Eckart Zitzler
Computer Engineering and Communication Networks Lab (TIK)
Swiss Federal Institute of Technology (ETH) Zurich
Gloriastr. 35, CH–8092 Zürich, Switzerland
zitzler@tik.ee.ethz.ch

Abstract

This paper gives an introduction into evolutionary computation, in particular in the light of multiobjective optimization, and demonstrates how evolutionary algorithms can be used to tackle a highly demanding application in telecommunications, namely the design of a network processor.

1 Introductory Example

To illustrate the basic principles of multiobjective optimization and evolutionary algorithms, consider the following example: given is a set of items together with a profit and a weight associated with each item; the goal is to determine a subset of items such that the overall profit, i.e., the sum of the profits of the selected items, is maximum, while the overall weight, i.e., the sum of the weights of the selected items, is minimal. This problem is generally denoted as knapsack problem.

Now assume that four items are available: a camera (weight = 750, profit = 5), a thermos flask (weight = 1500, profit = 8), a pocket knife (weight = 300, profit = 7), and a book (weight = 1000, profit = 3). The set of all possible selections contains 16 possible subsets (cf. Fig. 1), in general 2^n where n is the number of items. In this context, two observations can be made: there is no single optimal selection of items, and some subsets are better than others. With respect to the first issue, the empty subset minimizes the overall weight, while the set containing all items maximizes the overall profit. We say the two optimization criteria are conflicting. Nevertheless, subsets for which there exists another subset that is better in at least one criterion, while not being worse in the other criterion, can be neglected. As a consequence, a set of optimal trade-offs emerges as shown in Fig. 1 at the bottom. At the end, though, we are interested in a single solution, and therefore a decision making process is necessary: which of the optimal trade-offs represents the best compromise for our needs?

In practice, the search for optimal solutions by using an appropriate optimization algorithm and the decision making process can be integrated in different ways. One possibility is to aggregate the multiple optimization criteria into a single one. That means the decision is made before the search. In our example, one could transform the second objective into a constraint and look for the selection with maximum profit that does not exceed a given weight bound. Alternatively, we can first search for all optimal trade-offs and then choose one solution out of them. In this case, decision making is done after the search, which is especially useful if little is known about the underlying problem.

Although being simple regarding the problem formulation, the above knapsack problem reflects two problem difficulties that arise in many real-world applications: i) the set of possible solutions is large, and ii) multiple, competing optimization criteria are involved. Thus, efficient search strategies are required that are able to deal with both difficulties.

Evolutionary algorithms possess several characteristics that are desirable in this context. The term evolutionary algorithm (EA) stands for a class of randomized search

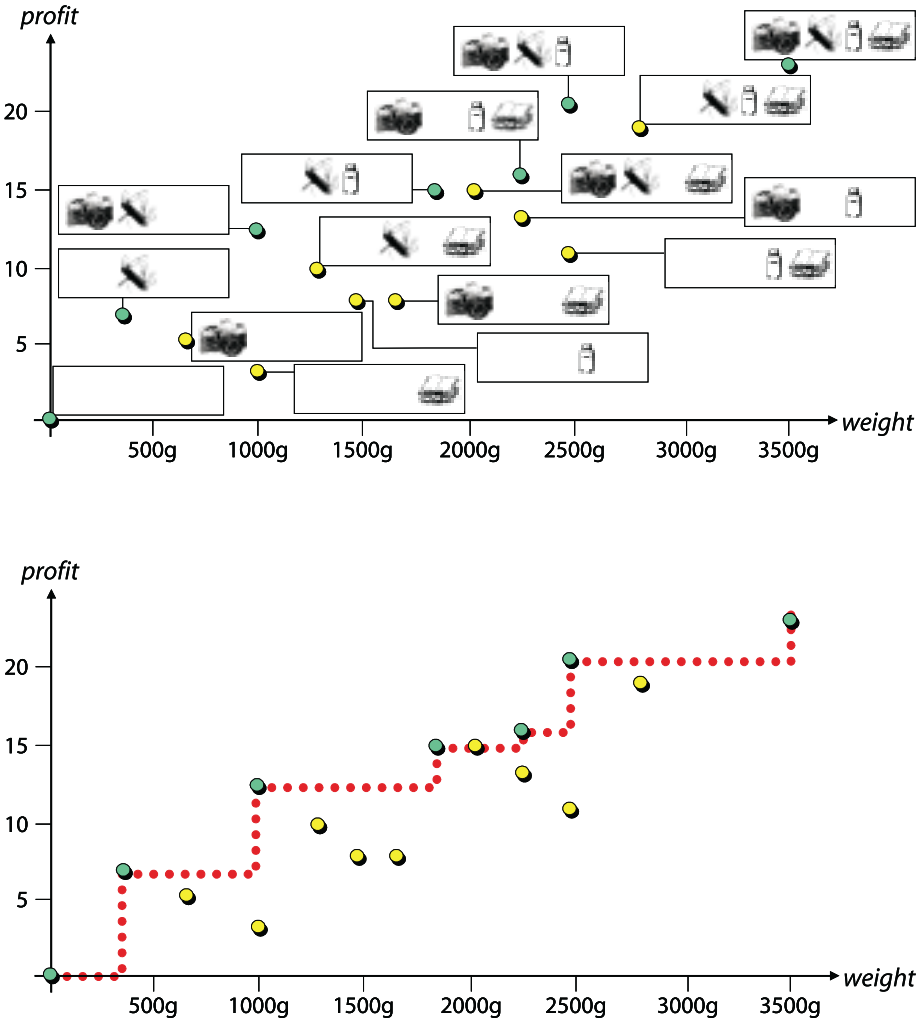

Figure 1: Illustration of the search space for a simple knapsack problem instance. At the bottom, the dark-shaeded solutions connected by the dotted line represent the optimal trade-offs.

strategies that simulate the process of natural evolution. The origins of EAs can be traced back to the late 1950s, and since the 1970s several evolutionary methodologies have been proposed, mainly genetic algorithms, evolutionary programming, and evolution strategies (Bäck, Hammel, and Schwefel 1997). All of these approaches operate on a set of candidate solutions. Using strong simplifications, this set is subsequently modified by two basic principles: selection and variation. While selection mimics the competition for reproduction and resources among living beings, the other principle, variation, imitates the natural capability of creating "new" living beings by means of recombination and mutation. Although the underlying mechanisms are simple, these algorithms have proven themselves as a general, robust and powerful search mechanism (Bäck, Hammel, and Schwefel 1997).

In this lecture, it will be discussed how evolutionary algorithms work, how they can be tailored to a problem at hand, and how they can be used to tackle a complex application in telecommunications, namely the design of a network processor.

2 Optimization and Randomized Search Algorithms

2.1 Basic Terms

The scenario considered in this paper involves an arbitrary optimization problem with k objectives, which are, without loss of generality, all to be maximized and all equally important, i.e., no additional knowledge about the problem is available. We assume that a solution to this problem can be described in terms of a *decision vector* $(x_1, x_2, \ldots, x_n)$ in the *decision space* $\boldsymbol{X}$. A function $\boldsymbol{f} : \boldsymbol{X} \rightarrow \boldsymbol{Y}$ evaluates the quality of a specific solution by assigning it an *objective vector* $(y_1, y_2, \ldots, y_k)$ in the *objective space* $\boldsymbol{Y}$ (cf. Fig. 2).

Now, let us suppose that the objective space is a subset of the real numbers, i.e., $\boldsymbol{Y} \subseteq \mathbb{R}$, and that the goal of the optimization is to maximize the single objective. In such a single-objective optimization problem, a solution $\boldsymbol{x}^1 \in \boldsymbol{X}$ is better than another solution $\boldsymbol{x}^2 \in \boldsymbol{X}$ if $\boldsymbol{y}^1 > \boldsymbol{y}^2$ where $\boldsymbol{y}^1 = \boldsymbol{f}(\boldsymbol{x}^1)$ and $\boldsymbol{y}^2 = \boldsymbol{f}(\boldsymbol{x}^2)$. Although several optimal solutions may exist in decision space, they are all mapped to the same objective vector, i.e., there exists only a single optimum in objective space.

In the case of a vector-valued evaluation function $\boldsymbol{f}$ with $\boldsymbol{Y} \subseteq \mathbb{R}^k$ and $k > 1$, the situation of comparing two solutions $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ is more complex. Following the well-known concept of Pareto dominance, an objective vector $\boldsymbol{y}^1$ is said to *dominate* another objective vectors $\boldsymbol{y}^2$ ($\boldsymbol{y}^1 \succ \boldsymbol{y}^2$) if no component of $\boldsymbol{y}^1$ is smaller than the corresponding component of $\boldsymbol{y}^2$ and at least one component is greater. Accordingly, we can say that a solution $\boldsymbol{x}^1$ is better than another solution $\boldsymbol{x}^2$, i.e., $\boldsymbol{x}^1$ *dominates* $\boldsymbol{x}^2$ ($\boldsymbol{x}^1 \succ \boldsymbol{x}^2$), if $\boldsymbol{f}(\boldsymbol{x}^1)$ dominates $\boldsymbol{f}(\boldsymbol{x}^2)$. Here, optimal solutions, i.e., solutions not dominated by any other solution, may be mapped to different objective vectors. In other words: there may exist several optimal objective vectors representing different trade-offs between the objectives.

The set of optimal solutions in the decision space $\boldsymbol{X}$ is in general denoted as the *Pareto set* $\boldsymbol{X}^* \subseteq \boldsymbol{X}$, and we will denote its image in objective space as *Pareto front* $\boldsymbol{Y}^* = \boldsymbol{f}(\boldsymbol{X}^*) \subseteq \boldsymbol{Y}$. With many multiobjective optimization problems, knowledge

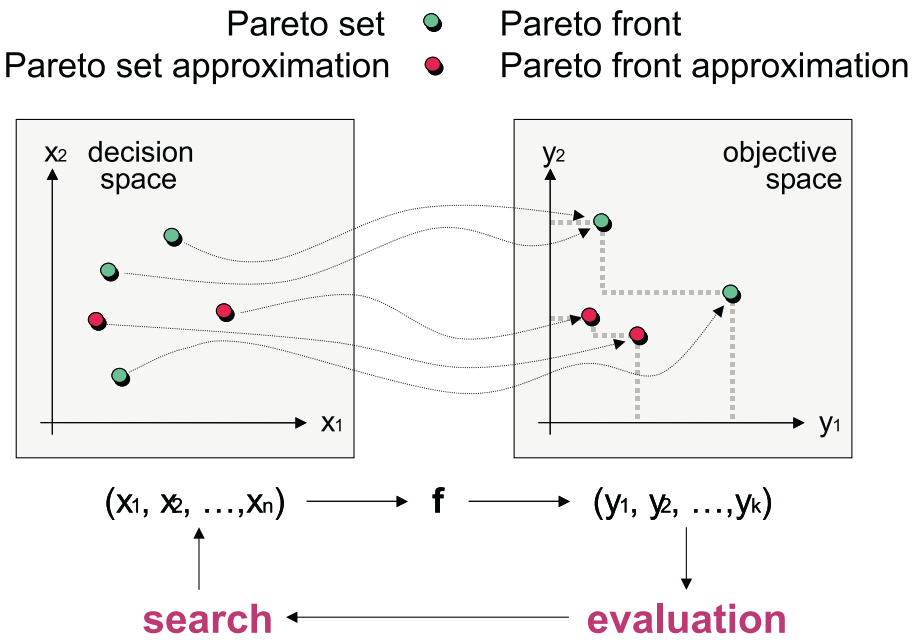

Figure 2: Illustration of a general (multiobjective) optimization problem

about this set helps the decision maker in choosing the best compromise solution. For instance, when designing telecommunication systems, engineers often perform a so-called design space exploration to learn more about the Pareto set. Thereby, the design space is reduced to the set of optimal trade-offs: a first step in selecting an appropriate system implementation.

In the following, we will assume that the goal of the optimization process is to find or approximate the Pareto set (in the case of a single objective, the Pareto front consists of a single objective vector only). Therefore, the outcome of an algorithm is considered to be a set of mutually nondominated solutions, or *Pareto set approximation* for short.

2.2 Blackbox Optimization

Randomized search algorithms form a class of heuristics that aim at finding good solutions to the optimization problem at hand without investigating all solutions. Different types of randomized search algorithms have been proposed such as evolutionary algorithms and simulated annealing, and they are characterized by the fact that minimal requirements with respect to the objective functions are made, i.e., they have been designed for so-called blackbox optimization scenarios. A blackbox optimization scenario assumes that nothing is known about the optimization criteria. Each objective function is considered as a black box, and the only way to obtain information about it is by asking for the objective vector to which a particular decision vector is mapped to.

In general, a randomized search algorithm works as follows. First, a decision vector $\boldsymbol{x}^1$ is chosen at random, and the corresponding objective vector $\boldsymbol{y}^1 = \boldsymbol{f}(\boldsymbol{x}^1)$ is determined by using the black boxes for the objective functions. In the next step, another solution $\boldsymbol{x}^2$ is selected randomly on the basis of the information given by $\boldsymbol{x}^1$ and $\boldsymbol{y}^1$. This process is repeated many times, where in iteration t all the previously investigated solutions $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^{t-1}$ and its objective function values $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^{t-1}$ can be used to select the next decision vector $\boldsymbol{x}^t$. The algorithm terminates until a certain termination criterion is fulfilled. In practice, though, randomized search algorithms do not store all previously considered solutions but rather only keep the best ones. Local search strategies, the Metropolis algorithm, or simulated annealing, for instance, only store one solution, while evolutionary algorithms usually work with a population of solutions. Furthermore, the different variants of randomized search algorithms used in practice distinguish themselves by the way new solutions are generated and by the criteria on the basis of which the solutions to be kept in the memory are selected.

In this context, we may ask why to use randomized search algorithms and which variant of randomized search algorithms to use. As to the first question, we often do not have sufficient time resources or insight into the problem to design a problem-specific algorithm, or the problem is too complex to be solved by exact methods. Since randomized search algorithms have minimum requirements with respect to the objective functions, they can be useful tools to tackle hard optimization problems. The second question is more difficult to answer. The No-Free-Lunch (NFL) theorem states that over all possible problems all search algorithms have the same average performance (Wolpert and Macready 1997). This makes clear that we cannot expect to find the perfect search method that outperforms all other techniques. One rather tries to identify classes of problems for which particular algorithms are well suited for. Evolutionary algorithms, e.g., are for practical reasons a good choice in a multiobjective scenario as the Pareto set can be approximated in a single optimization run.

3 Design Issues in Evolutionary Computation

In contrast to other randomized search algorithms, an evolutionary algorithm is characterized by three features:

1. a set of solution candidates is maintained,
2. a selection process is performed on this set which determines which solutions are considered to generate new solutions, and
3. several solutions may be combined in terms of recombination to generate new solutions.

By analogy to natural evolution, the solution candidates are called *individuals* and the set of solution candidates is called the *population*. Each individual represents a possible solution, i.e., a decision vector, to the problem at hand; however, an individual *is not* a decision vector but rather encodes it based on an appropriate representation.

At the beginning, the population is filled with a certain number of randomly chosen individuals. Each of these individuals is then evaluated on the basis of the objective

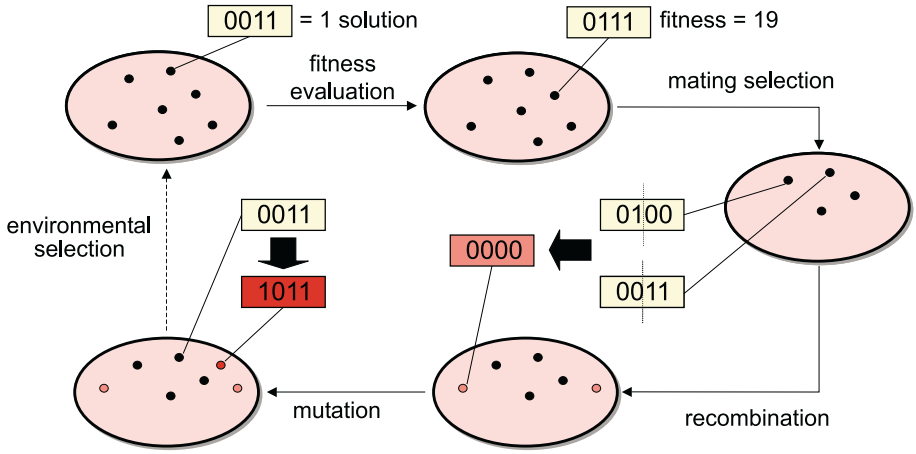

Figure 3: Outline of a general evolutionary algorithm for a problem with four binary decision variables

functions and is assigned a scalar value, the fitness value, which reflects its quality. Afterwards, a selection process is performed in which high-quality individuals are chosen for the generation of new individuals. Variation, the process of creating new solutions, is usually implemented on the basis of two operators: recombination and mutation. While recombination assembles a new solution by combining two or several individuals, mutation creates new individuals by slightly modifying single individuals. Finally, there is another selection procedure, environmental selection, which determines which of the old individuals and newly generated ones are kept in memory, i.e., in the new population. The steps fitness evaluation, mating selection, recombination, mutation, and environmental selection form one iteration of the algorithm, which is called *generation*. The number of iterations to be executed can be defined beforehand or may depend on on other conditions, e.g., stagnation in the population or existence of an individual with sufficient quality. The general flow of an EA is shown in Fig. 3.

In the following, we will briefly discuss the different issues arising when tailoring an EA to a specific problem, namely representation of the solution space, fitness assignment, selection, and variation.

3.1 Representation

With many optimization problems, the decision space has a straight-forward representation on a computer, e.g., for the knapsack problem a subset can be encoded by a binary bitstring where each position is associated with a particular item. For other problems, though, an appropriate encoding has to be defined, e.g., for graph problems (network topologies), scheduling problems, symbolic regression, etc. The choice of the representation is often underestimated, although it influences the performance of the algorithm; this holds for randomized search algorithms in general.

Most commonly used are vector representations (binary, integer, real), where the elements represent atomic units that are modified as a whole in the variation process. While vectors are usually of fixed length, tress can be used to encode solutions of variable length such as symbolic expressions and programs. Genetic programming, a subbranch of evolutionary computation, is devoted to EAs using tree representations. Many other representations such as matrices are possible, in particular mixed encoding can be often found with many real-world applications.

What the optimal choice of an encoding for a given problem is also depends on the variation operators; however, in general and as a rule of thumb, a representation should be

- *complete*, i.e., for each potential solution a corresponding encoding exists,
- *one-to-one*, i.e., each solution has a unique encoding, and if not at least
- *uniform*, i.e., each solution is represented by the same number of possible encodings,
- *feasible*, i.e., each possible encoding is mapped to a feasible solution, and
- *locality preserving*, i.e., the distance between two encoded solutions is the same as between the two decoded solutions with respect to appropriate metrics.

These criteria only can serve as guidelines and often not all of them can be fulfilled.

3.2 Fitness Assignment

The fitness of an individual describes its quality with regard to the optimization task under consideration on the basis of a real number (there may be exceptions, though). In the case of a single objective, often the objective function value is taken as the fitness value. However, there are different situations which require more complex fitness assignment strategies:

- multiple objectives are involved and the aim is to approximate the Pareto set,
- multiple optima are sought,
- constraints divide the decision space into feasible and infeasible solutions.

In the following, we will deal with each of these aspects separately.

3.2.1 Multiple Objectives

In the presence of multiple optimization criteria, the question is how to assign scalar fitness values such that the population is guided towards the Pareto set. There are different approaches, the major three are aggregation-based, criterion-based, and Pareto-based fitness assignment strategies, cf. Fig 7.

One approach which is built on the traditional techniques for generating trade-off surfaces is to aggregate the objectives into a single parameterized objective function. The parameters of this function are systematically varied during the optimization run in

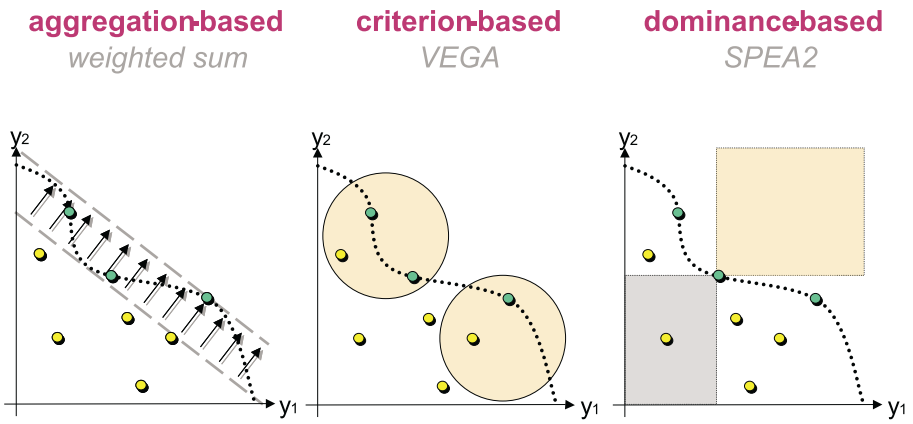

Figure 4: Different fitness assignment strategies

order to find a set of nondominated solutions instead of a single trade-off solution. For instance, some EA implementations use weighted-sum aggregation, where the weights represent the parameters which are changed during the evolution process (Hajela and Lin 1992; Ishibuchi and Murata 1996).

Criterion-based methods switch between the objectives during the selection phase. Here, the fitness of an individual is identical to the objective vector, i.e., it is not a scalar value. Each time an individual is chosen for reproduction, potentially a different objective will decide which member of the population will be selected for variation. For example, Schaffer (Schaffer 1985) proposed to divide the mating selection phase into k phases, where at phase i the individuals are chosen according to objective i; at each phase, the same number of individuals is selected. In contrast, Kursawe (Kursawe 1991) suggested assigning a probability to each objective which determines whether the objective will be the sorting criterion in the next selection step—the probabilities can be user-defined or chosen randomly over time.

The idea of calculating an individual's fitness on the basis of Pareto dominance goes back to Goldberg (Goldberg 1989), and different ways of exploiting the partial order on the population have been proposed. Some approaches use the dominance rank, i.e., the number of individuals by which an individual is dominated, to determine the fitness values (Fonseca and Fleming 1993). Others make use of the dominance depth; here, the population is divided into several fronts and the depth reflects to which front an individual belongs to (Srinivas and Deb 1994; Deb, Agrawal, Pratap, and Meyarivan 2000). Alternatively, also the dominance count, i.e., the number of individuals dominated by a certain individual, can be taken into account. For instance, SPEA (Zitzler and Thiele 1999) and SPEA2 (Zitzler, Laumanns, and Thiele 2002) assign fitness values on the basis of both dominance rank and count. Independent of the technique used, the fitness is related to the whole population in contrast to aggregation-based methods which calculate an individual's raw fitness value independently of other individuals.

3.2.2 Multiple Optima and Diversity Preservation

In the presence of multiple optimization criteria, we are often interested in finding the Pareto-optimal solutions. With many applications, though, this goal cannot achieved, and instead the aim is to generate a well-distributed subset of the Pareto set. A similar situation may arise in single-objective optimization, if we would like to find multiple different optima that all have the same objective function value. The problem, however, is a phenomenon known in Biology as *genetic drift*. Genetic drift denotes random changes in allele frequencies (alleles are the different "values" a gene can take) due to sampling errors in finite and particularly small populations. Here, we mean the tendency of small populations to converge to a single optimal solution.

If no specific means are incorporated, the diversity within the population may be lost due to genetic drift. One way to circumvent this problem is to incorporate density information into the fitness such that an individual's chance of being selected is decreased the greater the density of individuals in its neighborhood. This issue is closely related to the estimation of probability density functions in statistics, and the methods used in EAs can be classified according to the categories for techniques in statistical density estimation (Silverman 1986).

Kernel methods (Silverman 1986) define the neighborhood of a point in terms of a so-called Kernel function K which takes the distance to another point as an argument. In practice, for each individual the distances d_i to all other individuals $\boldsymbol{i}$ are calculated and after applying K the resulting values $K(d_i)$ are summed up. The sum of the K function values represents the density estimate for the individual. Fitness sharing is the most popular technique of this type within the field of evolutionary computation, which is used, e.g., in MOGA (Fonseca and Fleming 1993), NSGA (Srinivas and Deb 1994), and NPGA (Horn, Nafpliotis, and Goldberg 1994).

Nearest neighbor techniques (Silverman 1986) take the distance of a given point to its kth nearest neighbor into account in order to estimate the density in its neighborhood. Usually, the estimator is a function of the inverse of this distance. SPEA2 (Zitzler, Laumanns, and Thiele 2002), for instance, makes use of this density estimation technique as will be discussed in Section 4.

Histograms (Silverman 1986) define a third category of density estimators that use a hypergrid to define neighborhoods within the space. The density around an individual is simply estimated by the number of individuals in the same box of the grid. The hypergrid can be fixed, though usually it is adapted with regard to the current population as, e.g., in PAES (Knowles and Corne 1999).

Each of the three approaches is visualized in Fig. 5. However, due to space-limitations, a discussion of strengths and weaknesses of the various methods cannot be provided here—the interested reader is referred to Silverman's book (Silverman 1986). Furthermore, note that all of the above methods require a distance measure which can be defined on the encoded decision vectors, on the decoded decision vectors, or on the objective vectors. Most approaches consider the distance between two individuals as the distance between the two corresponding objective vectors.

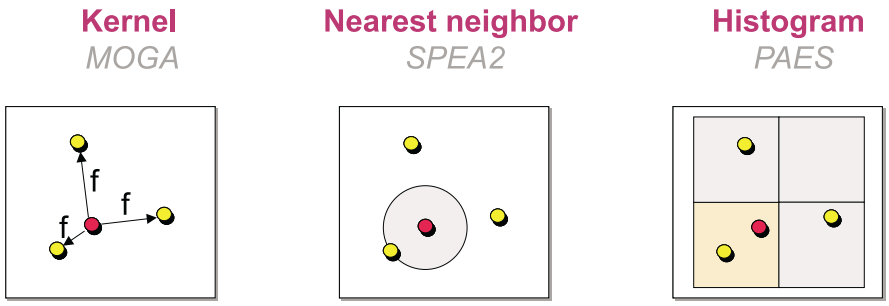

Figure 5: Illustration of diversity preservation techniques

3.2.3 Constraint Handling

With many applications, constraints restrict the set of admissible solutions, and we are interested in the solutions that meet these constraints *and* are optimal with respect to all other feasible solutions.

Assume that we have a set of inequality constraints $\boldsymbol{g}_1 \geq 0, \boldsymbol{g}_2 \geq 0, \ldots, \boldsymbol{g}_l \geq 0$ (other constraints can easily be expressed in this manner). The question we consider in the following is how to handle constraints within an EA such that the search focuses on the feasible solutions. In principle, there are three different ways:

- Firstly, the representation can be chosen such that the decoder function maps each individual to a feasible solutions.
- Secondly, we can make sure that only feasible solutions are generated. To this end, the initialization procedure of the population as well as the variation operators need to be designed accordingly.
- Most general is the penalty approach where the fitness of individuals violating any constraints is diminished. Usually, an overall constraint violation $C_{\boldsymbol{i}} \in \mathbb{R}$ is computed as

$$C_{\boldsymbol{i}} = \sum_{p=0}^{l} |\min\{\boldsymbol{g}_p(\boldsymbol{i}), 0\}|$$

  Provided that fitness is to be minimized, the fitness $F_{\boldsymbol{i}}$ of an individual can be considered as the sum of the original fitness $F'_{\boldsymbol{i}}$ and the overall constraint violation: $F_{\boldsymbol{i}} = F'_{\boldsymbol{i}} + C_{\boldsymbol{i}}$. Another possibility is to calculate the fitness in the following way, if we know the worst fitness value $F_{\max}$ possible for a feasible solution:

$$F_{\boldsymbol{i}} = \begin{cases} F'_{\boldsymbol{i}} & \text{if } C_{\boldsymbol{i}} = 0 \\ F_{\max} + C_{\boldsymbol{i}} & \text{else} \end{cases}$$

  The latter method ensures that feasible solutions are alway preferred over infeasible solutions.

Actually, only the last approach is directly related to fitness assignment, the other two are just mentioned here for reasons of completeness. Moreover, further possibilities emerge in multiobjective optimization, e.g., some authors have suggested a modified definition of Pareto dominance that takes constraints into account (Fonseca and Fleming 1998; Deb 2001).

3.3 Selection

Selection, which can be divided into mating and environmental selection, decides which individuals are considered for variation and which individuals survive, i.e., are kept in memory. It serves two conflicting goals: exploitation and exploration. The first term means we are trying to generate better solutions by modifying the current best solutions; in this sense, selection should favor the best individuals in the population. On the other hand, keeping diversity in the population is advantageous in avoiding to get stuck in local optima; in this respect, selection should focus on the diversity of the chosen individuals.

3.3.1 Mating Selection

Mostly, mating selection is implemented in terms of a randomized selection procedure. In this context, one has to distinguish two phases: sampling rate assignment and sampling. In the first phase, each individual is assigned a probability of being selected. The second phase realizes the actual selection, i.e., a predefined number of individuals is chosen on the basis of the sampling rates.

In the literature, different sampling rate assignment schemes have been proposed. Evolution strategies, e.g., assign each individual the same probability, while genetic algorithms traditionally used a fitness proportionate scheme. Fitness proportionate means the sampling rate is set to the ratio of an individual's fitness divided by the sum of the fitness values of all individuals in the population. The disadvantage of this scheme is that adding a constant to all fitness values results in different sampling rates; the larger the constant, the more likely it is that all individuals have similar sampling rates. Rank-based schemes avoid this problem by first sorting the individual according to the fitness values, and afterwards assigning the sampling rates in dependence of the position within the resulting order.

As to sampling, there are two main methods. Both methods can be best illustrated by thinking of a roulette wheel which is divided into N parts where N is the number of individuals in the population. The size of the slot associated with individual $\boldsymbol{i}$ is in proportion to its sampling rate $C_{\boldsymbol{i}}$. The first technique, known as roulette wheel reproduction, simply spins the roulette wheel as many times as individuals need to be selected; each time that individual that is associated with the slot under the pointer is selected. In contrast, with stochastic universal sampling (SUS) the roulette wheel is spun only once; instead N pointers are distributed evenly spaced around the roulette wheel. Each pointer determines one individual for selection. The advantage of SUS over roulette wheel reproduction is the lower variance with respect to the selected individuals.

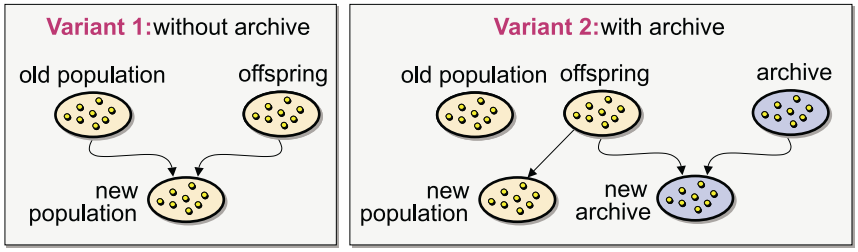

Figure 6: Two possible ways to implement environmental selection in a multiobjective EA

Finally, tournament selection integrates sampling rate assignment and sampling in the same procedure. A certain number of individuals is chosen uniformly from the population, and the individual with the best fitness within this group is selected. This process is iterated until the predefined number of individuals has been selected.

3.3.2 Environmental Selection

Environmental selection determines which of the individuals are kept in the population and is usually realized via a deterministic algorithm. One strategy is to replace the old population by the set of individuals that have been generated using mating selection and variation. Alternatively, parents and offspring can be combined and afterwards the best N individuals from the union form the next population, where N is the population size. Especially in the presence of multiple objectives, it is important to choose an appropriate environmental selection scheme as we would like to prevent nondominated individuals from being lost due to random effects. Therefore, the discussion will focus on multiobjective optimization in the following.

The two fundamental approaches used in multiobjective EAs are depicted in Fig. 6. The first one corresponds to the aforementioned strategy of combining parent and offspring population and to take the best N individuals. Alternatively, a secondary population, the so-called archive, can be maintained to which promising solutions in the population are copied at each generation. The archive may just be used as an external storage separate from the optimization engine or may be integrated into the EA by including archive members in the selection process.

As the memory resources are usually restricted, with both variants criteria have to be defined on this basis of which the solutions to be kept are selected. The dominance criterion is most commonly used. If an archive is maintained, the archive comprises only the current approximation of the Pareto set, i.e., dominated archive members are removed. Otherwise, special care is taken to ensure that nondominated solutions are preferred to dominated ones. However, the dominance criterion is in general not sufficient (e.g., for continuous problems the Pareto set may contain an infinite number of solutions); therefore, additional information is taken into account to reduce the number of stored solutions further. Examples are density information (Zitzler and Thiele 1999; Knowles and Corne 1999) and the time that has passed since the individual entered the

archive (Rudolph and Agapie 2000).

Most multiobjective EAs make use of a combination of dominance and density to choose the individuals that will be kept in the archive at every generation. However, these approaches may suffer from the problem of deterioration, i.e., solutions contained in the archive at generation t may be dominated by solutions that were members of the archive at any generation $t' < t$ and were discarded later. Recently, Laumanns et al. (Laumanns, Thiele, Zitzler, and Deb 2002) presented an archiving strategy which avoids this problem and guarantees to maintain a diverse set of Pareto-optimal solutions (provided that the optimization algorithm is able to generate the Pareto-optimal solutions).

3.4 Variation

Variation aims at generating new individuals on the basis of those individuals that were chosen during the mating selection phase. While mutation creates a new solution by modifying a given one, the recombination operator takes two or more individuals, combines them in a randomized fashion, and outputs one or more offspring. Since the choice of the operators is strongly problem-dependent and many different variation procedures have been suggested, we will only sketch the underlying ideas assuming a bitvector representation.

With binary vectors, mutation is usually implemented by flipping each bit independently with a predefined mutation probability p_m; a standard setting is $p_m = 1/n$, where n is the number of bits, such that in average one bit is flipped per individual. As to recombination, a popular operator is one-point or, in general, N-point crossover. The two parents are cut at randomly chosen positions into $N + 1$ parts, and afterwards children a created by alternately choosing parts from the first and from the second parent. As with mutation, there is a crossover probability associated with this operator. With probability p_c, the two parents are recombined and the two resulting children are returned; otherwise, two copies of the parents are returned.

4 An Example Evolutionary Algorithm: SPEA2

As an illustrative example, we here present a generic implementation of a multiobjective EA, namely SPEA2 (Zitzler, Laumanns, and Thiele 2002), that has been used to tackle the network processor application discussed in the lecture. The overall algorithm is as follows:

Algorithm 1 (SPEA2 Main Loop)

Input:	N	*(population size)*
	$\overline{N}$	*(archive size)*
	T	*(maximum number of generations)*
Output:	$\boldsymbol{A}$	*(nondominated set)*

Step 1: **Initialization***: Generate an initial population* $\boldsymbol{P}_0$ *and create the empty archive (external set)* $\overline{\boldsymbol{P}}_0 = \emptyset$*. Set* $t = 0$*.*

Step 2: **Fitness assignment**: *Calculate fitness values of individuals in $\boldsymbol{P}_t$ and $\overline{\boldsymbol{P}}_t$ (cf. Section 4.1).*

Step 3: **Environmental selection**: *Copy all nondominated individuals in $\boldsymbol{P}_t$ and $\overline{\boldsymbol{P}}_t$ to $\overline{\boldsymbol{P}}_{t+1}$. If size of $\overline{\boldsymbol{P}}_{t+1}$ exceeds $\overline{N}$ then reduce $\overline{\boldsymbol{P}}_{t+1}$ by means of the truncation operator, otherwise if size of $\overline{\boldsymbol{P}}_{t+1}$ is less than $\overline{N}$ then fill $\overline{\boldsymbol{P}}_{t+1}$ with dominated individuals in $\boldsymbol{P}_t$ and $\overline{\boldsymbol{P}}_t$ (cf. Section 4.2).*

Step 4: **Termination**: *If $t \geq T$ or another stopping criterion is satisfied then set $\boldsymbol{A}$ to the set of decision vectors represented by the nondominated individuals in $\overline{\boldsymbol{P}}_{t+1}$. Stop.*

Step 5: **Mating selection**: *Perform binary tournament selection with replacement on $\overline{\boldsymbol{P}}_{t+1}$ in order to fill the mating pool.*

Step 6: **Variation**: *Apply recombination and mutation operators to the mating pool and set $\boldsymbol{P}_{t+1}$ to the resulting population. Increment generation counter ($t = t + 1$) and go to Step 2.*

SPEA2 uses a fine-grained fitness assignment strategy which incorporates density information as will be described in Section 4.1. Furthermore, an archive of fixed size is maintained that contains a representation of the current nondominated front. Whenever the number of nondominated individuals is less than the predefined archive size, the archive is filled up by dominated individuals; otherwise, a truncation method is applied which is described in Section 4.2.

4.1 Fitness Assignment

To avoid the situation that individuals dominated by the same archive members have identical fitness values, with SPEA2 for each individual both dominating and dominated solutions are taken into account. In detail, each individual $\boldsymbol{i}$ in the archive $\overline{\boldsymbol{P}}_t$ *and* the population $\boldsymbol{P}_t$ is assigned a strength value $S_{\boldsymbol{i}}$, representing the number of solutions it dominates:

$$S_{\boldsymbol{i}} = |\{\boldsymbol{j} \mid \boldsymbol{j} \in \boldsymbol{P}_t + \overline{\boldsymbol{P}}_t \wedge \boldsymbol{i} \succ \boldsymbol{j}\}|$$

where $|\cdot|$ denotes the cardinality of a set, $+$ stands for multiset union and the symbol $\succ$ corresponds to the Pareto dominance relation. On the basis of the S values, the raw fitness $R_{\boldsymbol{i}}$ of an individual $\boldsymbol{i}$ is calculated:

$$R_{\boldsymbol{i}} = \sum_{\boldsymbol{j} \in \boldsymbol{P}_t + \overline{\boldsymbol{P}}_t, \boldsymbol{j} \succ \boldsymbol{i}} S_{\boldsymbol{j}}$$

That is the raw fitness is determined by the strengths of its dominators in both archive and population. It is important to note that fitness is to be minimized here, i.e., $R_{\boldsymbol{i}} = 0$ corresponds to a nondominated individual, while a high $R_{\boldsymbol{i}}$ value means that $\boldsymbol{i}$ is dominated by many individuals (which in turn dominate many individuals). This scheme is illustrated in Figure 7.

Although the raw fitness assignment provides a sort of niching mechanism based on the concept of Pareto dominance, it may fail when most individuals do not dominate each other. Therefore, additional density information is incorporated to discriminate

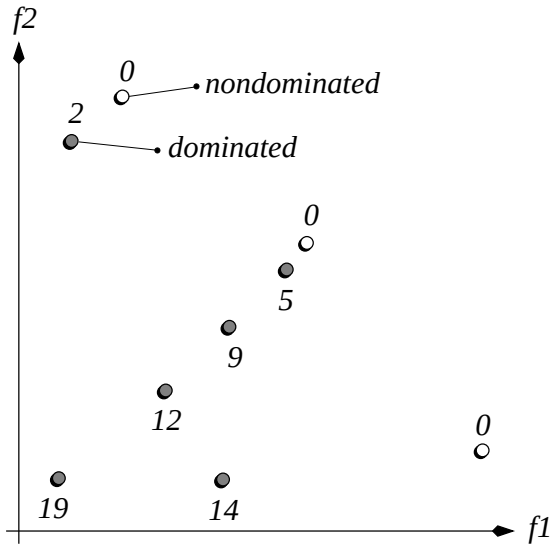

Figure 7: Illustration of the fitness assignment scheme used SPEA2 for a maximization problem with two objectives f_1 and f_2; the numbers give the raw fitness values of the corresponding individuals.

between individuals having identical raw fitness values. The density estimation technique used in SPEA2 is an adaptation of the k-th nearest neighbor method (Silverman 1986), where the density at any point is a (decreasing) function of the distance to the k-th nearest data point. Here, we simply take the inverse of the distance to the k-th nearest neighbor as the density estimate. To be more precise, for each individual $\boldsymbol{i}$ the distances (in objective space) to all individuals $\boldsymbol{j}$ in archive and population are calculated and stored in a list. After sorting the list in increasing order, the k-th element gives the distance sought, denoted as $\sigma_{\boldsymbol{i}}^k$. As a common setting, we use k equal to the square root of the sample size (Silverman 1986), thus, $k = \sqrt{N + \overline{N}}$. Afterwards, the density $D_{\boldsymbol{i}}$ corresponding to $\boldsymbol{i}$ is defined by

$$D_{\boldsymbol{i}} = \frac{1}{\sigma_{\boldsymbol{i}}^k + 2}$$

In the denominator, two is added to ensure that its value is greater than zero and that $D_{\boldsymbol{i}} < 1$. Finally, adding $D_{\boldsymbol{i}}$ to the raw fitness value $R_{\boldsymbol{i}}$ of an individual $\boldsymbol{i}$ yields its fitness $F_{\boldsymbol{i}}$:

$$F_{\boldsymbol{i}} = R_{\boldsymbol{i}} + D_{\boldsymbol{i}}$$

4.2 Environmental Selection

The archive update operation (Step 3 in Algorithm 1) in SPEA2 was designed in such a way that i) the number of individuals contained in the archive is constant over time,

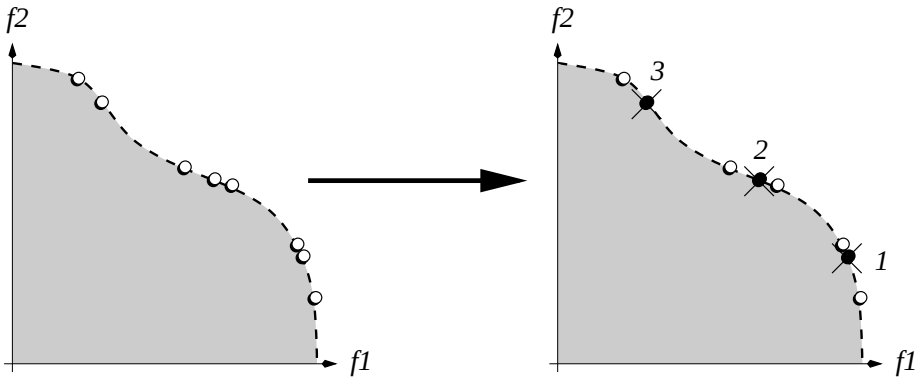

Figure 8: Illustration of the archive truncation method used in SPEA2. On the right, a nondominated set is shown. On the left, it is depicted which solutions are removed in which order by the truncate operator (assuming that $\overline{N} = 5$).

and ii) boundary solutions are not lost.

During environmental selection, the first step is to copy all nondominated individuals, i.e., those which have a fitness lower than one, from archive and population to the archive of the next generation:

$$\overline{\boldsymbol{P}}_{t+1} = \{\boldsymbol{i} \mid \boldsymbol{i} \in \boldsymbol{P}_t + \overline{\boldsymbol{P}}_t \wedge F_{\boldsymbol{i}} < 1\}$$

If the nondominated front fits exactly into the archive ($|\overline{\boldsymbol{P}}_{t+1}| = \overline{N}$) the environmental selection step is completed. Otherwise, there can be two situations: Either the archive is too small ($|\overline{\boldsymbol{P}}_{t+1}| < \overline{N}$) or too large ($|\overline{\boldsymbol{P}}_{t+1}| > \overline{N}$). In the first case, the best $\overline{N} - |\overline{\boldsymbol{P}}_{t+1}|$ dominated individuals in the previous archive and population are copied to the new archive. This can be implemented by sorting the multiset $\boldsymbol{P}_t + \overline{\boldsymbol{P}}_t$ according to the fitness values and copy the first $\overline{N} - |\overline{\boldsymbol{P}}_{t+1}|$ individuals $\boldsymbol{i}$ with $F_{\boldsymbol{i}} \geq 1$ from the resulting ordered list to $\overline{\boldsymbol{P}}_{t+1}$. In the second case, when the size of the current nondominated (multi)set exceeds $\overline{N}$, an archive truncation procedure is invoked which iteratively removes individuals from $\overline{\boldsymbol{P}}_{t+1}$ until $|\overline{\boldsymbol{P}}_{t+1}| = \overline{N}$. Here, at each iteration that individual $\boldsymbol{i}$ is chosen for removal for which $\boldsymbol{i} \leq_d \boldsymbol{j}$ for all $\boldsymbol{j} \in \overline{\boldsymbol{P}}_{t+1}$ with

$$\begin{aligned} \boldsymbol{i} \leq_d \boldsymbol{j} \quad :\Leftrightarrow \quad & \forall\, 0 < k < |\overline{\boldsymbol{P}}_{t+1}| \,:\, \sigma_{\boldsymbol{i}}^k = \sigma_{\boldsymbol{j}}^k \quad \vee \\ & \exists\, 0 < k < |\overline{\boldsymbol{P}}_{t+1}| \,: \\ & \left[\left(\forall\, 0 < l < k \,:\, \sigma_{\boldsymbol{i}}^l = \sigma_{\boldsymbol{j}}^l\right) \wedge\, \sigma_{\boldsymbol{i}}^k < \sigma_{\boldsymbol{j}}^k\right] \end{aligned}$$

where $\sigma_{\boldsymbol{i}}^k$ denotes the distance of $\boldsymbol{i}$ to its k-th nearest neighbor in $\overline{\boldsymbol{P}}_{t+1}$. In other words, the individual which has the minimum distance to another individual is chosen at each stage; if there are several individuals with minimum distance the tie is broken by considering the second smallest distances and so forth. How this truncation technique works is illustrated in Figure 8.

5 Applications

There are numerous applications of EAs in the area of telecommunications ranging from network design to routing problems. An overview of multicriteria studies in this context can be found in (Coello Coello, Van Veldhuizen, and Lamont 2002).

In the lecture, we will present a network processor design application that involves several objectives. Due to space restrictions, the problem is not discussed here; instead, the interested reader is referred to the original paper (Thiele, Chakraborty, Gries, and Künzli 2002).

References

Bäck, T., U. Hammel, and H.-P. Schwefel (1997). Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation 1*(1), 3–17.

Coello Coello, C. A., D. A. Van Veldhuizen, and G. B. Lamont (2002). *Evolutionary Algorithms for Solving Multi-Objective Problems*. New York: Kluwer.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. Chichester, UK: Wiley.

Deb, K., S. Agrawal, A. Pratap, and T. Meyarivan (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel (Eds.), *Parallel Problem Solving from Nature – PPSN VI*, Berlin, pp. 849–858. Springer.

Fonseca, C. M. and P. J. Fleming (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, California, pp. 416–423. Morgan Kaufmann.

Fonseca, C. M. and P. J. Fleming (1998). Multiobjective optimization and multiple constraint handling with evolutionary algorithms—part ii: Application example. *IEEE Transactions on Systems, Man, and Cybernetics 28*(1), 38–47.

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

Hajela, P. and C.-Y. Lin (1992). Genetic search strategies in multicriterion optimal design. *Structural Optimization 4*, 99–107.

Horn, J., N. Nafpliotis, and D. E. Goldberg (1994). A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Computation*, Volume 1, Piscataway, NJ, pp. 82–87. IEEE Press.

Ishibuchi, H. and T. Murata (1996). Multi-objective genetic local search algorithm. In *Proceedings of 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, Piscataway, NJ, pp. 119–124. IEEE Press.

Knowles, J. D. and D. W. Corne (1999). The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Congress on Evolutionary Computation (CEC99)*, Volume 1, Piscataway, NJ, pp. 98–105. IEEE Press.

Kursawe, F. (1991). A variant of evolution strategies for vector optimization. In H.-P. Schwefel and R. Männer (Eds.), *Parallel Problem Solving from Nature*, Berlin, pp. 193–197. Springer.

Laumanns, M., L. Thiele, E. Zitzler, and K. Deb (2002, 9-13 July). Archiving with guaranteed convergence and diversity in multi-objective optimization. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska (Eds.), *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, New York, pp. 439–447. Morgan Kaufmann Publishers.

Rudolph, G. and A. Agapie (2000). Convergence properties of some multi-objective evolutionary algorithms. In *Congress on Evolutionary Computation (CEC 2000)*, Volume 2, Piscataway, NJ, pp. 1010–1016. IEEE Press.

Schaffer, J. D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette (Ed.), *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, pp. 93–100. sponsored by Texas Instruments and U.S. Navy Center for Applied Research in Artificial Intelligence (NCARAI).

Silverman, B. W. (1986). *Density estimation for statistics and data analysis*. London: Chapman and Hall.

Srinivas, N. and K. Deb (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation 2*(3), 221–248.

Thiele, L., S. Chakraborty, M. Gries, and S. Künzli (2002). *Network Processor Design: Issues and Practices, Volume 1*, Chapter Exploration of Network Processor Architectures, pp. 55–89. Morgan Kaufmann.

Wolpert, D. H. and W. G. Macready (1997, April). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation 1*(1), 67–82.

Zitzler, E., M. Laumanns, and L. Thiele (2002). SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In K. Giannakoglou, D. Tsahalis, J. Periaux, K. Papaliliou, and T. Fogarty (Eds.), *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems. Proceedings of the EUROGEN2001 Conference, Athens, Greece, September 19-21, 2001*, Barcelona, Spain, pp. 95–100. International Center for Numerical Methos in Engineering (CIMNE).

Zitzler, E. and L. Thiele (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation 3*(4), 257–271.