beta P_h + (1-\beta)\bar{P}, \tag{3.7}$$

where $0 < \beta < 1$ is a parameter called *contraction coefficient*. P_h is replaced by $P^{\star\star}$ unless $f(P^{\star\star}) > min\{f(P_h), f(P^\star)\}$, in which case all points P_i are replaced by $(P_i + P_l)/2$ before the next iteration begins.

The termination criterion can be either a minimum displacement in the search space, a minimum decrease in the objective function value, or a maximum number of iterations. This method has been used many times as a local search component of other algorithms (e.g., [123, 250, 383, 540, 607, 680, 719]).

3.5.2.2 Gradient Strategy: Powell's Direction Set Method

This method was proposed by M. J. D. Powell [728]. It tries to minimize an objective function $f : \mathbb{R}^n \to \mathbb{R}$ by constructing a set of *conjugate directions* through a series of line searches. Directions $v_i \ \ i \in \{1:n\}$ are said to be conjugate with respect to an $n \times n$ positive definite matrix A, if

$$v_i^T A v_j = 0\,, \quad \forall i, j \in \{1:n\}\ ,\ i \neq j\,. \tag{3.8}$$

Furthermore, to be conjugate, directions $v_i \ \ i \in \{1:n\}$ must be linearly independent.

Conjugate search directions are attractive because if A is the Hessian matrix of the objective function, it can be minimized in exactly n line searches [731]. Although Powell's method does not need information about the objective function's derivatives, it can be considered a gradient strategy because it uses (implicitly) second order properties of the objective function [801].

The basic procedure of this method is the following: First, from an initial point $P_0 \in \mathbb{R}^n$, it performs n line searches using the unit vectors e_i as initial search directions u_i. At each step, the new initial point from which the next line search is carried out is the point where the previous line search found a relative minimum. A point P_n denotes the minimum discovered after all n line searches. Second, the method eliminates the first search direction by doing $u_i = u_{i+1}\ \forall i \in \{1:n-1\}$, and replacing the last direction u_n for $P_n - P_0$. Next, a move to the minimum along the direction u_n is performed.

Performing n iterations of the procedure described above, would minimize a quadratic objective function using a total of $n(n+1)$ line searches. However, under some circumstances, it is possible that the set of constructed directions become linearly dependent, which would make the algorithm fail. To prevent this from happening, after n or $n+1$ iterations, the set of search directions can be reset to either the original unit vectors, or to the columns of an orthogonal matrix (e.g., obtained by computing the principal components of the old direction set). Another approach

is to substitute search directions in such a way as to maximize the determinant of the normalized direction vectors.

The method terminates when the magnitude of change of all variables is smaller than a predefined threshold. Powell's method has been used in numerous hybrid algorithms (e.g., [609, 680, 683, 766, 806]).

3.5.2.3 Quasi-Newton Strategy: Davidon-Fletcher-Powell Method

A Newton strategy uses the objective function's first and second order derivatives to rapidly optimize quadratic forms. In most practical cases, however, a quasi-Newton strategy is preferred. In a quasi-Newton strategy, the inverse of the objective function's Hessian is not computed directly, it is approximated from the objective function's gradient.

The Davidon-Fletcher-Powell method, also known as the variable metric strategy, was proposed by Fletcher and Powell [266], who based their proposal on Davidon's work [182]. It is an iterative procedure that works as follows. First, the user needs to specify an initial guess P_0 for the solution. An $n \times n$ matrix H_0 must also be initialized. Normally, $H_0 = I$. For any iteration k, we have

$$P_{k+1} = P_k - s_k H_k^T \nabla f(P_k), \tag{3.9}$$

where the step length s_k is computed by line search and it is the value that minimizes the objective function along the direction $-H_k^T \nabla f(P_k)$ from the current solution P_k.

The matrix H_k is updated as follows

$$H_{k+1} = H_k + \frac{y_k y_k^T}{y_k^T z_k} - \frac{H_k z_k (H_k z_k)^T}{z_k^T H_k z_k}, \tag{3.10}$$

where $y_k = P_{k+1} - P_k = -s_k H_k^T \nabla f(P_k)$, and $z_k = \nabla f(P_{k+1}) - \nabla f(P_k)$.

When the derivatives of the objective function are not available, the modifications introduced by Stewart [849] can be used. The Davidon-Fletcher-Powell method has been used as a local search component in hybrid local-global optimization algorithms, for instance in [29, 336, 510, 515].

3.5.2.4 Trust-Region Strategy: NEWUOA

In this category of methods, the objective function is approximated by a local model defined over a neighborhood of the current best solution. The quality of this model is trusted only in this neighborhood, therefore the name *trust region* [142].

A trust-region method works as follows: At any iteration, a model of the objective function is defined over a trust region of a certain radius. The trust region is centered on the current best-so-far solution P_k. A trial point s_k is then generated such that a new point $P_k + s_k$ sufficiently reduces the model, and it is still within the trust region. An evaluation of the objective function at $P_k + s_k$ is performed, and the value returned is compared to the prediction of the model. If the prediction is sufficiently accurate, the new point is accepted as the new best-so-far solution, and the next

iteration is executed. In the next iteration, the trust-region radius can be larger or equal to the one used in the previous iteration. However, if the prediction is not sufficiently accurate, the new point is rejected and the next iteration is executed with a reduced trust-region radius.

Existing trust-region methods differ in the way they define the model of the objective function, the criteria used to accept a new solution, and the way they update both the model and the trust-region radius. Here, we briefly describe a state-of-the-art trust-region method for unconstrained continuous optimization. This method is called NEWUOA (NEW Unconstrained Optimization Algorithm) [730]. For approximating an n-dimensional objective function, NEWUOA uses a quadratic interpolation of $\mathcal{O}(n)$ points within the trust region (a common value being $2n+1$ points). It is possible to use a linear number of interpolation points if the Hessian of the model changes as little as possible from one iteration to the next. Once the model is computed, it is minimized using a truncated conjugate gradient method [952]. The point that minimizes the model is accepted if some conditions on the accuracy of the interpolation and the trust region size are met. Besides the number of interpolation points, the initial and final trust region radii are parameters of the method.

Trust-region methods have been applied in the context of global optimization in [682, 687, 878, 956]

3.5.2.5 Other Methods

The family of methods described in the previous section have been the most common choice for performing local search in memetic and other hybrid local-global search continuous optimization algorithms. However, in principle, any method that explores a solution's neighborhood can be used as a local search mechanism. In fact, there are several hybrid algorithms proposed in the literature that use algorithms that can be used as global optimizers. Interestingly, the main distinctive feature of these mechanisms is that they belong to the class of stochastic local search methods (cf. Section 3.5.1).

Examples of this approach are Solis and Wets' minimization technique [834], which is used, for example, in [383, 528, 607]. The covariance matrix adaptation evolution strategy [361] (a state-of-the-art continuous optimization technique at the time of writing) has been used as a local search method in [607, 608, 643]. A particle swarm optimization algorithm [457] has been used with genetic algorithms [326] to refine elite solutions [436]. Simulated annealing [468] has been also used as a local search method in [658]. As a final example, we want to mention the simultaneous perturbation stochastic approximation method [838], which has also been used as a gradient-estimation local search method in [515].

Acknowledgements. The authors thank Thomas Stützle for his help in early drafts of this chapter. This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium, by Spanish MICINN under project NEMESIS (TIN2008-05941) and Junta de Andalucía under project TIC-6083, and by the Academy of Finland, Akatemiatutkija 130600, Algorithmic Design Issues in Memetic Computing.