

Chapter 5

Search Strategies

In the previous chapter, we discussed important elements of modern heuristics. We gave an overview of representations and search operators and discussed various aspects of the design of fitness functions and initialization methods. This section delivers the final design element and describes concepts for controlling the search. Different strategies for controlling search differ in the design and control of the intensification and diversification phases (see Sect. 3.4.3 and Blum and Roli (2003)). It is important for search strategies to balance intensification and diversification during search and to allow search methods to escape from local optima. This is achieved by various diversification techniques based on the representation, search operator, fitness function, initialization, or explicit diversification steps controlled by the search strategy.

In analogy to search operators, we distinguish between two fundamental concepts for heuristic search: local search methods versus recombination-based search methods. The choice between these two different concepts is problem-specific. If the problem at hand has high locality and the distances between solutions correspond to their fitness difference (Sect. 2.4.2), local search methods are the methods of choice. Then, the structure of the search space guides local search methods towards optimal solutions and local search outperforms random search (see the discussion in Sect. 3.4.4). The situation is slightly different for recombination-based search approaches, as these methods show good performance if the problem at hand is decomposable (Sect. 2.4.3). Then, the problem can be solved by decomposing the problem into smaller subproblems, solving these subproblems independently of each other, and determining the overall optimal solution by combining optimal solutions for the subproblems. As many real-world problems have high locality and are decomposable, both types of search methods often show good performance and are able to return high-quality solutions. However, direct comparisons between these two concepts are only meaningful for particular problem instances, and general statements on the superiority of one or other of these basic concepts are unjustified. Which of the two concepts is more appropriate for solving a particular problem instance depends on the specific characteristics of the problem (locality versus decomposability).

Therefore, comparing the performance of different search strategies by studying their performance for a set of predefined test functions and generalizing the results can be problematic (Thierens, 1999), as test functions are usually biased to favor one particular concept. For example, common test functions for recombination-based search strategies are usually decomposable. Representative examples for decomposable problems are the one-max problem (also known as the bit-counting problem), concatenated deceptive traps (Ackley, 1987; Deb and Goldberg, 1993a), or royal road functions (Mitchell et al, 1992; Jansen and Wegener, 2005). For such types of test problems, recombination-based search approaches often show better performance than local search approaches. In contrast, problems that are commonly used as test problems for local search methods often show a high degree of locality (e.g. the corridor model (5.1), sphere model (5.2), or the Rosenbrock function (3.1)) resulting in high performance of local search methods.

This chapter starts with a discussion on how different search strategies ensure diversification in the search. Diversification can be introduced into search by a proper design of a representation or operator, a fitness function, initial solutions, or an explicit control of the search strategy. Section 5.1 gives an overview of representative local search approaches that follow these principles. We describe variable neighborhood search, guided local search, iterated local search, simulated annealing, Tabu search, and evolution strategies. We especially discuss how the different approaches balance diversification and intensification. Section 5.2 focuses on recombination-based search methods and discusses representative approaches like genetic algorithms, estimation of distribution algorithms, and genetic programming. Again, we study intensifying and diversifying elements of the search strategies.

5.1 Local Search Methods

The idea of local search is to iteratively create neighboring solutions. Since such strategies usually consider only one solution, recombination operators are not meaningful. For the design of efficient local search methods it is important to incorporate intensification as well as diversification phases into the search.

Local search methods are also called *trajectory methods* since the search process can be described as a trajectory in the search space. The search space is a result of the interplay between representation and operator. A trajectory depends on the initial solution, the fitness function, the representation/operator combination, and the search strategy used. The behavior and the dynamics of local as well as recombination-based search methods can be described using *Markov processes* and concepts of statistical mechanics (Rudolph, 1996; Vose, 1999; Reeves and Rowe, 2003). In Markov processes, states are used which represent the subsequently generated solutions (or populations of solutions). A search step transforms one state into a following state. The behavior of search algorithms can be analyzed by studying possible sequences of states and their corresponding probabilities. The transition

matrix describes how states depend on each other and depends on the initial solution, fitness function, representation/operator combination, and search strategy.

Existing local as well as recombination-based search strategies mainly differ in how they control diversification and intensification. Diversification is usually achieved by applying variation operators or making larger modifications of solutions. Intensification steps use the fitness of solutions to control search and usually ensure that the search moves in the direction of solutions with higher fitness. The goal is to find a trajectory that overcomes local optima by using diversification and ends in a global optimal solution.

In greedy search approaches (like the best-first search strategy discussed in Sect. 3.3.2), intensification is maximal as in each search step the neighboring solution with highest fitness is chosen. No diversification is possible and the search stops at the nearest local optimum. Therefore, greedy search finds the global optimum if we have an unimodal problem where only one local optimum exists. However, as problems usually have a larger number of local optima, the probability of finding the global optimum using greedy search is low.

Based on the design elements of modern heuristics, there are different strategies to introduce diversification into the search and to escape from local optima:

- **Representation and search operator:** Choosing a combination of representation and search operators is equivalent to defining a metric on the search space and defines which solutions are neighbors. By using different types of neighborhoods, it is possible to escape from local optima and explore larger areas of the search space. Different neighborhoods can be the result of different genotype-phenotype mappings or search operators applied during search. Standard examples for local search approaches that use modifications of representations or operators to diversify the search are variable neighborhood search (Hansen and Mladenović, 2001), problem space search (Storer et al, 1992) (see also Sect. 6.2.1), the rollout algorithm (Bertsekas et al, 1997), and the pilot method (Duin and Voß, 1999).
- **Fitness function:** The fitness function measures the quality of solutions. Modifying the fitness function has the same effect as changing the representation as it assigns different fitness values to the problem solutions. Therefore, variations and modifications of the fitness function lead to increased diversification in local search approaches. A common example is guided local search (Voudouris, 1997; Balas and Vazacopoulos, 1998) which systematically changes the fitness function with respect to the progress of search.
- **Initial solution:** As the search trajectory depends on the choice of the initial solution (for example, greedy search always finds the nearest local optimum), we can introduce diversification by performing repeated runs of search heuristics using different initial solutions. Such multi-start search approaches allow us to explore a larger area of the search space and lead to higher diversification. Variants of multi-start approaches include iterated descent (Baum, 1986a,b), large-step Markov chains (Martin et al, 1991), iterated Lin-Kernighan (Johnson, 1990), chained local optimization (Martin and Otto, 1996), and iterated local search (Lourenco et al, 2001).

- Search strategy: An important element of modern heuristics are intensification steps like those performed in local search that push the search towards high-quality solutions. To avoid “pure” local search ending in the nearest local optimum, diversification steps are necessary. The search strategy can control the sequence of diversification and intensification steps. Diversification steps that do not move towards solutions with higher quality can either be the results of random, larger, search steps or based on information gained in previous search steps. Examples of search strategies that use a controlled number of search steps towards solutions of lower quality to increase diversity are simulated annealing (Aarts and van Laarhoven, 1985; van Laarhoven and Aarts, 1988) (see Sect. 3.4.3), threshold accepting (Dueck and Scheuer, 1990), or stochastic local search (Gu, 1992; Selman et al, 1992; Hoos and Stützle, 2004). Representative examples of search strategies that consider previous search steps for diversification are tabu search (Glover, 1986; Glover and Laguna, 1997) or adaptive memory programming (Taillard et al, 2001).

The following paragraphs discuss the functionality and properties of selected local search strategies. The different examples illustrate the four different approaches (representation/operator, fitness function, initial solution, search strategy) to introduce diversity into search. The section ends with a discussion of evolution strategies which are representative examples of local search approaches that use a population of solutions. In principle, all local search concepts that have been developed for a single solution can be extended to use a population of solutions.

5.1.1 Variable Neighborhood Search

Variable Neighborhood Search (VNS) (Mladenovic and Hansen, 1997; Hansen and Mladenović, 2001) combines local search strategies with dynamic neighborhood structures that are changed subject to the progress made during search. VNS is based on the following observations (Hansen and Mladenović, 2003):

- A local minimum with respect to a neighborhood structure is not necessarily a local optimum with respect to a different neighborhood (see also Sects. 2.3.2 and 4.2.2). The neighborhood structure of the search space depends on the metric used and is different for different search operators and representations. This observation goes back to earlier work (Liepins and Vose, 1990; Jones, 1995b,a) which found that different types of operators result in different fitness landscapes.
- A global minimum is a global minimum with respect to all possible neighborhood structures. Different neighborhood structures only result in different similarity definitions but do not change the fitness of the solutions. Therefore, the global optimum is independent of the search operators used and remains the global optimum for all possible metrics.
- Hansen and Mladenović (2003) conjecture that in many real-world problems, local optima with respect to different neighborhood structures have low distance to

each other and local optima have some properties that are also relevant for the global optimum. This observation is related to the decomposability of problems which is relevant for recombination-based search. Local optima are not randomly distributed in the search space but local optima already contain some optimal solutions to subproblems. Therefore, as they share common properties, the average distance between local optima is low.

Figure 5.1 illustrates the basic idea of VNS. The goal is to repeatedly perform a local search using different neighborhoods N . The global optimum x^* remains the global optimum with respect to all possible neighborhoods. However, as different neighborhoods result in different neighbors, x can be a local optimum with respect to neighborhood N_1 but it is not necessarily a local optimum with respect to N_2 . Thus, performing a local search starting from x and using N_2 can find the global optimum.

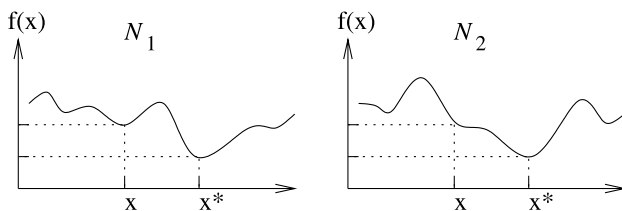


Fig. 5.1 Changing the neighborhood from N_1 to N_2 allows local search to find the global optimum

The functionality of VNS is described in Algorithm 2. During initialization, a set of k different neighborhoods is defined. A neighborhood function $N(x) : X \rightarrow 2^X$ (2.9) describes which solutions are neighbors of $x \in X$. It is based on a metric and its cardinality $|N(x)|$ is the (average) number of neighbors of $x \in X$. For VNS, it is important to choose neighborhoods with an appropriate cardinality. At the extreme, if the search space is fully connected and the cardinality of a neighborhood is similar to the problem size (for example $|N(x)| = |X| - 1$), all solutions are neighboring solutions and can be reached in one search step. Then, guided search is not possible any more as we have no meaningful metric to measure similarities between solutions (we have a trivial topology, see Sect. 2.3.1). In principle, the set of k different neighborhoods can be arbitrarily chosen, but often a sequence $|N_1| < |N_2| < \dots < |N_{k_{\max}}|$ of neighborhoods with increasing cardinality is used. VNS iteratively performs a “shaking phase” and a local search phase. During shaking, a solution x' in the k th neighborhood of the current solution x is randomly selected. x' is generated at random to avoid cycling and to lead local search towards a new local optimum different from x . During local search, we start with x' and perform a local search until a local optimum x'' is found. If x'' is better than x , it replaces x and the algorithm starts anew from this solution using the first neighborhood N_1 . Otherwise, we continue with shaking.

Algorithm 2 Variable Neighborhood Search

```

Select a set of neighborhood structures  $N_k, k \in \{1, \dots, k_{max}\}$ 
Create initial solution  $x$ 
while termination criterion is not met do
     $k = 1$ 
    while  $k < k_{max}$  do
        Shaking: choose a random neighbor  $x' \in N_k(x)$ 
        Local search: perform a local search starting with  $x'$  and return  $x''$  as the local optimum
        with respect to  $N_k$ 
        if  $f(x'') < f(x)$  (minimization problem) then
             $x = x''$ 
             $k = 1$ 
        else
             $k = k + 1$ 
        end if
    end while
end while

```

VNS contains intensification and diversification elements. The local search focuses search as it searches in the direction of high-quality solutions. Diversification is a result of changing neighborhoods as a solution x is not necessarily locally optimal with respect to a different neighborhood. Therefore, by changing neighborhoods, VNS can easily escape from local optima. Furthermore, due to the increasing cardinality of the neighborhoods (the neighborhoods are ordered with respect to their cardinality), diversification gets stronger as the shaking steps can choose from a larger set of solutions and local search covers a larger area of the search space (the *basin of attraction* increases).

Although in the last few years VNS has become quite popular and many publications have shown successful applications (for an overview see Hansen and Mladenović (2003)), the underlying ideas are older and more general. The goal is to introduce diversification into modern heuristics by changing the metric of the problem with respect to the progress that is made during search. A change in the metric can be achieved either by using different search operators or a different genotype-phenotype mapping. Both lead to different metrics and neighborhoods. Early ideas on varying the representation (adaptive representations) with respect to the search progress go back to Holland (1975). First implementations have been presented by Grefenstette et al (1985), Shaefer (1987), Schraudolph and Belew (1992), and Storer et al (1992) (see also the discussion in Sect. 6.2.1). Other examples are approaches that use additional transformations (Sebald and Chellapilla, 1998b), a set of pre-selected representations (Sebald and Chellapilla, 1998a), or multiple and evolvable representations (Liepins and Vose, 1990; Schnier, 1998; Schnier and Yao, 2000).

5.1.2 Guided Local Search

A fitness landscape is the result of the interplay between a metric that defines similarities between solutions and a fitness function that assigns a fitness value to each solution. VNS uses modifications of the metric to create different fitness landscapes and to introduce diversification into the search process. *Guided local search* (GLS) (Voudouris and Tsang, 1995; Voudouris, 1997) uses a similar principle and dynamically changes the fitness landscape subject to the progress that is made during the search. In GLS, the neighborhood structure remains constant. Instead, it dynamically modifies the fitness of solutions near local optima so that local search can escape from local optima.

GLS considers problem-specific knowledge by using the concept of *solution features*. A solution feature can be any property or characteristics that can be used to distinguish high-quality from low-quality solutions. Examples of solution features are edges used in a tree or graph, city pairs (for the TSP), or the number of unsatisfied clauses (for the SAT problem; see p. 126). The indicator function $I_i(x)$ indicates whether a solution feature $i \in \{1, \dots, M\}$ is present in solution x . For $I_i(x) = 1$, solution feature i is present in x , for $I_i(x) = 0$ it is not present. GLS modifies the fitness function f such that the fitness of solutions with solution features that exist in many local optimal solutions is reduced. For a minimization problem, $f(x)$ is modified to yield a new fitness function

$$f'(x) = f(x) + \lambda \sum_{i=1}^M p_i I_i(x),$$

with the *regularization parameter* λ and the *penalty parameters* p_i . The p_i are initialized as $p_i = 0$. M denotes the number of solution features. λ weights the impact of the solution features on the original fitness function f and p_i balances the impact of solution features of different importance.

Algorithm 3 describes the functionality of GLS. It starts from a random solution x_0 and performs a local search returning the local optimum x_1 . To escape the local optimum, a penalty is added to the fitness function f such that the resulting fitness function h allows local search to escape. The strength of the penalty depends on the utility u_i which is calculated for all solution features $i \in \{1, \dots, M\}$ as

$$u_i(x_1) = I_i(x_1) \times c_i / (1 + p_i),$$

where c_i is the cost of solution feature i . The c_i are problem-specific and usually remain unchanged during search. They are determined by the user and describe the relative importance of the solution features. Examples of c_i are the weights of edges (graph or tree problems) or the city-pair distances (TSP). The function $u_i(x)$ is unequal to zero for all solution features that are present in x . After calculating the utilities, the penalty parameters p_i are increased for those solution features i that yield the highest utility value. After modifying the fitness function, we start a new

local search from x_1 using the modified fitness function h . Search continues until a termination criterion is met.

Algorithm 3 Guided Local Search

```

 $k = 0$ 
Create initial solution  $x_0$ 
for  $i = 1$  to  $M$  do
   $p_i = 0$ 
end for
while termination criterion is not met do
   $h = f + \lambda \sum_{i=1}^M p_i I_i$ 
  perform a local search using fitness function  $h$  starting with  $x_k$  and return the local optimum  $x_{k+1}$ 
  for  $i = 1$  to  $M$  do
     $u_i(x_{k+1}) = I_i(x_{k+1}) \times c_i / (1 + p_i)$ 
  end for
  for all  $i$  where  $u_i$  is maximum do
     $p_i = p_i + 1$ 
  end for
   $k = k + 1$ 
end while
return  $x^*$  (best solution found so far according to  $f$ )
  
```

The utility function u penalizes solution features i with high cost c_i and allows us to consider problem-specific knowledge by choosing appropriate values for c_i . The presence of a solution feature with high cost leads to a high fitness value of the corresponding solution allowing local search to escape from this local optimum. Figure 5.2 illustrates the idea of changing the fitness of local optima. By modifying the fitness function $f(x)$ and adding a penalty to $f(x)$, $h(x)$ assigns a lower fitness to x . Thus, local search can leave x and is able to find the global optimum x^* .

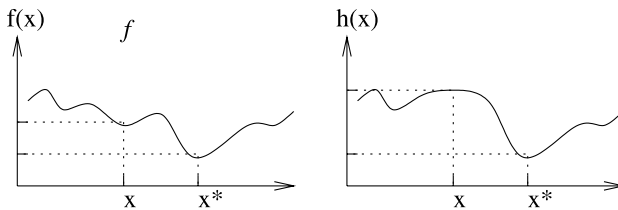


Fig. 5.2 Changing the fitness function from f to h allows local search to find the global optimum

In GLS, diversification is a result of the systematic modification of the fitness function f . The intensity of diversification is controlled by the parameter λ . Large values of λ lead to local search steps that find solution features that were not present in previous solutions. However, if λ is too large, GLS behaves like random search and randomly moves through the search space. Problems can also occur if λ is too low as no information about previous search steps can be considered for the fitness

function $h(x)$ and local search repeatedly finds the same local optimum (Mills and Tsang, 2000). In general, the setting of λ is problem-specific and must be done with care.

Examples of the successful application of GLS are TSPs (Voudouris and Tsang, 1999), bin-packing problems (Faroe et al, 2003b), VLSI design problems (Faroe et al, 2003a), and SAT problems (Mills and Tsang, 2000; Zhang, 2004).

5.1.3 Iterated Local Search

Heuristics that use only intensification steps (like local search) are often able to quickly find a local optimal solution but unfortunately cannot leave a local optimum again. A straightforward way to introduce diversification is to perform sequential local search runs using different initial solutions. Such approaches are commonly known as *multi-start approaches*. The simplest variants of multi-start approaches iteratively generate random solutions and perform local search runs starting from those randomly generated solutions. Thus, we have distinct diversification phases and can explore larger areas of the search space. Search strategies that randomly generate initial solutions and perform a local search are also called *multi-start descent* search methods.

However, to randomly create an initial solution and perform a local search often results in low solution quality as the complete search space is uniformly searched and search cannot focus on promising areas of the search space. *Iterated local search* (ILS) (Martin et al, 1991; Stützle, 1999; Lourenco et al, 2001) is an approach to connect the unrelated local search phases as it creates initial solutions not randomly but based on solutions found in previous local search runs. Therefore, it is based on the same observations as VNS which assumes that local optima are not uniformly distributed in the search space but similar to each other (Sect. 5.1.1, p. 134).

Algorithm 4 outlines the basic functionality of ILS. Relevant design criteria for ILS are the modification of x and the acceptance criterion. If the perturbation steps are too small, the following local search cannot escape from a local optimum and again finds the same local optimum. If perturbation is too strong, ILS shows the same behavior as multi-start descent search methods. The modification step as well as the acceptance criterion can depend on the search history.

Algorithm 4 Iterated Local Search

```

Create initial solution  $x_0$ 
Perform a local search starting with  $x_0$  and return the local optimum  $x$ 
while termination criterion is not met do
    Modification: perturb  $x$  and return  $x'$ 
    Perform a local search starting with  $x'$  and return the local optimum  $x''$ 
    Acceptance Criterion: decide whether to continue with  $x$  or with  $x''$ 
end while

```

In ILS, diversification is controlled by the perturbation of the solution (which is problem-specific) and the acceptance criterion. Larger perturbations and continuing search with x'' lead to stronger diversification. Continuing search with x intensifies search as a previously used initial solution is re-used.

A similar concept is used by *greedy randomized adaptive search* (GRASP) (Feo et al, 1994; Feo and Resende, 1995). Like in ILS, each GRASP iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is investigated until a local minimum is found during the local search phase.

5.1.4 Simulated Annealing and Tabu Search

The previous examples illustrated how modern heuristics can make diversification steps by modifying the representation/search operator, fitness function, or initial solutions. Simulated annealing (SA) is a representative example of a modern heuristic where the search strategy used explicitly defines intensification and diversification phases/steps. The functionality and properties of SA are discussed in detail in Sect. 3.4.3 (pp. 94-97). Its functionality is outlined in Algorithm 1, p. 96.

SA is a combination between a random walk through the search space and local search. Diversification of search is a result of the random walk process and intensification is due to the local search steps. The amount of intensification and diversification is controlled by the parameter T (see Fig. 3.21). With lower temperature T , intensification becomes stronger as solutions with lower fitness are accepted with lower probability. The cooling schedule which determines how T is changed during an SA run is designed such that at the beginning of the search diversification is high whereas at the end of a run pure local search steps are used to find a local optimum.

Tabu search (TS) (Glover, 1977, 1986; Glover and Laguna, 1997) is a popular modern heuristic. To diversify search and to escape from local optima, TS uses a list of previously visited solutions. A simple TS strategy combines such a *short term memory* (implemented as a *tabu list*) with local search mechanisms. New solutions that are created by local search are added to this list and older solutions are removed after some search steps. Furthermore, local search can only create new solutions that do not exist in the tabu list T . To avoid memory problems, usually the length of the tabu list is limited and older solutions are removed. Algorithm 5 outlines the basic functionality of a simple TS. It uses a greedy search, which evaluates all neighboring solutions $N(x)$ of x and continues with a solution x' that is not contained in T and has maximum fitness. Instead of removing the oldest element x_{oldest} from T , also other strategies for updating the tabu list are possible.

The purpose of the tabu list is to allow local search to escape from local optima. By prohibiting previously found solutions, new solutions must be explored. Figure 5.3 illustrates how the tabu list T contains all solutions of high fitness that are in the neighborhood of x . Therefore, new solutions with lower fitness are created and local search is able to find the global optimum.

Algorithm 5 Simple Tabu Search

```

Create initial solution  $x$ 
Create empty tabu list  $T$ 
while termination criterion is not met do
   $x' = \max_{N(x)-T} f(x)$  (Choose  $x'$  as neighboring solution of  $x$  with highest fitness that is not con-
    tained in  $T$ )
   $T = T + \{x'\}$ 
  if  $|T| > l$  then
    Remove oldest element  $x_{oldest}$  from  $T$ 
  end if
   $x := x'$ 
end while

```

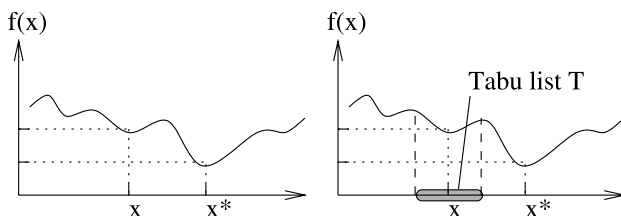


Fig. 5.3 Removing solutions $x \in T$ from the search space allows TS to escape from a local optimum

Diversification can be controlled by the length of the tabu list. Low values lead to stronger intensification as the memory of the search process is smaller. In many applications, the length of the tabu list is dynamically changed and depends on the progress made during search (Glover, 1990; Taillard, 1991; Battiti and Tecchiolli, 1994; Battiti and Protasi, 2001). Such approaches are also known as *reactive Tabu search*.

More advanced TS approaches do not store complete solutions in the tabu list but *solution attributes*. Similarly to the solution features used in GLS, solution attributes can be not only components of solutions but also search steps or differences between solutions. When using different solution attributes, we obtain a set of tabu criteria (called *tabu conditions*) which are used to filter the neighborhood of a solution. To overcome the problem that single tabu conditions prohibit the creation of appropriate solutions (each attribute prohibits the creation of a set of solutions), *aspiration criteria* are used which overwrite tabu conditions and allow search steps to create solutions where some tabu conditions are present. A commonly used aspiration criterion selects solutions with higher fitness than the current best one. For more information on TS and on application examples, we refer to the literature (Glover and Laguna, 1997; Gendreau, 2003; Voß et al, 1999; Osman and Kelly, 1996).

5.1.5 Evolution Strategy

Evolution strategies (ES) are local search methods for continuous search spaces. They were developed by Rechenberg and Schwefel in the 1960s at the Technical University of Berlin (Rechenberg, 1973a,b) and first applications were experimental and dealt with hydrodynamical problems like shape optimization of a bent pipe, drag minimization of a joint plate (Rechenberg, 1965), or structure optimization of a two-phase flashing nozzle (Schwefel, 1968). There are two different types of ES: Simple strategies that use only one individual (like the local search strategies discussed in the previous paragraphs) and advanced strategies that use a set of solutions which is called a population. The use of a population allows ES to exchange information between solutions in the population.

The simple $(1 + 1)$ -ES uses n -dimensional continuous vectors and iteratively creates one offspring $x' \in \mathbb{R}^n$ from one parent $x \in \mathbb{R}^n$ by adding randomly created values with zero mean and identical standard deviations σ to each parental decision variable x_i .

$$x'_i = x_i + \sigma \mathcal{N}_i(0, 1),$$

where $i \in \{1, \dots, n\}$. In ES and other biology-inspired search methods, a search step is usually denoted as a *mutation*. $\mathcal{N}(0, 1)$ is a normally distributed one-dimensional random variable with expectation zero and standard deviation one. $\mathcal{N}_i(0, 1)$ indicates that the random variable is sampled anew for each possible value of the counter i . In the $(1 + 1)$ -ES, the resulting individual is evaluated and compared to the original solution. The better one survives to be used for the creation of the next solution. Algorithm 6 summarizes the basic functionality (maximization problem).

Algorithm 6 $(1 + 1)$ -Evolution Strategy

```

Create initial solution  $x$ 
while termination criterion is not met do
  for  $i = 1$  to  $n$  do
     $x'_i = x_i + \sigma \mathcal{N}_i(0, 1)$ 
  end for
  if  $f(x') \geq f(x)$  then
     $x := x'$ 
  end if
  Update  $\sigma$  (e.g. according to 1/5-rule)
end while

```

For the $(1 + 1)$ -ES, theoretical convergence models for two simple problems, the *sphere model* and the *corridor model*, exist (Rechenberg, 1973a). Both problems are standard test problems for continuous local search methods. The corridor model is representative of the situation where the current solution x has a large distance to the optimum x^* :

$$f_{\text{corridor}}(x) = c_0 + c_1 x_1 \quad \forall i \in \{2, \dots, n\} : -b/2 \leq x_i \leq b/2 \quad (5.1)$$

and the sphere model describes the situation near the optimal solution x^* :

$$f_{sphere}(x) = c_0 + c_1 \sum_{i=1}^n (x_i - x_i^*)^2, \quad (5.2)$$

where c_0 , c_1 , and b are problem parameters.

$\zeta(t)$, where t indicates the number of search steps, is defined as the ratio of successful search steps that find a better solution to all search steps (for example, averaged over the last $10n$ search steps). For the corridor and sphere models, a ratio $\zeta(t) = \frac{1}{5}$ leads to maximal convergence speed (Rechenberg, 1973a). Therefore, if $\zeta(t)$ is greater than $\frac{1}{5}$, the standard deviation should be increased, if it is smaller, it should be decreased (Rechenberg, 1973a, p. 123). For sphere and corridor models this 1/5-rule results in fastest convergence. However, sometimes the probability of success cannot exceed 1/5. For problems where the fitness function has discontinuous first partial derivatives, or at the edge of the allowed search space, the 1/5-success rule does not work properly. Especially in the latter case, the success rule progressively forces the sequence of new solutions nearer to the boundary and continuously reduces the step length without approaching the optimum with comparable accuracy (Schwefel, 1995).

In $(1+1)$ -ES, the strength of diversification is controlled by the standard deviation σ . With increasing σ , step size increases and new solutions are less similar to the original solutions. If σ is too large, subsequently generated solutions are not related to each other and we obtain a random search behavior. The 1/5-rule adjusts σ and ensures that it becomes small enough to generate similar solutions. However, in contrast to a fixed σ , the 1/5-rule does not allow $(1+1)$ -ES to escape from local optima. Therefore, the 1/5-rule is not appropriate for multi-modal problems with more than one local optimum as it focuses the search on promising areas of the search space by reducing the step size. The second intensifying element in $(1+1)$ -ES is the selection process which continues with the better solution. This process is equivalent to local search.

Schwefel (1975) introduced the principle of a population into ES and proposed the $(\mu + \lambda)$ -ES and the (μ, λ) -ES to overcome problems of the $(1+1)$ -ES in escaping from local optima. In population-based ES approaches, a set of μ solutions (usually called *individuals*) forms a parent population. In each ES iteration (called a *generation*) a set of λ new solutions (offspring population) is created. For the $(\mu + \lambda)$ -ES, the next parent population is created by choosing the μ best individuals from the union of the parent and offspring population (“+”-selection). In the case of the (μ, λ) -ES, where $\lambda > \mu$, only the best μ solutions from the offspring population are chosen (“,”-selection). Both population-based ES start with a parent population of μ randomly generated individuals. Usually, each individual consists of an n -dimensional vector $x \in \mathbb{R}^n$ and an m -dimensional vector of standard deviations $\sigma \in \mathbb{R}_+^m$ (Bäck, 1996).

To exchange information between solutions, recombination operators are used in population-based ES as background search operators. Mostly, discrete crossover is used for creating the decision variables x'_i and intermediate crossover is used

for the standard deviations σ'_i . For discrete crossover, x'_i is randomly taken from one parent, whereas intermediate crossover creates σ'_i as the arithmetic mean of the parents' standard deviations (see Sect. 4.3).

The main search operator in ES is mutation. It is applied to every individual after recombination:

$$\sigma'_k = \sigma_k \exp(\tau_1 N(0, 1) + \tau_2 N_k(0, 1)) \quad \forall k = 1, 2, \dots, m, \quad (5.3)$$

$$x'_i = x_i + \sigma'_i \mathcal{N}_i(0, 1) \quad \forall i = 1, 2, \dots, n, \quad (5.4)$$

where τ_1 and τ_2 are method-specific parameters and usually $m = n$. If $m \neq n$, some standard deviations σ'_i are used for more than one decision variable. The standard deviations σ_k are mutated using a multiplicative, logarithmic normally distributed process with the factors τ_1 and τ_2 . Then, the decision variables x_i are mutated by using the modified σ'_k . This mutation mechanism enables the ES to evolve its own strategy parameters during the search. The logarithmic normal distribution is motivated as follows. A multiplicative modification process for the standard deviations guarantees positive values for σ and smaller modifications must occur more often than larger ones (Schwefel, 1977). Because the factors τ_1 and τ_2 are robust parameters, Schwefel (1977) suggests setting them as follows: $\tau_1 \propto (\sqrt{2\sqrt{n}})^{-1}$ and $\tau_2 \propto (\sqrt{2n})^{-1}$. Newer investigations indicate that optimal adjustments are in the interval $[0.1, 0.2]$ (Kursawe, 1999; Nissen, 1997). τ_1 and τ_2 can be interpreted as “learning rates” as in artificial neural networks, and experimental results indicate that the search process can be tuned for particular objective functions by modifying τ_1 and τ_2 .

One of the major advantages of ES is its ability to incorporate the most important parameters of the strategy, e.g. standard deviations, into the search process. Therefore, optimization not only takes place on object variables, but also on strategy parameters according to the actual local topology of the fitness function. This capability is called *self-adaption*.

In population-based ES, intensification is a result of the selection mechanism which prefers high-quality solutions. Recombination is a background operator which has both diversifying and intensifying character. By recombining two solutions, new solutions are created which lead to a more diversified population. However, especially intermediate crossover leads to reduced diversity during an ES run since the population converges to the mean values. Like for $(1+1)$ -ES, the main source of diversification is mutation. With larger standard deviations σ_i , diversification gets stronger as the step size increases. The balance between diversification and intensification is maintained by the self-adaptation of the strategy parameters. Solutions with standard deviations that result in high-quality solutions stay in the population, whereas solutions with inappropriate σ_i are removed by the selection process.

$(\mu + \lambda)$ -ES and (μ, λ) -ES are examples of local search strategies that use a population of solutions. Although local search is the main search strategy, the existence of a population allows ES to use recombination operators which exchange information between different solutions. For further information and applications of ES

we refer to the literature (Schwefel, 1981; Rechenberg, 1994; Schwefel, 1995; Bäck and Schwefel, 1995; Bäck, 1996, 1998; Beyer and Deb, 2001; Beyer and Schwefel, 2002).

5.2 Recombination-Based Search Methods

In contrast to local search approaches which exploit the locality of problems and show good performance for problems with high locality, recombination-based approaches make use of the decomposability of problems and perform well for decomposable problems. Recombination-based search solves decomposable problems by decomposing them into smaller subproblems, solving those smaller subproblems separately, and combining the resulting solutions for the subproblems to form overall solutions (Sect. 2.4.3). Hence, the main search operator used in recombination-based search methods is, of course, recombination.

Recombination operators should be designed such that they properly decompose the problem and combine high-quality sub-solutions in different ways (Goldberg, 1991a; Goldberg et al, 1992). A proper decomposition of problems is the key factor for the design of successful recombination operators. Recombination operators must be able to identify the relevant properties of a solution and transfer these as a whole to offspring solutions. In particular, they must detect between which parts of a solution a meaningful linkage exists and not destroy this linkage when creating an offspring solution. This property of recombination operators is often called *linkage learning* (Harik and Goldberg, 1996; Harik, 1997). If a problem is decomposable, the proper problem decomposition is the most demanding part as usually the smaller subproblems can be solved much more easily than the full problem. However, in reality, most often a problem is not completely separable but there exists still some linkage between different sub-problems. Therefore, usually it is not enough for recombination-based methods to be able to decompose the problem and solve the smaller sub-problems, but usually they must also try different combinations of high-quality sub-solutions to form an overall solution. This process of juxtaposing various high-quality sub-solutions to form different overall solutions is often called *mixing* (Goldberg et al, 1993b; Thierens, 1995; Sastry and Goldberg, 2002).

Recombination operators construct new solutions by recombining the properties of parent solutions. Therefore, recombination-based modern heuristics usually use a population of solutions since when using only single individuals, like in most local search approaches, no properties of different solutions can be recombined to form new solutions. Consequently, the main differences between local and recombination-based search are the use of a recombination operator and a population of solutions.

For local search approaches, we have been able to classify methods with respect to their main source of diversification. In principle, we can use the same mechanisms for recombination-based search methods, too. However, in most implementations of

recombination-based search methods, an initial population of diverse solutions is the only source of diversification. Usually, no additional explicit diversification mechanisms based on a systematic modification of representations/search operators or fitness functions are used. Search starts with a diversified population of solutions and intensification mechanisms iteratively intensify the search. Recombination-based search usually stops after the population has almost converged. This means that, at the end of the search, only little diversity in the population exists and all individuals are about the same. During a run, recombination (and most often also local search) operators are applied to parent solutions to create new solutions with similar properties. Usually, these variation operators do not increase the diversity in the population but create new solutions with similar properties.

When the population is almost homogeneous, the only source of diversification are small mutations which usually serve as background noise. Problems occur for recombination-based search if the population is almost converged (either to the correct optimal solution or a sub-optimal solution) but search continues. Then, recombination cannot work properly any more (since it needs a diversified population) and search relies solely on local search steps. However, local search steps are usually small and have only little effect. The situation that a population of solutions converges to a non-optimal solution and cannot escape from this local optimum is often called *premature convergence* (Goldberg, 1989c; Collins and Jefferson, 1991; Eshelman and Schaffer, 1991; Leung et al, 1997). In this situation and if no explicit diversification mechanisms are used, only local search and no recombination-based search is possible any more.

Possible strategies to overcome premature convergence and to keep or re-introduce diversification into a population could be based on the representation/operator, fitness function, initial solution, or search strategy and work in a similar way as described in the previous section on local search methods. A straightforward approach to increase diversity in the initial population is to increase the number of solutions in the population. Other approaches address operator aspects and prevent for example recombination between similar solutions in the population (Eshelman and Schaffer, 1991) or design operators which explicitly increase diversity (Bäck, 1996). Low-locality search operators and representations (Sect. 6.1) are other examples since such operators and representations randomize the search process and thus increase population diversity.

Many of the existing approaches limit the intensification strength of the *selection process*. Selection decides which solutions remain in the population and are used for future search steps. The selection process is equivalent to local search when using only one single solution. In the literature, approaches have been presented that decrease intensification by continuously reducing selection intensity during a run and not focusing only on better solutions but also accepting to some extent worse solutions (Bäck, 1996; Goldberg and Deb, 1991). Other approaches to reduce the intensifying character of the selection process limit the number of similar solutions in the population by either letting a new solution replace only a worse solution similar to itself (De Jong, 1975) or only adding a new solution to a population if it is entirely different (Whitley, 1989).

Finally, a substantial amount of work focuses on *niching methods* ((Deb and Goldberg, 1993b; Hocaoglu and Sanderson, 1995; Mahfoud, 1995; Horn and Goldberg, 1998). Niching methods deal with the simultaneous formation and evolution of different sub-populations in a population. The use of niching methods leads to higher diversification in a population as substantially different sub-populations exist. Niching methods are especially important for multi-objective optimization methods (Deb, 2001; Coello Coello et al, 2007) as such methods should return not only one solution but a set of different Pareto-optimal solutions.

The term *evolutionary algorithm* (EA) denotes optimization methods that are inspired by the principles of evolution and apply recombination, mutation, and selection operators to a population of solutions (for details on the operators, we refer to Sect. 5.2.1). Prominent examples of evolutionary algorithms are genetic algorithms (Sect. 5.2.1), evolution strategies (Sect. 5.1.5), genetic programming (Sect. 5.2.3), and *evolutionary programming* (Fogel et al, 1966; Fogel, 1999). Although the term evolutionary algorithm is commonly used in scientific literature, we do not follow this categorization since it classifies search methods based on their main source of inspiration (algorithms inspired by natural evolution) and not according to the underlying working principles. For example, genetic algorithms and genetic programming use recombination as main search operator, whereas evolution strategies and evolutionary programming use local search. Furthermore, genetic algorithms and *scatter search* (Glover, 1997; Laguna and Martí, 2003) share the same working principles (Glover, 1994), however, since scatter search is not inspired by evolution, it would not be ranked among evolutionary algorithms.

The following paragraphs present the simple genetic algorithm and two variants of it as representative examples of recombination-based search. The first variant, estimation of distribution algorithms, uses different types of search operators and the second one, genetic programming, uses solutions of variable length.

5.2.1 Genetic Algorithms

Genetic algorithms (GAs) were introduced by Holland (1975) and imitate basic principles of nature formulated by Darwin (1859) and Mendel (1866). They are (like population-based ES discussed on p. 142) based on three basic principles:

- There is a population of solutions. The properties of a solution are evaluated based on the phenotype, and variation operators are applied to the genotype. Some of the solutions are removed from the population if the population size exceeds an upper limit.
- Variation operators create new solutions with similar properties to existing solutions. In GAs, the main search operator is recombination and mutation serves as background operator. In ES, the situation is reversed.
- High-quality individuals are selected more often for reproduction by a selection process.

To illustrate the basic functionality of GAs, we want to use the standard *simple genetic algorithm* (SGA) popularized by Goldberg (1989c). This basic GA variant is commonly used and well understood and uses crossover as main operator (mutation serves only as background noise). SGAs use a constant population of size N , usually the N individuals $x^i \in \{0, 1\}^l$ ($i \in \{1, \dots, N\}$) are binary strings of length l , and recombination operators like uniform or n -point crossover are directly applied to the genotypes. The basic functionality of a SGA is shown in Algorithm 7. After randomly creating and evaluating an initial population, the algorithm iteratively creates new generations by recombining (with probability p_c) the selected highly fit individuals and applying mutation (with probability p_m) to the offspring. The function $\text{random}(0, 1)$ returns a uniformly distributed value in $[0, 1)$.

Algorithm 7 Simple Genetic Algorithm

```

Create initial population  $P$  with  $N$  solutions  $x^i$  ( $i \in \{1, \dots, N\}$ )
for  $i = 1$  to  $N$  do
    Calculate  $f(x^i)$ 
end for
while termination criterion is not met and population has not yet converged do
    Empty the mating pool:  $M = \{\}$ 
    Insert  $N$  individuals into  $M$  from  $P$  using a selection scheme
    Shuffle the position of all individuals in  $M$ 
     $\text{ind} = 1$ 
    repeat
        if  $\text{random}(0, 1) \leq p_c$  then
            Recombine  $x^{\text{ind}} \in M$  and  $x^{\text{ind}+1} \in M$  and place the resulting two offspring in  $P'$ 
        else
            Copy  $x^{\text{ind}} \in M$  and  $x^{\text{ind}+1} \in M$  to  $P'$ 
        end if
         $\text{ind} = \text{ind} + 2$ 
    until  $\text{ind} > N$ 
    for  $i = 1$  to  $N$  do
        for  $j = 1$  to  $l$  do
            if  $\text{random}(0, 1) \leq p_m$  then
                 $\text{mutate}(x_j^i)$ , where  $x^i \in P'$ 
            end if
        end for
        Calculate  $f(x^i)$ , where  $x^i \in P'$ 
    end for
     $P := P'$ 
end while
  
```

The following paragraphs briefly explain the basic elements of a GA. The selection process performed in population-based search approaches is equivalent to local search for single individuals as it distinguishes high-quality from low-quality solutions and selects promising solutions. Popular selection schemes are *proportionate selection* (Holland, 1975) and *tournament selection* (Goldberg et al, 1989). For proportionate selection, the expected number of copies a solution has in the next population is proportional to its fitness. The chance of a solution x^i being selected

for recombination is calculated as

$$\frac{f(x^i)}{\sum_{j=1}^N f(x^j)}.$$

With increasing fitness, an individual is chosen more often for reproduction.

When using tournament selection, a tournament between s randomly chosen different individuals is held and the one with the highest fitness is added to the mating pool M . After N tournaments of size s the mating pool is filled. We have to distinguish between tournament selection with and without replacement. Tournament selection with replacement chooses for every tournament s individuals from the population P . Then, M is filled after N tournaments. Tournament without replacement performs s rounds. In each round we have N/s tournaments and we choose the solutions for a tournament from those who have not already taken part in a tournament in this round. After all solutions have performed a tournament in one round (after N/s tournaments), the round is over and all $x \in P$ are considered again for the next round. Therefore, to completely fill the mating pool, s rounds are necessary. For more information concerning different selection schemes, see Bäck et al (1997, C2) and Sastry and Goldberg (2001).

The mating pool M consists of all solutions which are chosen for recombination. When using tournament selection, there are no copies of the worst solution, and either an average of s copies (with replacement), or exactly s copies (without replacement) of the best solution.

Crossover operators imitate the principle of sexual reproduction and are applied to all $x \in M$. Usually, crossover produces two new offspring from two parents by exchanging substrings. Common crossover operators are one-point crossover and uniform crossover (Sect. 4.3, p. 117-120). The mutation operator is a local search operator and slightly changes the genotype of a solution $x \in P'$. It is important for local search, or if some alleles are lost during a GA run. Mutation can reanimate solution properties that have previously been lost. The probability of mutation p_m must be selected to be at a low level because otherwise mutation would randomly change too many alleles and new solutions would have nothing in common with their parents. Offspring would be generated almost randomly and genetic search would degrade to random search. For details on mutation operators, see Sect. 4.3.2, p. 115-117.

In GAs, intensification is mainly a result of the selection scheme. In a selection step, the average fitness of a population increases as only high-quality solutions are chosen for the mating pool M . Due to selection, the population converges after a number of generations (for details on the convergence speed see Thierens (1995), Goldberg (2002), or Suzuki (1995)). Continuing recombination-based search after the population has converged (hopefully to the global optimum) makes no sense as diversity is minimal.

The main source of diversification is the initial population. Therefore, in GAs, often large population sizes of a few hundreds or even thousands of solutions are used (Goldberg, 2002). The effect of recombination operators is twofold: On the

one hand, recombination operators are able to create new solutions. On the other hand, usually recombination does not actively diversify the population but has an intensifying character. Crossover operators reduce diversity as the distances between offspring and parents are usually smaller than the distance between parents (see (4.1), p. 118). Therefore, the iterative application of crossover alone reduces the diversity of a population as either some solution properties can become extinct in the population (this effect is known as *genetic drift* (see p. 171) and especially relevant for binary solutions) or the decision variables converge to an average value (especially for continuous decision variables). The statistical properties of a population to which only recombination operators are applied, do not change (we have to consider that the aim of crossover is to re-combine different optimal solutions for subproblems) and, for example, the average fitness of a population remains constant (if we use a meaningful recombination operator and $N \rightarrow \infty$). However, diversity decreases as the solutions in the population become more similar to each other.

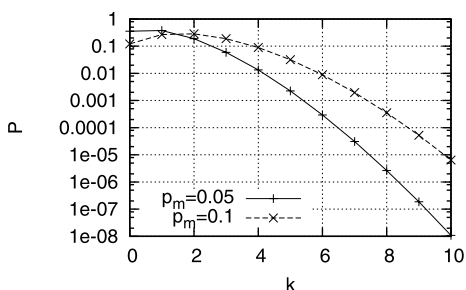


Fig. 5.4 Probability $P(k)$ of performing exactly k mutations versus k for $n = 20$ and various mutation rates p_m

In SGA, mutation has diversifying character. In contrast to many local search approaches where the neighborhood structure remains constant during search, the mutation operator used in SGA results in a varying neighborhood structure. Mutation does not generate only neighboring solutions with small distance but can reach all solutions in the search space in only one mutation step. When mutating one individual, mutation is iteratively applied to all l decision variables with probability p_m . Therefore, mutation can be modeled as a Bernoulli process and the probability of mutating exactly k decision variables can be calculated as $P(k) = \binom{n}{k} (p_m)^k (1 - p_m)^{n-k}$. Figure 5.4 plots $P(k)$ over k for $l = 20$ and $p_m = 0.05$ and $p_m = 0.1$. On average, $p_m l$ alleles are mutated and, for large values of p_m , the mutation operator can mutate all l decision variables and, thus, reach all possible points in the solution space. However, the probability of changing a large number of decision variables is low if p_m is low. The diversifying character of mutation increases with increasing p_m and for large p_m , SGA behaves like random search.

Further and more detailed information on functionality and application aspects of genetic algorithms can be found in standard textbooks like Goldberg (1989c), Mitchell (1996), Michalewicz (1996), Goldberg (2002), Michalewicz and Fogel (2004), Reeves and Rowe (2003), De Jong (2006), or Eiben and Smith (2010).

5.2.2 Estimation of Distribution Algorithms

Recombination operators should consider the linkage between different decision variables in a solution and transfer high-quality sub-solutions, which consist of several decision variables, as a whole from parents to offspring. However, the standard crossover operators like uniform or n -point crossover are position-dependent and thus sometimes of only limited use. n -point crossover can only work properly if the related decision variables are grouped together in the string. For example, for two-point crossover, sub-solutions for subproblems where the two decision variables x_0 and x_l are linked cannot be transferred as a whole from a parent to an offspring. In contrast, uniform crossover results in problems for large sub-problems as it disrupts such sub-solutions with high probability. For example, when using binary decision variables and assuming sub-problems of size k , uniform crossover disrupts sub-solutions with probability $1 - (1/2)^{k-1}$ ($k \geq 2$).

To overcome the shortcomings of traditional crossover methods, two different approaches have been proposed in the literature: the first line of research modifies the genotypes and the corresponding representation so that linked decision variables are grouped together in the genotype. Then, traditional recombination operators like n -point crossover show good performance as they do not not disrupt sub-solutions. Examples are the *fast messy GA* (Goldberg et al, 1993a) or the *gene expression messy GA* (Bandyopadhyay et al, 1998).

The second line of research designs recombination operators that are able to detect the linkage between decision variables and construct new offspring position-independently considering the linkage. To avoid disruptions of sub-solutions, recombination is replaced by generating new solutions according to the probability distribution of promising solutions of the previous generation. Algorithms that follow this concept are called *estimation of distribution algorithms* (EDA) and have been introduced by Mühlenbein and Paaß (1996). EDAs are based on the same concepts as GAs but replace variation operators like crossover and mutation by sampling new solutions from a probabilistic model which takes into account the problem-specific interactions among the decision variables. Linkage between decision variables can be expressed explicitly through joint probability distributions associated with the variables of the individuals. For intensification and generation of high-quality solutions, EDAs use the same selection methods as GAs. Algorithm 8 outlines the basic functionality of EDAs.

Critical for the performance of EDAs is the proper estimation of the probability distribution because this process decomposes the problem into sub-problems by detecting a possible linkage between decision variables. This process can only work properly if the problem at hand is decomposable. Therefore, EDAs show good performance for problems that are decomposable and where the sub-problems are of small size. As for GAs, non-decomposable problems cannot be solved effectively by EDAs.

Different EDA variants can be classified with respect to the type of decision variables (binary versus continuous) and the probability distribution which is used to describe the statistical properties of a population. Simple variants for discrete prob-

Algorithm 8 Estimation of Distribution Algorithm

```

Create initial population  $P$  with  $N$  solutions  $x^i$  ( $i \in \{1, \dots, N\}$ )
for  $i = 1$  to  $N$  do
    Calculate  $f(x^i)$ 
end for
while termination criterion is not met do
    Select  $M$  solutions, where  $M < N$ , from  $P$  using a selection scheme
    Estimate the (joint) probability distribution of all selected solutions
    Sample  $N$  solutions of the new population  $P'$  from the probability distribution
    Calculate  $f(x^i)$ , where  $x^i \in P'$ 
     $P = P'$ 
end while

```

lem domains are *population-based incremental learning* (PBIL) (Baluja, 1994), *univariate marginal distribution algorithm* (UMDA) (Mühlenbein and Paaß, 1996), and *competent GA* (cGA) (Harik et al, 1998). These algorithms assume that all variables are independent (which is an unrealistic assumption for most real-world problems) and calculate the probability of an individual as the product of the probabilities of every decision variable. More advanced approaches assume bivariate dependencies between decision variables. Examples are *mutual information maximization for input clustering* (MIMIC) (de Bonet et al, 1997), *combining optimizers with mutual information trees* (COMIT) (Baluja and Davies, 1997), *probabilistic incremental program evolution* (PIPE) (Salustowicz and Schmidhuber, 1997), and *bivariate marginal distribution algorithm* (BMDA) (Pelikan and Mühlenbein, 1999). The most complex EDAs that assume multivariate dependencies between decision variables are the *factorized distribution algorithm* (FDA) (Mühlenbein and Mahnig, 1999), the *extended compact GA* (Harik, 1999), the *polytree approximation of distribution algorithm* (PADA) (Soto et al, 1999), *estimation of Bayesian networks algorithm* (Ettxeberria and Larrañaga, 1999), and the *Bayesian optimization algorithm* (BOA) (Pelikan et al, 1999a). Although EDAs are still a young research field, EDAs using multivariate dependencies show promising results on binary problem domains and often outperform standard GAs (Larrañaga and Lozano, 2001; Pelikan, 2006). Their main advantage in comparison to standard crossover operators is the ability to learn the linkage between solution variables and the position-independent creation of new solutions.

The situation is different for continuous domains (Larrañaga et al, 1999; Bosman and Thierens, 2000; Gallagher and Freaan, 2005). Here, the probability distributions used (for example Gaussian) are often not able to model the structure of the landscape in an appropriate way (Bosman, 2003) leading to a low performance of EDAs for continuous search spaces (Grahl et al, 2005).

For more detailed information on the functionality and application of EDAs we refer to Larrañaga and Lozano (2001) and Pelikan (2006).

5.2.3 Genetic Programming

Genetic programming (GP) (Smith, 1980; Cramer, 1985; Koza, 1992) is a variant of GAs that evolves programs. Although most GP approaches use trees to represent programs (Koza, 1992, 1994; Koza et al, 1999, 2005; Banzhaf et al, 1997; Langdon and Poli, 2002), there are also a few approaches that encode programs using linear bitstrings (for example, *grammatical evolution* (Ryan, 1999; O'Neill and Ryan, 2003) or *Cartesian genetic programming* (Miller and Thomson, 2000)). The common feature of GP approaches is that the phenotypes are programs or variable-length structures like electronic circuits or controllers.

In analogy to GAs, GP starts with a population of random candidate programs. Each program is evaluated on a given task and its fitness value is assigned. Often, the fitness of an individual is determined by measuring how well the found solution (e.g. a computer program) performs a specific task. The basic functionality of GP follows GA functionality. The main differences are the search space which consists of tree structures of variable size and the corresponding search operators which have to be tree-specific. Solutions (programs) are usually represented as *parse trees*. Parse trees represent the syntactic structure of a string according to some formal grammar. In a parse tree, each node is either a root node, a branch node, or a leaf node. Interior nodes represent functions and leaf nodes represent variables, constants, or functions with no arguments. Therefore, the nodes in a parse tree are either members of

- the *terminal set* T (leaf nodes) representing independent variables of the problem, zero-argument functions, random constants, or terminals with side-effects (for example move operations like “turn-left” or “move forward”) or
- the *function set* F (interior nodes) representing functions (for example arithmetic or logical operations like “+”, \wedge , or \neg), control structures (for example “if” or “while” clauses), or functions with side-effects (for example “write to file” or “read”).

The definition of F and T is problem-specific and they should be designed such that

- each function is defined for each possible parameter. Parameters are either terminals or function returns.
- T and F must be chosen such that a parse tree can represent a solution for the problem. Solutions that cannot be constructed using the sets T and F are not elements of the search space and cannot be found by the search process.

In GP, the depth k and structure of a parse tree are variable. Figure 5.5 gives two examples of parse trees.

The functionality of GP is analogous to GAs (see Algorithm 7, p. 148). Different are the use of a direct representation (parse trees) and tree-specific initialization and variation operators. During initialization, we generate random parse trees of maximal depth k_{max} . There are three basic methods (Koza, 1992): *grow*, *full* and *ramped-half-and-half*. The grow method starts with an empty tree and iteratively assigns a node either to be a function or a terminal. If a node is a terminal, a random terminal from the terminal set T is chosen. If the node is a function, we choose a

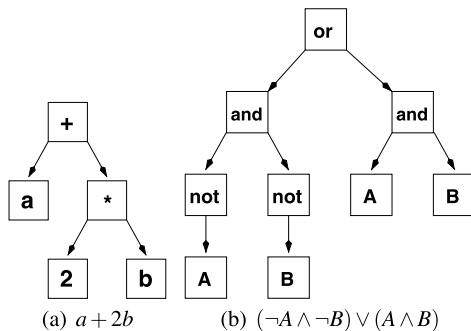


Fig. 5.5 Two examples of parse trees

random function from F . Furthermore, a number of child nodes are added such that their number equals the number of arguments necessary for the chosen function. The procedure stops either at depth $k = k_{\max}$ or when all leaf nodes are terminals. The full method also starts with an empty tree but all nodes at depth $k \in \{1, \dots, k_{\max} - 1\}$ are functions. All nodes at depth $k = k_{\max}$ become terminals. Therefore, all resulting trees have depth k_{\max} . For the ramped-half-and-half method, the population is evenly divided into $(k_{\max} - 1)$ parts. Half of each part is created by the grow method and half by the full method. For both halves, the depth of the nodes in the i th part is equal to i , where $i \in \{2, \dots, k_{\max}\}$. Thus, the diversity is high in the resulting initial population.

As no standard search operators can be applied to parse trees, tree-specific crossover and mutation operators are necessary. Like in SGAs, crossover is the main search operator and mutation acts as background noise. Crossover exchanges randomly selected sub-trees between two parse trees, whereas mutation usually replaces a randomly selected sub-tree by a randomly generated one. Like in GAs, standard GP crossover (Koza, 1992) chooses two parent solutions and generates two offspring by swapping sub-trees (see Fig. 5.6). Analogously, mutation chooses a random sub-tree and replaces it with a randomly generated new one (see Fig. 5.7). Other operators used in GP (Koza et al, 2005) are permutation, which changes the order of function parameters, editing, which replaces sub-trees by semantically simpler expressions, and encapsulation, which encodes a sub-tree as a more complex single node.

In recent years, GP has shown encouraging results finding programs or strategies for problem-solving (Koza et al, 2005; Kleinau and Thonemann, 2004). However, often the computational effort for finding high-quality solutions even for problems of small sizes is extremely high. Currently, open problems in GP are the low locality of the representation/operator combination (compare Chap. 7), the bias of the search operators (compare the discussion in Sect. 6.2.3, p. 171) and *bloat*. Bloat describes the problem that during a GP run, the average size of programs has been seen to grow large, sometimes exponentially. Although there is a substantial amount of work trying to fix problems with bloat (Nordin and Banzhaf, 1995; Langdon and Poli, 1997; Soule, 2002; Luke and Panait, 2006), there is no solution for this problem yet

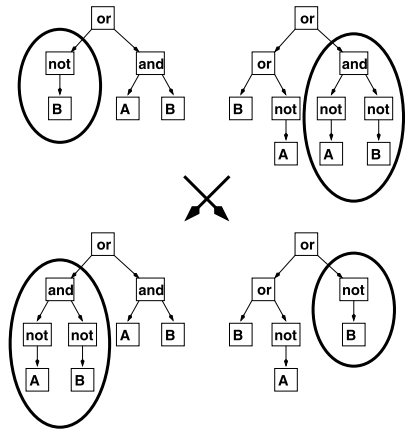


Fig. 5.6 Standard crossover operator for GP

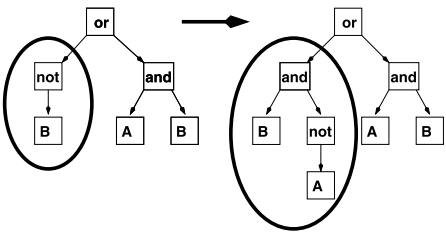


Fig. 5.7 Standard mutation operator for GP

(Banzhaf and Langdon, 2002; Gelly et al, 2005; Luke and Panait, 2006; Whigham and Dick, 2010).



<http://www.springer.com/978-3-540-72961-7>

Design of Modern Heuristics

Principles and Application

Rothlauf, F.

2011, XI, 267 p., Hardcover

ISBN: 978-3-540-72961-7