E. Zitzler, M. Laumanns, and S. Bleuler. A Tutorial on Evolutionary Multiobjective Optimization. In X. Gandibleux et al., editors, Metaheuristics for Multiobjective Optimisation, volume 535 of Lecture Notes in Economics and Mathematical Systems. Springer, 2004.

A Tutorial on Evolutionary Multiobjective Optimization

Eckart Zitzler, Marco Laumanns, and Stefan Bleuler

Swiss Federal Institute of Technology (ETH) Zurich,
Computer Engineering and Networks Laboratory (TIK),
Gloriastrasse 35, CH-8092 Zurich, Switzerland
{zitzler,laumanns,bleuler}@tik.ee.ethz.ch

Abstract. Multiple, often conflicting objectives arise naturally in most real-world optimization scenarios. As evolutionary algorithms possess several characteristics that are desirable for this type of problem, this class of search strategies has been used for multiobjective optimization for more than a decade. Meanwhile evolutionary multiobjective optimization has become established as a separate subdiscipline combining the fields of evolutionary computation and classical multiple criteria decision making.
This paper gives an overview of evolutionary multiobjective optimization with the focus on methods and theory. On the one hand, basic principles of multiobjective optimization and evolutionary algorithms are presented, and various algorithmic concepts such as fitness assignment, diversity preservation, and elitism are discussed. On the other hand, the tutorial includes some recent theoretical results on the performance of multiobjective evolutionary algorithms and addresses the question of how to simplify the exchange of methods and applications by means of a standardized interface.

1 Introduction

The term evolutionary algorithm (EA) stands for a class of stochastic optimization methods 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 [1]. All of these approaches operate on a set of candidate solutions. Using strong simplifications, this set is subsequently modified by the 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 [1]. In particular, they possess several characteristics that are desirable for problems involving i) multiple conflicting objectives, and ii) intractably large and highly complex

search spaces. As a result, numerous algorithmic variants have been proposed and applied to various problem domains since the mid-1980s. The rapidly growing interest in the area of multiobjective evolutionary algorithms (MOEAs) is reflected by, e.g., a conference series [45] and two recent books dedicated to this subject [7, 4].

This paper gives an overview of this relatively new field with the focus on methods and theory. Section 2 summarizes basic principles of multiobjective optimization and evolutionary computation and forms the basis for the remainder of this tutorial. The following section focuses on algorithm design issues and presents concepts and techniques that have been developed to deal with the additional complexity caused by multiple objectives. These issues will be illustrated on the basis of a specific algorithmic variant, namely SPEA2 [46]. Afterwards, we will discuss some recent theoretical results with respect to the performance of multiobjective EAs: limit behavior, run-time complexity, and quality measures. Finally, a practically important issue will be addressed: a platform and programming language independent interface for search algorithms that allows to provide search procedures and test problems in a precompiled format.

2 Basic Principles

2.1 Multiobjective Optimization

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 $\boldsymbol{f} : \boldsymbol{X} \to \boldsymbol{Y}$ 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. 1).

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 $\boldsymbol{y}^1 = f(\boldsymbol{x}^1)$ and $\boldsymbol{y}^2 = f(\boldsymbol{x}^2)$. 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 to 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.

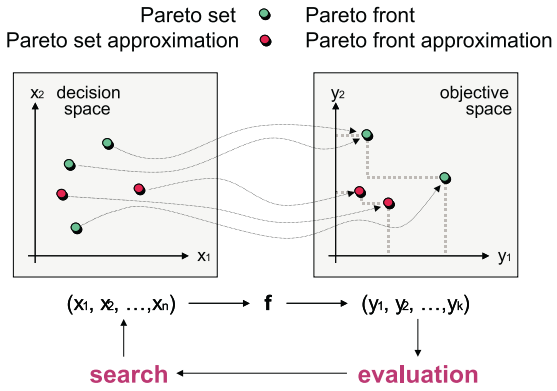


Fig. 1. Illustration of a general multiobjective optimization problem

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 about this set helps the decision maker in choosing the best compromise solution. For instance, when designing computer 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 implementation.

Although there are different ways to approach a multiobjective optimization problem, e.g., by aggregation of the objectives into a single one, most work in the area of evolutionary multiobjective optimization has concentrated on the approximation of the Pareto set. Therefore, we will assume in the following that the goal of the optimization is to find or approximate the Pareto set. Accordingly, the outcome of an MOEA is considered to be a set of mutually nondominated solutions, or *Pareto set approximation* for short.

2.2 Evolutionary Computation

Generating the Pareto set can be computationally expensive and is often infeasible, because the complexity of the underlying application prevents exact methods from being applicable. For this reason, a number of stochastic search strategies such as evolutionary algorithms, tabu search, simulated annealing, and ant colony optimization have been developed: they usually do not guarantee to identify optimal trade-offs but try to find a good approximation, i.e., a set of solutions whose objective vectors are (hopefully) not too far away from the optimal objective vectors.

Roughly speaking, a general stochastic search algorithm consists of three parts: i) a working memory that contains the currently considered solution candidates, ii) a selection module, and iii) a variation module as depicted in Fig. 2.

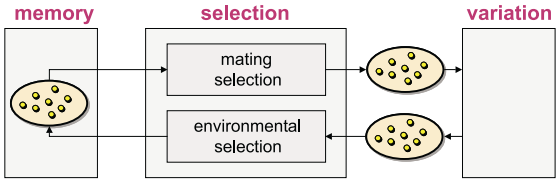


Fig. 2. Components of a general stochastic search algorithm

As to the selection, one can distinguish between mating and environmental selection. Mating selection aims at picking promising solutions for variation and usually is performed in a randomized fashion. In contrast, environmental selection determines which of the previously stored solutions and the newly created ones are kept in the internal memory. The variation module takes a set of solutions and systematically or randomly modifies these solutions in order to generate potentially better solutions. In summary, one iteration of a stochastic optimizer includes the consecutive steps mating selection, variation, and environmental selection; this cycle may be repeated until a certain stopping criterion is fulfilled.

Many stochastic search strategies have been originally designed for single-objective optimization and therefore consider only one solution at a time, i.e., the working memory contains just a single solution. As a consequence, no mating selection is necessary and variation is performed by modifying the current solution candidate.

In contrast, an evolutionary algorithm is characterized by three features:

1. a set of solution candidates is maintained,
2. a mating selection process is performed on this set, 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.

The mating selection process usually consists of two stages: fitness assignment and sampling. In the first stage, the individuals in the current population are evaluated in the objective space and then assigned a scalar value, the *fitness*, reflecting their quality. Afterwards, a so-called mating pool is created by random sampling from the population according to the fitness values. For instance, a commonly used sampling method is binary tournament selection. Here, two individuals are randomly chosen from the population, and the one with the better fitness value is copied to the mating pool. This procedure is repeated until the mating pool is filled.

Then, the variation operators are applied to the mating pool. With EAs, there are usually two of them, namely the recombination and the mutation operator.

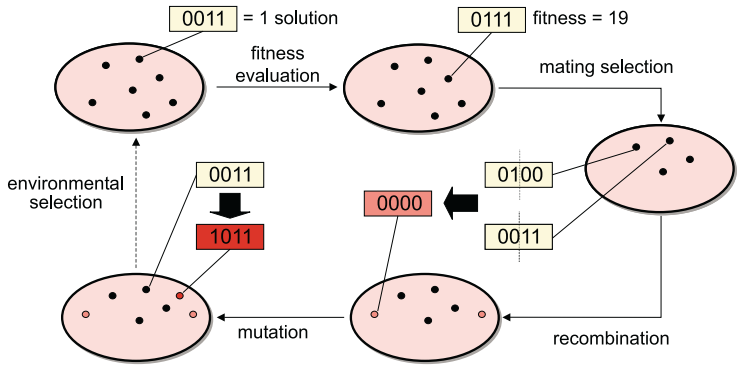


Fig. 3. Outline of a general evolutionary algorithm for a problem with four binary decision variables

The recombination operator takes a certain number of parents and creates a predefined number of children by combining parts of the parents. To mimic the stochastic nature of evolution, a so-called crossover probability is associated with this operator. While there are various EAs variants that do not make use of recombination operators, mutation is essential for any EA implementation. The mutation operator modifies individuals by changing small parts in the associated vectors according to a given mutation rate. Note that due to random effects some individuals in the mating pool may not be affected by variation and therefore simply represent a copy of a previously generated solution.

Finally, environmental selection determines which individuals of the population and the modified mating pool form the new population. The simplest way is to use the latter set as the new population. An alternative is to combine both sets and deterministically choose the best individuals for survival. There are various other possibilities, which will no be discussed in detail here.

Based on the above concepts, natural evolution is simulated by an iterative computation process as shown in Fig. 3. First, an initial population is created at random (or according to a predefined scheme), which is the starting point of the evolution process. Then a loop consisting of the steps evaluation (fitness assignment), selection, recombination, and/or mutation is executed a certain number of times. Each loop iteration is called a *generation*, and often a predefined maximum number of generations serves as the termination criterion of the loop. But also other conditions, e.g., stagnation in the population or existence of an individual with sufficient quality, may be used to stop the simulation. At the end, the best individuals in the final population represent the outcome of the EA.

3 Algorithm Design Issues

The goal of approximating the Pareto set is itself multiobjective. For instance, we would like to minimize the distance of the generated solutions to the Pareto set and to maximize the diversity of the achieved Pareto set approximation. This is certainly a fuzzy statement, and we will see in Section 5.3 that it is impossible to exactly describe what a good approximation is in terms of a number of criteria such as closeness to the Pareto set, diversity, etc. Nevertheless, it well illustrates the two fundamental goals in MOEA design: guiding the search towards the Pareto set and keeping a diverse set of nondominated solutions. There are different problem difficulties that make these goals hard to achieve as discussed in detail in [6, 9]; accordingly various design variants have been suggested for different types of problems, and we will briefly summarize the main variants in this section.

The first MOEA design goal is mainly related to mating selection, in particular to the problem of assigning scalar fitness values in the presence of multiple optimization criteria. The second design goal concerns selection in general because we want to avoid that the population contains mostly identical solutions (with respect to the objective space and the decision space). Finally, a third issue which addresses both of the above goals is elitism, i.e., the question of how to prevent nondominated solutions from being lost.

In the following, each of these aspects will be discussed: fitness assignment, diversity preservation, and elitism. Remarkably, they are well reflected by the development of the field of evolutionary multiobjective optimization. While the first studies on multiobjective evolutionary algorithms were mainly concerned with the problem of guiding the search towards the Pareto set [35, 14, 25], all approaches of the second generation incorporated in addition a niching concept in order to address the diversity issue [13, 38, 20]. The importance of elitism was recognized and supported experimentally in the late nineties [31, 48, 44], and most of the third generation MOEAs implement this concept in one or the other way, e.g., [24, 8, 46].

3.1 Fitness Assignment

In contrast to single-objective optimization, where objective function and fitness function are often identical, both fitness assignment and selection must allow for several objectives with multi-criteria optimization problems. In general, one can distinguish aggregation-based, criterion-based, and Pareto-based fitness assignment strategies, cf. Fig 4.

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 order to find a set of nondominated solutions instead of a single trade-off solution. For instance, some MOEAs use weighted-sum aggregation, where the weights represent the parameters which are changed during the evolution process [16, 21].

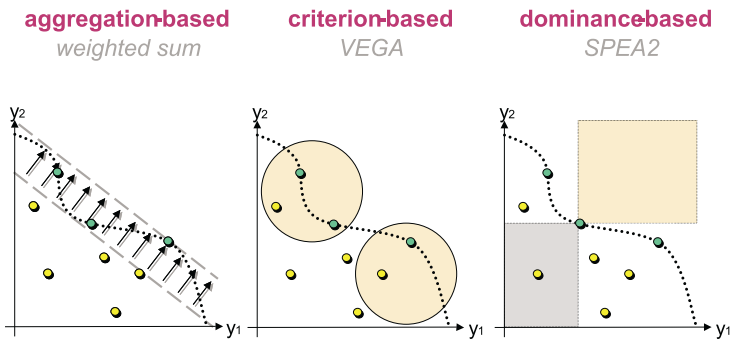


Fig. 4. Different fitness assignment strategies

Criterion-based methods switch between the objectives during the selection phase. Each time an individual is chosen for reproduction, potentially a different objective will decide which member of the population will be copied into the mating pool. For example, Schaffer [35] proposed filling equal portions of the mating pool according to the distinct objectives, while Kursawe [25] 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 [15], 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 [13]. 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 [38, 8]. Alternatively, also the dominance count, i.e., the number of individuals dominated by a certain individual, can be taken into account. For instance, SPEA [48] and SPEA2 [46] 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 Diversity Preservation

Most MOEAs try to maintain diversity within the current Pareto set approximation by incorporating density information into the selection process: 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 MOEAs can be classified according to the categories for techniques in statistical density estimation [37].

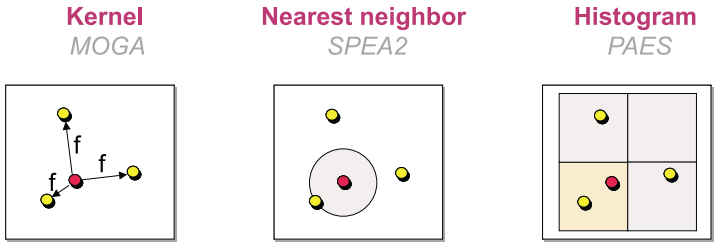


Fig. 5. Illustration of diversity preservation techniques

Kernel methods [37] 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 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 [13], NSGA [38], and NPGA [20].

Nearest neighbor techniques [37] 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 [46], for instance, calculates for each individual the distance to the kth nearest individual and adds the reciprocal value to the raw fitness value (fitness is to be minimized).

Histograms [37] 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 [24].

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 [37]. Furthermore, note that all of the above methods require a distance measure which can be defined on the genotype, on the phenotype with respect to the decision space, or on the phenotype with respect to the objective space. Most approaches consider the distance between two objective vectors as the distance between the corresponding individuals.

3.3 Elitism

Elitism addresses the problem of losing good solutions during the optimization process due to random effects. One way to deal with this problem is to combine the old population and the offspring, i.e., the mating pool after variation,

and to apply a deterministic selection procedure—instead of replacing the old population by the modified mating pool. 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. These two general approaches are illustrated in Fig. 6.

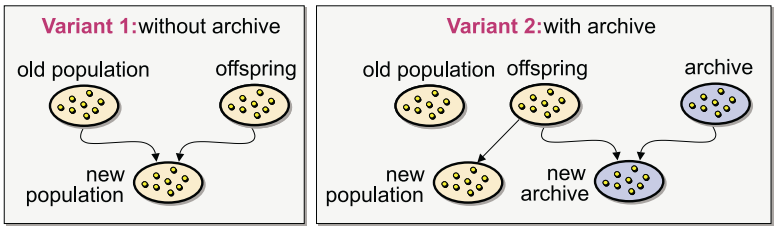


Fig. 6. Two possible ways to implement elitism

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 [48, 24] and the time that has passed since the individual entered the archive [34].

Most elitist MOEAs 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. [28] 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). This approach will be discussed in Section 5.1

3.4 Further Design Aspects

Many EA implementations integrate other randomized search strategies such as local search in order to maximize the overall efficiency. In [47], e.g., specific domain knowledge was incorporated by means of a heuristic procedure that locally

improves individuals; in detail, the heuristic is applied to each individual in the population before evaluation and fitness assignment takes places. Various studies have shown the usefulness of such hybrid approaches [21, 23]. Jaskziewicz [22], e.g., investigated a local search EA hybrid for the multiobjective 0/1 knapsack problem; this algorithm outperformed other pure multiobjective EAs in terms of convergence and diversity for this particular problem.

Appropriate data structures that allow fast domination checks represent another design aspect, which is particularly important if a large archive in combination with a large population is used. When the archive is updated, possibly a large number of solutions need to be compared to many other solutions in the archive, which can have a significant effect on the algorithm's run-time. Tree-based data structures, though, can help to more efficiently perform the update operation as Mostaghim, Teich, and Tyagi have shown [30]; the interested reader is referred to this study for an in-depth discussion of this subject.

4 An Example: SPEA2

The Strength Pareto Evolutionary Algorithm (SPEA) [48] is a relatively recent technique for finding or approximating the Pareto set for multiobjective optimization problems. In different studies [48, 44], SPEA compared favorably with other MOEAs and therefore has been a point of reference in various recent investigations, e.g., [5]. Furthermore, it has been used in different applications, e.g., [26]. Here, an improved version, namely SPEA2, is described in order to illustrate how the concepts described in Section 3 can be implemented in an MOEA.

In the design of SPEA2, the goal was to eliminate the potential weaknesses of its predecessor and to incorporate most recent results in order to create a powerful and up-to-date MOEA. The main differences of SPEA2 in comparison to SPEA are:

- An improved fitness assignment scheme, which takes for each individual into account how many individuals it dominates and it is dominated by.
- A nearest neighbor density estimation technique, which allows a more precise guidance of the search process.
- A new archive truncation methods that guarantees the preservation of boundary solutions.

As has been shown in a comparative case study [46], the proposed algorithm provides good performance in terms of convergence and diversity, outperforms SPEA, and compares well to PESA and NSGA-II on various, well-known test problems.

4.1 Differences between SPEA and SPEA2

As SPEA (Strength Pareto Evolutionary Algorithm) [48] forms the basis for SPEA2, a brief summary of the algorithm is given here. For a more detailed description the interested reader is referred to [43].

SPEA uses a regular population and an archive (external set). Starting with an initial population and an empty archive, the following steps are performed per iteration. First, all nondominated population members are copied to the archive; any dominated individuals or duplicates (regarding the objective values) are removed from the archive during this update operation. If the size of the updated archive exceeds a predefined limit, further archive members are deleted by a clustering technique which preserves the characteristics of the nondominated front. Afterwards, fitness values are assigned to both archive and population members:

- Each individual $\boldsymbol{i}$ in the archive is assigned a strength value $S(\boldsymbol{i}) \in [0, 1)$, which at the same time represents its fitness value $F(\boldsymbol{i})$. $S(\boldsymbol{i})$ is the number of population members $\boldsymbol{j}$ that are dominated by or equal to $\boldsymbol{i}$ with respect to the objective values, divided by the population size plus one.
- The fitness $F(\boldsymbol{j})$ of an individual $\boldsymbol{j}$ in the population is calculated by summing the strength values $S(\boldsymbol{i})$ of all archive members $\boldsymbol{i}$ that dominate or are equal to $\boldsymbol{j}$, and adding one at the end.

The next step represents the mating selection phase where individuals from the union of population and archive are selected by means of binary tournaments. Please note that fitness is to be minimized here, i.e., each individual in the archive has a higher chance to be selected than any population member. Finally, after recombination and mutation the old population is replaced by the resulting offspring population.

Although SPEA performed well in different comparative studies [48, 44], this algorithm has potential weaknesses:

Fitness assignment: Individuals that are dominated by the same archive members have identical fitness values. That means in the case when the archive contains only a single individual, all population members have the same rank independent of whether they dominate each other or not. As a consequence, the selection pressure is decreased substantially and in this particular case SPEA behaves almost like a random search algorithm.

Density estimation: If many individuals of the current generation are incomparable, i.e., do not dominate each other, none or very little information can be obtained on the basis of the partial order defined by the dominance relation. In this situation, which is very likely to occur in the presence of more than two objectives, density information has to be used in order to guide the search more effectively. Clustering makes use of this information, but only with regard to the archive and not to the population.

Archive truncation: Although the clustering technique used in SPEA is able to reduce the nondominated set without destroying its characteristics, it may lose outer solutions. However, these solutions should be kept in the archive in order to obtain a good spread of nondominated solutions.

Next, we will address these issues and describe the improvements made in SPEA2 in detail.

Algorithm 1 SPEA2 Main Loop

Input: M (offspring population size)
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) $\boldsymbol{A}_0 = \emptyset$. Set $t = 0$.

Step 2: **Fitness assignment**: Calculate fitness values of individuals in $\boldsymbol{P}_t$ and $\boldsymbol{A}_t$ (cf. Section 4.3).

Step 3: **Environmental selection**: Copy all nondominated individuals in $\boldsymbol{P}_t$ and $\boldsymbol{A}_t$ to $\boldsymbol{A}_{t+1}$. If size of $\boldsymbol{A}_{t+1}$ exceeds N then reduce $\boldsymbol{A}_{t+1}$ by means of the truncation operator, otherwise if size of $\boldsymbol{A}_{t+1}$ is less than N then fill $\boldsymbol{A}_{t+1}$ with dominated individuals in $\boldsymbol{P}_t$ and $\boldsymbol{A}_t$ (cf. Section 4.4).

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 $\boldsymbol{A}_{t+1}$. Stop.

Step 5: **Mating selection**: Perform binary tournament selection with replacement on $\boldsymbol{A}_{t+1}$ 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.

4.2 The SPEA2 Main Loop

Algorithm 1 forms the core of SPEA2. In contrast to SPEA, SPEA2 uses a fine-grained fitness assignment strategy which incorporates density information as will be described in Section 4.3. Furthermore, the archive size is fixed, i.e., whenever the number of nondominated individuals is less than the predefined archive size, the archive is filled up by dominated individuals; with SPEA, the archive size may vary over time. In addition, the clustering technique, which is invoked when the nondominated front exceeds the archive limit, has been replaced by an alternative truncation method which has similar features but does not loose boundary points. Details on the environmental selection procedure will be given in Section 4.4. Finally, another difference to SPEA is that only members of the archive participate in the mating selection process.

4.3 SPEA2 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 $\boldsymbol{A}_t$ *and* the population $\boldsymbol{P}_t$ is assigned a strength value $S(\boldsymbol{i})$, representing

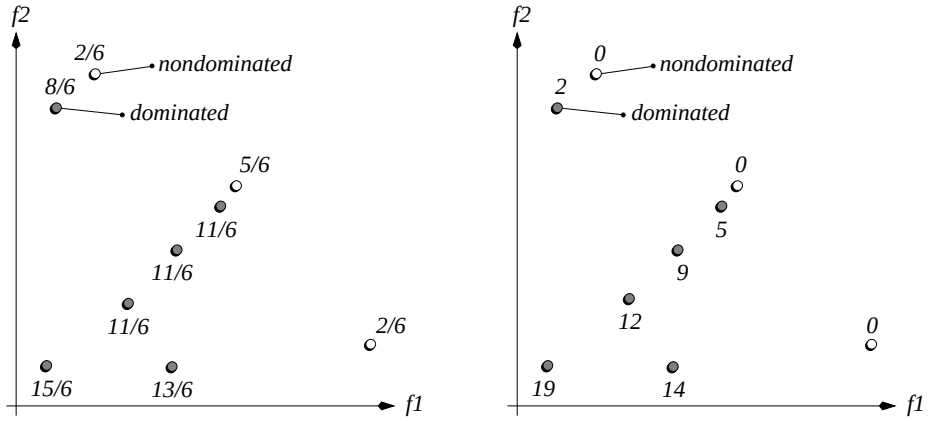


Fig. 7. Comparison of fitness assignment schemes in SPEA and SPEA2 for a maximization problem with two objectives f_1 and f_2. On the left, the fitness values for a given population according to the SPEA scheme is shown. On the right, the raw SPEA2 fitness values for the same population are depicted.

the number of solutions it dominates:[1]

$$S(i) = |\{j \mid j \in P_t + A_t \wedge i \succ j\}|$$

where $|\cdot|$ denotes the cardinality of a set, $+$ stands for multiset union and the symbol $\succ$ corresponds to the Pareto dominance relation extended to individuals ($i \succ j$ if the decision vector encoded by i dominates the decision vector encoded by j). On the basis of the S values, the raw fitness $R(i)$ of an individual i is calculated:

$$R(i) = \sum_{j \in P_t + A_t, j \succ i} S(j)$$

That is the raw fitness is determined by the strengths of its dominators in both archive and population, as opposed to SPEA where only archive members are considered in this context. It is important to note that fitness is to be minimized here, i.e., $R(i) = 0$ corresponds to a nondominated individual, while a high $R(i)$ value means that 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 between individuals having identical raw fitness values. The density estimation technique used in SPEA2 is an adaptation of the k-th

[1] This (and the following) formula slightly differs from the one presented in [2], where also individuals which have identical objective values contribute to the strength of an individual.

nearest neighbor method [37], 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 $\sigma_{\boldsymbol{i}}^k$. A common setting is to use the square root of the sample size for k [37]; however, $k = 1$ is often sufficient and lead to a more efficient implementation. 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 $F(\boldsymbol{i})$:

$$F(\boldsymbol{i}) = R(\boldsymbol{i}) + D(\boldsymbol{i})$$

The run-time of the fitness assignment procedure is dominated by the density estimator ($\mathcal{O}(L^2 \log L)$), while the calculation of the S and R values is of complexity $\mathcal{O}(L^2)$, where $L = M + N$.

The combination of a raw fitness based on the dominance relation and a secondary fitness based on density information is very common for MOEAs. The individual algorithms distinguish themselves by i) how these two fitness values are calculated and ii) how they are combined to form a scalar overall fitness value. Various ways to calculate raw fitness values based on the dominance relation are described in Section 3.1, while Section 3.2 discusses different density estimation techniques for calculating secondary fitness values. The early approaches of MOGA [13] and NSGA [38], for instance, both used a kernel density estimator as a secondary fitness value to modify the raw fitness values, which are given by the dominance rank and the dominance depth, repectively. Subsequently, the NSGA-II [8] replaced the kernel density estimator by a computationally more efficient coordinate-wise nearest neighbor technique. Concerning the second point of combining the raw fitness and the secondary fitness, it is common practise to give the raw fitness highest priority in the sense that an individual with a worse raw fitness never receives a better overall fitness that another. This essentially corresponds to a lexicographic ordering of the two sub-goals of convergence to the Pareto set and diversity, though other weightings are thinkable.

4.4 SPEA2 Environmental Selection

The archive update operation (Step 3 in Alg. 1) in SPEA2 differs from the one in SPEA in two aspects: i) the number of individuals contained in the archive is constant over time, and ii) the truncation method prevents boundary solutions from being removed.

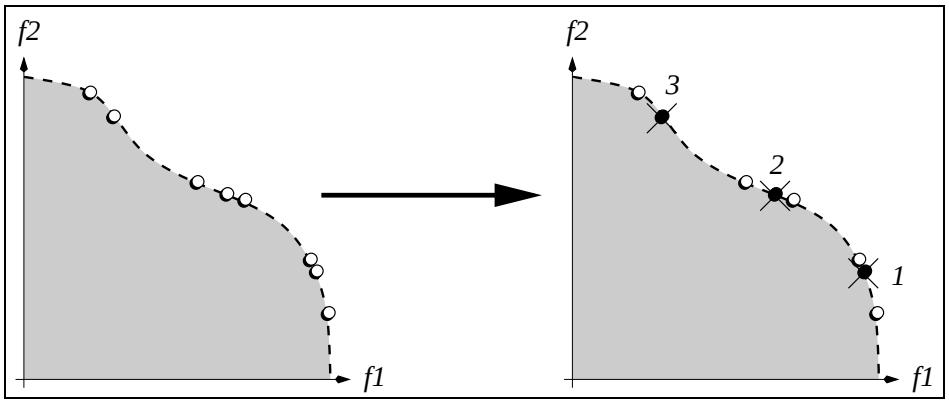


Fig. 8. Illustration of the archive truncation method used in SPEA2. On the right, a nondominated set is shown. On the left, it is depicted which solutions are removed in which order by the truncate operator (assuming that $N = 5$).

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:

$$\boldsymbol{A}_{t+1} = \{\boldsymbol{i} \mid \boldsymbol{i} \in \boldsymbol{P}_t + \boldsymbol{A}_t \wedge F(\boldsymbol{i}) < 1\}$$

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

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

where $\sigma_{\boldsymbol{i}}^k$ denotes the distance of $\boldsymbol{i}$ to its k-th nearest neighbor in $\boldsymbol{A}_{t+1}$. 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.

The worst run-time complexity of the truncation operator is $\mathcal{O}(L^3)$ ($L = M + N$); however, an efficient implementation can lead to a substantially smaller average run-time complexity. This can be achieved by, e. g., a lazy evaluation of the

k-th nearest neighbors. Normally, the individuals' k-th nearest neighbors are already different for very low k values, thus the more distant neighbors are only calculated when they are actually used and not in advance. Hence, an a priori computation, sorting, and update of the nearest neighbor lists is avoided. An efficient implementation of SPEA2 can be found on `http://www.tik.ee.ethz.ch/pisa/`.

5 Performance of Multiobjective Evolutionary Algorithms

Basically, there are two ways to assess the performance of MOEAs: i) theoretically by analysis or ii) empirically by simulation. In the following, we will present some recent results with respect to both approaches. On the one hand, we will discuss the limit behavior of MOEAs and provide a run-time analysis of two simple MOEAs. On the other hand, we will address the question of how to assess the quality of the outcome of an MOEA from a theoretical perspective.

5.1 Limit Behavior

The limit behavior of MOEAs is of interest when we want to investigate their global convergence properties. It refers to the question what the algorithm is able to achieve in the limit, i.e., when unlimited time resources are available.

Global Convergence Roughly speaking, an MOEA is called globally convergent if the sequence of Pareto front approximations $\boldsymbol{A}^{(t)}$ it produces converges to the true Pareto front $\boldsymbol{Y}^*$ while the number of generations t goes to infinity. It is intuitively clear that this property can only be fulfilled with unlimited memory resources, as the cardinality of the Pareto front can be arbitrary large in general [34]. Practical implementations, however, always have to deal with limited memory resources. In this case one is restricted to finding a subset of the Pareto front, and a globally convergent algorithm should guarantee $\boldsymbol{A}^{(t)} \longrightarrow \boldsymbol{Y}' \subseteq \boldsymbol{Y}^*$.

In the single-objective case, two conditions are sufficient to guarantee global convergence:

1. A strictly covering mutation distribution, which ensures that any solution $\boldsymbol{x}' \in \boldsymbol{X}$ can be produced from every $\boldsymbol{x} \in \boldsymbol{X}$ by mutation with a positive probability, and
2. An elitist (environmental) selection rule, which ensures that an optimal solution is not lost and no deterioration can occur.

While the mutation condition transfers easily to the multiobjective case, the elitist selection rule does not. This is due to the fact that a total order of the solutions is not given anymore and solutions can become incomparable to each other. If more nondominated solutions arise than can be stored in the population, some have to be discarded. This environmental selection strategy essentially determines whether an algorithm is globally convergent or not.

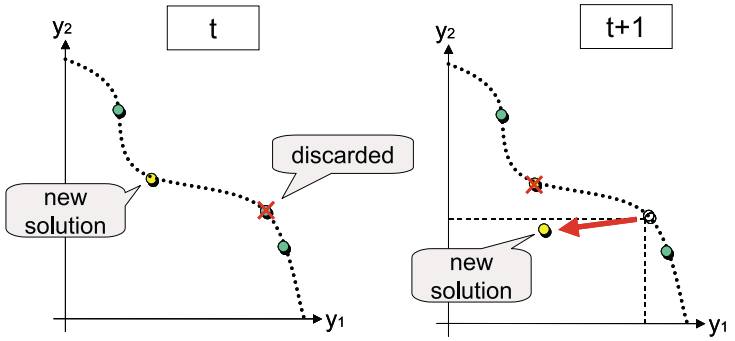


Fig. 9. A possible deterioration of a hypothetical population of size 3: In generation t, a forth nondominated solution is found and a truncation operation is invoked, e.g., based on density information, to reduce the population to its maximum size. In generation $t+1$, another solution is found that is dominated by the former, now discarded solution. The new solution, however, is not dominated by the *current* population members. Now, the truncation procedure again has to decide which solution to discard and might take a decision to keep this new solution (e.g., as it has a lower density around it). In comparing the new situation after generation $t+1$ with the situation before generation t, one immediately notices that the population became worse: the outer solutions remained the same, while the inner solution 'deteriorated'.

Rudolph [33], Hanne [17, 18] and Rudolph and Agapie [34] proposed different selection schemes that preclude deterioration and guarantee convergence. The basic idea is that solutions are only discarded if they are replaced by a dominating alternative. This ensures the sufficient monotonicity in the sequence of accepted solutions. However, no statements could be made with respect to the final distribution of solutions.

Most state-of-the-art MOEAs, though, take in addition to the dominance criterion density information into account. Nevertheless, for all of these algorithms it can be proven that a succession such selection steps can lead to deterioration, as depicted in Fig. 9. Hence, convergence can no longer be guaranteed for any of these algorithms. Deb [7] suggested a steady-state MOEA which attempts to maintain spread while attempting to converge to the true Pareto-optimal front, but there is no proof for its convergence properties. Knowles [23] has analyzed two further possibilities, metric-based archiving and adaptive grid archiving. The metric-based strategy requires a function that assigns a scalar value to each possible approximation set reflecting its quality and fulfilling certain monotonicity conditions. Convergence is then defined as the achievement of a local optimum of the quality function. The adaptive grid archiving strategy implemented in PAES provably maintains solutions in some 'critical' regions of the Pareto set once they have been found, but convergence can only be guaranteed for the solutions at the extremes of the Pareto set.

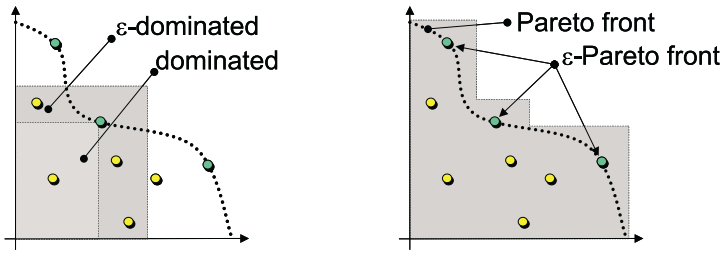


Fig. 10. The concept of ϵ-dominance and ϵ-Pareto fronts

In order to design a selection rule that enables global convergence with limited memory resources *together* with a well distributed subset of solutions we have to define what we understand by a well distributed Pareto set approximation and then define a selection algorithm which respects this *and* fulfills the monotonicity condition to preclude deterioration.

Concept of Pareto Set Approximation Since finding the Pareto front of an arbitrary objective space $\boldsymbol{Y}$ is usually not practical because of its size, one needs to be less ambitious in general. Therefore, the ϵ-approximate Pareto set was proposed in [27] as practical solution concept as it not only represents all vectors $\boldsymbol{Y}^*$ but also consists of a smaller number of elements. The ϵ-approximate Pareto front is based on the following generalization of the dominance relation (see also Figure 10):

Definition 1 (ϵ-Dominance). *Let $\boldsymbol{a}, \boldsymbol{b} \in \boldsymbol{Y}$. Then $\boldsymbol{a}$ is said to ϵ-dominate $\boldsymbol{b}$ for some $\epsilon > 0$, denoted as $\boldsymbol{a} \succ_\epsilon \boldsymbol{b}$, if*

$$\epsilon \cdot a_i \geq b_i \quad \forall i \in \{1, \ldots, k\} \tag{1}$$

Definition 2 (ϵ-approximate Pareto front). *Let $\boldsymbol{Y} \subseteq \mathbb{R}^{+k}$ be a set of vectors and $\epsilon \geq 1$. Then a set $\boldsymbol{Y}_\epsilon$ is called an ϵ-approximate Pareto front of $\boldsymbol{Y}$, if any vector $\boldsymbol{b} \in \boldsymbol{Y}$ is ϵ-dominated by at least one vector $\boldsymbol{a} \in \boldsymbol{Y}_\epsilon$, i.e.*

$$\forall \boldsymbol{b} \in \boldsymbol{Y} \; \exists \boldsymbol{a} \in \boldsymbol{Y}_\epsilon : \quad \boldsymbol{a} \succ_\epsilon \boldsymbol{b}. \tag{2}$$

The set of all ϵ-approximate Pareto fronts of $\boldsymbol{Y}$ is denoted as $\boldsymbol{P}_\epsilon(\boldsymbol{Y})$.

Of course, the set $\boldsymbol{Y}_\epsilon$ is not unique. Many different concepts for ϵ-efficiency and the corresponding Pareto front approximations exist in the operations research literature, a survey is given by [19]. As most of the concepts deal with infinite sets, they are not practical for our purpose of producing and maintaining a representative subset. Nevertheless, they are of theoretical interest and have nice properties which can for instance be used in convergence proofs, see [17] for an application in MOEAs.

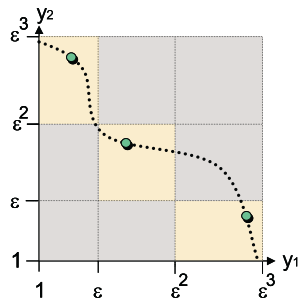


Fig. 11. Grid on the objective space induced by Alg. 3

Note that this concept of approximation can also be used with slightly different definitions of ϵ-dominance, e.g. the following additive approximation

$$\epsilon_i + a_i \geq b_i \quad \forall i \in \{1, \ldots, k\} \tag{3}$$

where ϵ_i are constants, separately defined for each coordinate.

A further refinement of the concept of ϵ-approximate Pareto sets leads to the following definition.

Definition 3 (ϵ-Pareto front). *Let $\boldsymbol{Y} \subseteq \mathbb{R}^{+m}$ be a set of vectors and $\epsilon > 0$. Then a set $\boldsymbol{Y}^*_\epsilon \subseteq \boldsymbol{Y}$ is called an ϵ-Pareto front of $\boldsymbol{Y}$ if*

1. *$\boldsymbol{Y}^*_\epsilon$ is an ϵ-approximate Pareto set of $\boldsymbol{Y}$, i.e. $\boldsymbol{Y}^*_\epsilon \in \boldsymbol{P}_\epsilon(\boldsymbol{Y})$, and*
2. *$\boldsymbol{Y}^*_\epsilon$ contains Pareto points of $\boldsymbol{Y}$ only, i.e. $\boldsymbol{Y}^*_\epsilon \subseteq \boldsymbol{Y}^*$.*

The set of all ϵ-Pareto fronts of $\boldsymbol{Y}$ is denoted as $\boldsymbol{P}^*_\epsilon(\boldsymbol{Y})$.

A Selection Algorithm for Guaranteed Convergence and Diversity Based on the above concept of Pareto front approximation a selection strategy can be constructed that fulfills the second sufficient condition for global convergence. The following algorithm (Alg. 2) has a two level concept. On the coarse level, the search space is discretized by a division into boxes (see Alg. 3 and Fig. 11), where each vector uniquely belongs to one box. Using a generalized dominance relation on these boxes, the algorithm always maintains a set of non-dominated boxes, thus guaranteeing the ϵ-approximation property. On the fine level at most one element is kept in each box. Within a box, each representative vector can only be replaced by a dominating one, thus guaranteeing convergence.

The following theorem shows that an MOEA using the above selection strategy fulfills the monotonicity criterion and never loses 'important' solutions.

Theorem 1 ([28]). *Let $\boldsymbol{Y}^{(t)} = \bigcup_{j=1}^{t} \boldsymbol{y}^{(j)}, 1 \leq \boldsymbol{y}_i^{(j)} \leq B$, be the set of all objective vectors created by an MOEA and given to the selection operator as defined in Alg. 2. Then $\boldsymbol{A}^{(t)}$ is an ϵ-Pareto set of $\boldsymbol{Y}^{(t)}$ with bounded size, i.e.,*

1. $\boldsymbol{A}^{(t)} \in \boldsymbol{P}_{\epsilon}^{*}(\boldsymbol{Y}^{(t)})$
2. $|\boldsymbol{A}^{(t)}| \leq \left(\frac{\log B}{\log \epsilon}\right)^{(k-1)}$

Now, if the mutation distribution guarantees that every solution will be produced we can prove global convergence of this MOEA.

Though the limit behavior might be of mainly theoretical interest, it is of high practical relevance that now the problem of partial deterioration, which is imminent even in most modern MOEAs, can be solved. Using the proposed archiving strategy maintaining an ϵ-Pareto front, the user can be sure to have in addition to a representative, well distributed approximation also a true elitist algorithm in the sense that no better solution had been found and subsequently lost during the run.

Algorithm 2 Selection function for ϵ-Pareto front

Input: $\boldsymbol{A}, \boldsymbol{y}$
$D := \{\boldsymbol{y}' \in \boldsymbol{A} | box(\boldsymbol{y}) \succ box(\boldsymbol{y}')\}$
if $D \neq \emptyset$ **then**
  $\boldsymbol{A}' := \boldsymbol{A} \cup \{\boldsymbol{y}\} \setminus D$
else if $\exists \boldsymbol{y}' : (box(\boldsymbol{y}') = box(\boldsymbol{y}) \wedge \boldsymbol{y} \succ \boldsymbol{y}')$ **then**
  $\boldsymbol{A}' := \boldsymbol{A} \cup \{\boldsymbol{y}\} \setminus \{\boldsymbol{y}'\}$
else if $\nexists \boldsymbol{y}' : box(\boldsymbol{y}') = box(\boldsymbol{y}) \vee box(\boldsymbol{y}') \succ box(\boldsymbol{y})$ **then**
  $\boldsymbol{A}' := \boldsymbol{A} \cup \{\boldsymbol{y}\}$
else
  $\boldsymbol{A}' := \boldsymbol{A}$
end if
Output: $\boldsymbol{A}'$

Algorithm 3 function *box*

Input: $\boldsymbol{y}$
for all $i \in \{1, \ldots, k\}$ **do**
  $b_i := \lfloor \frac{\log \boldsymbol{y}_i}{\log \epsilon} \rfloor$
end for
$b := (b_1, \ldots, b_k)$
Output: b {box index vector}

5.2 Run-Time Analysis

In addition to limit behavior, we are often interested in a quantitative analysis, specifically the expected running time for a given class of problems and the success probability for a given optimization time. For single-objective evolutionary algorithms many such results are contained in [32]. For the optimization

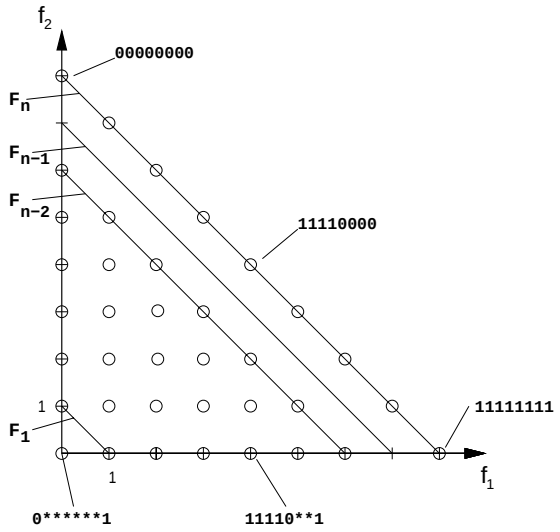


Fig. 12. Objective space of the LOTZ function with $n = 8$

of pseudo-Boolean functions an extensive theory has been built up by Wegener et al., see e.g. [42], and Droste, Jansen, and Wegener [10, 11]; a methodological overview is given in [41].

For the multiobjective case, no run-time analysis was available until recently. Scharnow et al. [36] analyzed a (1+1)-EA under multiple, non-conflicting objectives.[2] A first analysis of different population-based MOEAs on a two-objective problem with conflicting objectives was given by [29], which will be described here.

The model problem for this investigation, LOTZ, is a multiobjective generalization of the LEADINGONES problem which has been thoroughly analyzed for example in [32] and [11]. The algorithms are instances of a steady state $(\mu+1)$-EA with variable population size and differ in the manner how the parents are sampled from the population.

The Model Problem As the example problem for this analysis, we consider the maximization of a 2-dimensional vector valued function, LOTZ, which maps n binary decision variables to 2 objective functions.

[2] The term $(\mu + \lambda)$ means that i) the population contains μ individuals, ii) λ new individuals are created by means of variation in each iteration, and iii) the best μ individuals among parents and offspring survive.

Definition 4. *The pseudo-Boolean function* LOTZ : $\{0,1\}^n \to \mathbb{N}^2$ *is defined as*

$$\text{LOTZ}(x_1, \ldots, x_n) = \left(\sum_{i=1}^{n} \prod_{j=1}^{i} x_j, \sum_{i=1}^{n} \prod_{j=i}^{n} (1 - x_j) \right)$$

The abbreviation LOTZ stands for "Leading Ones, Trailing Zeroes" and means that we want to simultaneously maximize the number of leading ones and trailing zeroes in a bit-string.

The objective space of this problem can be partitioned into $n+1$ sets $\boldsymbol{F}_i, i = 0, \ldots, n$ (see Fig. 12). The index i corresponds to the sum of both objective values, i.e., $(f_1, f_2) \in \boldsymbol{F}_i$ if $i = f_1 + f_2$. Obviously, $\boldsymbol{F}_n$ represents the Pareto front $\boldsymbol{Y}^*$. The sub-domains $\boldsymbol{X}_i$ are defined as the sets containing all decision vectors which are mapped to elements of $\boldsymbol{F}_i$. They are of the form $1^a 0 *^{(n-i-2)} 10^b$ with $a + b = i$ for $i < n$, and $1^a 0^b$ with $a + b = n$ for $\boldsymbol{X}_n$.

Multi-start Strategies How long does it take to optimize the LOTZ function? Droste et al. [11] have proven that the expected running time of a (1+1)-EA on LEADINGONES is $\Theta(n^2)$. Using the same algorithm with an appropriate generalization of the acceptance criterion (either accepting only dominating offspring or by using a weighted sum as a scalar surrogate objective) will certainly lead to finding *one* element of the Pareto set in the same amount of time.

To find the entire Pareto set with such a (1+1) EA we can consider the multi-start option, i.e. to run to the EA several times, and collect all non-dominated solutions in an archive. For the acceptance criterion based on the dominance relation, the random variable describing the number of ones in the final solution of each single run follows a binomial distribution with $p = 0.5$. Hence the probability of finding the "outer" points of the Pareto set decreases exponentially. This would mean that the running time of this strategy until all Pareto optimal points are found is exponentially large in n.

Another possibility would be to use the multi-start option together with a weighted sum of the objective values. However, an appropriate choice of the weights is very difficult. In our case, equal weights would lead to the same situation as before, with a very low probability to reach the outer points. Any other selection of weights will let the sequence of search points converge to one of the outer points of the Pareto set. The remaining points must be found "on the way", but the probability of such events is not easy to calculate. Even if we could supply $n + 1$ different weights corresponding to each of the $n + 1$ optimal objective vectors, this strategy would still need $(n + 1) \cdot \Theta(n^2) = \Theta(n^3)$ steps.

A last possibility would be to use a simple strategy known from classical multiobjective function optimization. In this case, we optimize only one objective, e.g. the number of leading ones, and constrain the other objective to be strictly larger than its value obtained in the previous optimization run. Therefore, we find all $n + 1$ Pareto vectors in $n + 1$ runs of a single-objective EA with an additional constraint. At the best, this strategy again needs $\Theta(n^3)$ steps.

Algorithm 4 Simple Evolutionary Multiobjective Optimizer (SEMO)

1: Choose an initial individual $\boldsymbol{x}$ uniformly from $\boldsymbol{X} = \{0,1\}^n$
2: $\boldsymbol{P} \leftarrow \{\boldsymbol{x}\}$
3: **loop**
4: Select one element $\boldsymbol{x}$ out of $\boldsymbol{P}$ uniformly.
5: Create offspring $\boldsymbol{x}'$ by flipping a randomly chosen bit.
6: $\boldsymbol{P} \leftarrow \boldsymbol{P} \setminus \{\boldsymbol{x}'' \in \boldsymbol{P} | \boldsymbol{x}' \succ \boldsymbol{x}''\}$
7: **if** $\nexists \boldsymbol{x}'' \in \boldsymbol{P}$ such that $(\boldsymbol{x}'' \succ \boldsymbol{x}' \vee \boldsymbol{f}(\boldsymbol{x}'') = \boldsymbol{f}(\boldsymbol{x}'))$ **then**
8: $\boldsymbol{P} \leftarrow \boldsymbol{P} \cup \{\boldsymbol{x}'\}$
9: **end if**
10: **end loop**

The above discussion indicates that a (1+1) strategy may not be the best approach to find the Pareto set. Moreover, most of the current multiobjective optimization algorithms use the concept of an archive that maintains a set of vectors nondominated among all decision vectors visited so far. This indicates that the concept of a population is vital in multiobjective evolutionary optimization. In the next sections, we analyze two simple *population-based* steady state EAs.

Two Population-based MOEAs In this section, the running time analysis of two population-based MOEAs is presented, SEMO (see Alg. 4) and FEMO (see Alg. 5). The two algorithm are the same except for the fact that SEMO uses uniform sampling from the population while FEMO always chooses the parent that has produced the least number of children so far. For the running time analysis, we consider the number of necessary evaluations of the objective function until the optimum is reached.

For the analysis, we divide the run of SEMO into two distinct phases: the first phase lasts until the first individual representing an optimal objective vector has entered the population, and the second phase ends when the whole Pareto set has been found.

Theorem 2 (Expected running time of SEMO [29]). *The expected running time of SEMO until the first optimal objective vector is found is* $O(n^2)$.

After the first optimal objective vector has been found, the expected running time of SEMO until the entire Pareto front is generated is $\Theta(n^3)$.

The total expected running time of Alg. 4 until all Pareto-optimal points are found is $\Theta(n^3)$.

The proof is based on the fact that in the first phase the population always consists of one individual only. Here we have to wait for at most n events of choosing the leftmost zero or the rightmost one for mutation, each of which happens with a probability of $1/n$. In the second phase, the population grows with the generation of new Pareto-optimal solutions from which only the outer ones regarding the objective space can mutate to a new solution that is not

Algorithm 5 Fair Evolutionary Multiobjective Optimizer (FEMO)

1: Choose an initial individual $\boldsymbol{x}$ uniformly from $\boldsymbol{X} = \{0,1\}^n$
2: $w(\boldsymbol{x}) \leftarrow 0$ {Initialize offspring count}
3: $\boldsymbol{P} \leftarrow \{\boldsymbol{x}\}$
4: **loop**
5: Select one element $\boldsymbol{x}$ out of $\{y \in \boldsymbol{P} | w(y) \leq w(\boldsymbol{x}'') \ \forall \boldsymbol{x}'' \in \boldsymbol{P}\}$ uniformly.
6: $w(\boldsymbol{x}) \leftarrow w(\boldsymbol{x}) + 1$ {Increment offspring count}
7: Create offspring $\boldsymbol{x}'$ by flipping a randomly chosen bit.
8: $\boldsymbol{P} \leftarrow \boldsymbol{P} \setminus \{\boldsymbol{x}'' \in \boldsymbol{P} | \boldsymbol{x}' \succ \boldsymbol{x}''\}$
9: **if** $\nexists \boldsymbol{x}'' \in \boldsymbol{P}$ such that $(\boldsymbol{x}'' \succ \boldsymbol{x}' \vee \boldsymbol{f}(\boldsymbol{x}'') = \boldsymbol{f}(\boldsymbol{x}'))$ **then**
10: $\boldsymbol{P} \leftarrow \boldsymbol{P} \cup \{\boldsymbol{x}'\}$
11: $w(\boldsymbol{x}') \leftarrow 0$ {Initialize offspring count}
12: **end if**
13: **end loop**

already in the population. This event has a probability of $\Theta(1/(n|P|))$, and summing over all sizes of P from 1 to $n+1$ leads to the total running time of $\Theta(n^3)$.

The main weakness of SEMO for the optimization problem under consideration lies in the fact that a large number of mutations are allocated to parents whose neighborhood has already been explored sufficiently. On the other hand, an optimal sampling algorithm would use always the most promising parent at the border of the current population. Of course, this information is not available in a black box optimization scenario.

The uniform sampling leads to a situation, where individuals representing optimal objective vectors have been sampled unevenly depending on when each individual entered the population. The following *fair* sampling strategy guarantees that the end all individuals receive about the *same* number of samples.

Alg. 5 implements this strategy by counting the number of offspring each individual produces (line 6). The sampling procedure deterministically chooses the individual which has produced the least number of offspring so far, ties are broken randomly (line 5).

For the analysis of Alg. 5, we focus only on the second phase as the first phase is identical to the simple Alg. 4 described before.

Once the first two optimal objective vectors have been found, there is exactly one possible parent for each of the remaining $n-1$ objective vectors. We are interested in the number of mutations that must be allocated to *each* of these $n-1$ parents in order to have at least one successful mutation each that leads to the desired child. The following lemma provides a lower bound on the probability that a certain number of mutations per parent are sufficient.

Lemma 1 (Minimal success probability [29]). *Let p be the success probability for each single mutation and $c > 0$ an arbitrary constant. With probability at least $1 - n^{1-c}$ all $n-1$ remaining offspring have been constructed in at most $c \cdot 1/p \cdot \log n$ mutation trials for each corresponding parent.*

Now we can translate the number of mutations that are needed into the running time of Alg. 5.

Theorem 3 (Running time bounds of FEMO [29]). *With probability at least* $1 - O(1/n)$ *the number of objective function evaluations* T *Alg. 5 needs from the discovery of the first two optimal solutions until the whole Pareto set has been found lies in the interval* $[1/4 \cdot 1/p \cdot n \log n, 2 \cdot 1/p \cdot n \log n]$. *Hence,* $Prob\{T = \Theta(1/p \cdot n \log n)\} = 1 - O(1/n)$. *Furthermore,* $E(T) = O(1/p \cdot n \log n)$.

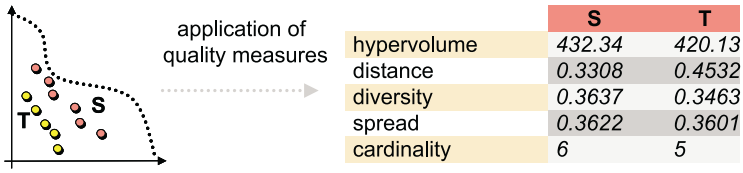
The time to find the first one or two elements of the Pareto set can be neglected and the total running time is mainly determined by Theorem 3. For our case the mutation success probability is $p = 1/n$, which leads to a run-time of $\Theta(n^2 \log n)$. This is a considerable improvement in comparison to any multi-start strategy of a single-objective EA and to SEMO.

5.3 Quality Assessment of Pareto set approximations

The notion of performance includes both the quality of the outcome as well as the computational resources needed to generate this outcome. Concerning the latter aspect, it is common practice to monitor either the number of fitness evaluations or the overall run-time on a particular computer—in this respect, there is no difference between single- and multiobjective optimization. As to the quality aspect, however, there is a difference. In single-objective optimization, we can define quality by means of the objective function: the smaller (or larger) the value, the better the solution. If we compare two solutions in the presence of multiple optimization criteria, the concept of Pareto dominance can be used, though, the possibility of two solutions being incomparable, i.e., neither dominates the other, complicates the situation. However, it gets even more complicated when we compare two sets of solutions because some solutions in either set may be dominated by solutions in the other set, while others may be incomparable. Accordingly, it is not clear what quality means with respect to Pareto set approximations: closeness to the optimal solutions in objective space, coverage of a wide range of diverse solutions, or other properties? It is difficult to define appropriate quality measures for Pareto set approximations, and as a consequence graphical plots have been used to compare the outcomes of MOEAs until recently, as Van Veldhuizen and Lamont point out [40].

Nevertheless, quality measures are necessary in order to compare the outcomes of multiobjective optimizers in a quantitative manner, and meanwhile several quality measures have been proposed in the literature. Certainly, the simplest comparison method would be to check whether an outcome entirely dominates another. The reason, however, why quality measures have been used is to be able to make more precise statements in addition to that, which are inevitably based on certain assumptions about the decision maker's preferences:

- If one algorithm is better than another, can we express how much better it is?



	S	T
hypervolume	*432.34*	*420.13*
distance	*0.3308*	*0.4532*
diversity	*0.3637*	*0.3463*
spread	*0.3622*	*0.3601*
cardinality	*6*	*5*

Fig. 13. Transformation of Pareto front approximations into real vectors by means of unary quality measures

– If no algorithm can be said to be better than the other, are there certain aspects for which we can say the former is better than the latter?

Hence, the key question when designing quality measures is how to best summarize Pareto set approximations by means of a few characteristic numbers—similarly to statistics where the mean, the standard deviation, etc. are used to describe a probability distribution in a compact way. It is unavoidable to lose information by such a reduction, and the crucial point is not to lose the information one is interested in.

Most popular are unary quality measures, i.e., the measure assigns each Pareto set approximation a number that reflects a certain quality aspect, and usually a combination of them is used, e.g., [48, 40, 8]. For instance, the generational distance measure [39] gives the average distance of the objective vectors in the Pareto front approximation under consideration to the closest optimal objective vector, and the hypervolume measure [48] considers the volume of the objective space dominated by a Pareto front approximation. Often, different unary quality measures are combined, and as a result two Pareto set approximations $\boldsymbol{S}, \boldsymbol{T} \subseteq \boldsymbol{X}$ are compared by comparing the corresponding quality measures as depicted in Fig. 13. The question is, though, what statements can be made on the basis of the information provided by these quality measure values. Is it, for instance, possible to conclude from the quality "measurements" that $\boldsymbol{S}$ is undoubtedly better than $\boldsymbol{T}$ in the sense that $\boldsymbol{S}$, loosely speaking, entirely dominates $\boldsymbol{T}$? This is a crucial issue in any comparative study, and implicitly most papers in this area rely on the assumption that this property is satisfied for the measures used.

Recently, Zitzler et al. [49] investigated quality measures from this perspective and proved theoretical limitations of unary quality measures. In particular, they showed that

– there exists no unary quality measure that is able to indicate *whether* a Pareto set approximation $\boldsymbol{S}$ is better than a Pareto set approximation $\boldsymbol{T}$;
– the above statement even holds if we consider a finite combination of unary measures;
– most quality measures that have been proposed to indicate *that* $\boldsymbol{S}$ is better than $\boldsymbol{T}$ at best allow to infer that $\boldsymbol{S}$ is not worse than $\boldsymbol{T}$, i.e., $\boldsymbol{S}$ is better than or incomparable to $\boldsymbol{T}$;

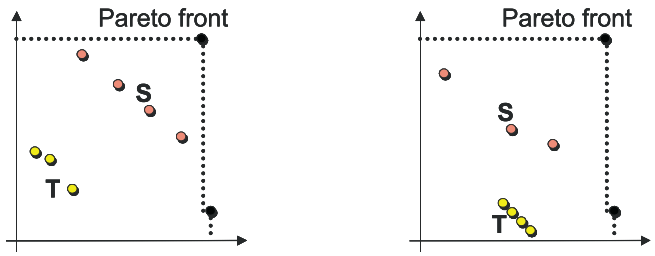


Fig. 14. Two scenarios where a Pareto front approximation entirely dominates another Pareto front approximation

- unary measures being able to detect *that* $\boldsymbol{S}$ is better than $\boldsymbol{T}$ exist, but their use is in general restricted;
- binary quality measures overcome the limitations of unary measures and, if properly designed, are capable of indicating *whether* $\boldsymbol{S}$ is better than $\boldsymbol{T}$.

This means that in general the quality of a Pareto set approximation set cannot be completely described by a (finite) set of distinct criteria such as diversity and distance. Consider, e.g., the following combination of unary quality measures used in [40]: average distance to the Pareto front, diversity of the Pareto front approximation, and the number of vectors in the Pareto front approximation. In Fig. 14, the Pareto set approximation $\boldsymbol{S}$ dominates all solutions of the Pareto set approximation $\boldsymbol{T}$ (the corresponding images in the objective space are shown in the figure). On the left hand side, $\boldsymbol{S}$ is assessed better than $\boldsymbol{T}$ with respect to all of the three quality measures; on the right hand side, $\boldsymbol{T}$ is assigned better quality values. Thus, this quality measure combination does not allow to make any conclusions about whether one outcome is better than another in terms of Pareto dominance.

One possibility to overcome the limitations of unary measure is to use binary quality measures. For instance, a binary ϵ-quality measure can be defined on the basis of the concept of ϵ-Pareto dominance (cf. Def. 1):

Definition 5 (Binary ϵ-quality measure). *Let $\boldsymbol{S}, \boldsymbol{T} \subseteq \boldsymbol{X}$. Then the binary ϵ-quality measure $I_\epsilon(\boldsymbol{S}, \boldsymbol{T})$ is defined as the minimum $\epsilon \in \mathbb{R}$ such that any solution $\boldsymbol{b} \in \boldsymbol{T}$ is ϵ-dominated by at least one solution $\boldsymbol{a} \in \boldsymbol{S}$:*

$$I_\epsilon(\boldsymbol{S}, \boldsymbol{T}) = \min\{\epsilon \in \mathbb{R} \mid \forall \boldsymbol{b} \in \boldsymbol{T}\, \exists \boldsymbol{a} \in \boldsymbol{S} : \boldsymbol{a} \succ_\epsilon \boldsymbol{b}\} \tag{4}$$

In the light of the previous dicussion, this quality measure possesses several desirable features. Whenever $I_\epsilon(\boldsymbol{S}, \boldsymbol{T}) < 1$, we know that all solutions in $\boldsymbol{T}$ are dominated by a solution in $\boldsymbol{S}$; if $I_\epsilon(\boldsymbol{S}, \boldsymbol{T}) = 1$ and $I_\epsilon(\boldsymbol{T}, \boldsymbol{S}) = 1$, then $\boldsymbol{S}$ and $\boldsymbol{T}$ represent the same Pareto front approximation; if $I_\epsilon(\boldsymbol{S}, \boldsymbol{T}) > 1$ and $I_\epsilon(\boldsymbol{T}, \boldsymbol{S}) > 1$, then $\boldsymbol{S}$ and $\boldsymbol{T}$ are incomparable, i.e., both contain solutions not dominated by the other set. Furthermore, the I_ϵ measure represents a natural extension to the evaluation of approximation schemes in theoretical computer science [12] and

gives the factor by which an outcome is worse than another. In addition to that, it is cheap to compute. However, there may be particular scenarios where the ϵ-quality measure is not appropriate.

6 A Text-based Interface for Search Algorithms

6.1 Current Situation

As discussed in Section 3, current MOEAs use increasingly complex operators. Re-implementing these algorithms for each usage scenario becomes time-consuming and error-prone. Mainly two groups are affected by this problem:

- Application engineers who need to choose, implement, and apply state-of-the-art algorithms without in-depth programming knowledge and expertise in the optimization domain.
- Developers of optimization methods who want to evaluate algorithms on different test problems and compare a variety of competing methods.

There is a clear need for a method to provide and distribute ready-to-use implementations of optimization methods and ready-to-use benchmark and real-world problems. These modules should be freely combinable. Since the above-mentioned issues are not constrained to evolutionary optimization a candidate solution should be applicable to a broader range of search algorithms.

Programming libraries have been designed to facilitate the implementation of optimization algorithms. They are usually geared to a particular technique, e.g., evolutionary algorithms, and provide reusable and extendible program components that can be combined in different ways. If a specific optimization method is to be tailored to a specific application and knowledge in both algorithm domain and application domain is available, then these libraries provide valuable tools to reduce the programming effort. However, still considerable implementation work is necessary, if we intend to test various algorithms on a certain application or apply a specific algorithm to different test problems. Furthermore, programming libraries require a certain training period, and their use is restricted to specific programming languages and often also to specific computing platforms.

A different approach, called PISA (A Platform and programming language independent Interface for Search Algorithms), was presented in [3]. The underlying concept is discussed in the following section.

6.2 Concept of PISA

The basic idea is to divide the implementation of an optimization method into an algorithm-specific part and an application-specific part as shown in Fig. 15. The former contains the selection procedure, while the latter encapsulates the representation of solutions, the generation of new solutions, and the calculation of objective function values. All problem-specific operators reside in the application part, which is called variator; the problem-independent part is called selector.

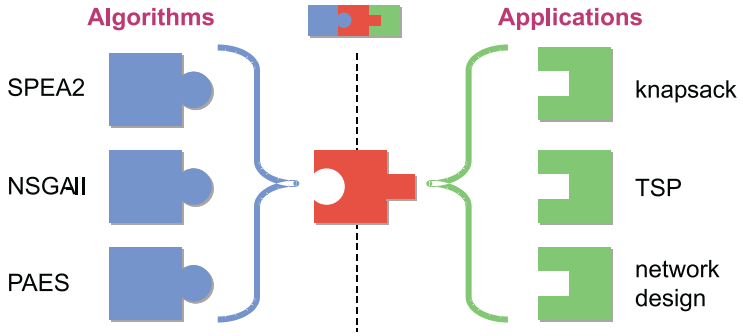


Fig. 15. Illustration of the concept underlying PISA. The optimizers on the left hand side and the applications on the right hand side are examples only and can be replaced arbitrarily. Each optimizer can be connected to each application.

These two parts are realized by distinct programs that communicate via a text-based interface. Selector and variator programs can be compiled independently and distributed as binaries and they are freely interchangeable as long as they adhere to the interface specification.

6.3 Implementation of PISA

As mentioned in the previous section the variator performs all operations which are specific to the optimization problem. It starts by generating a initial population and calculating the objective values for each individual. It then writes the IDs of all individuals and the corresponding objective vectors to a file. Since the selector is problem-independent this data is sufficient for the selection operator. The selector reads this file and chooses a collection of promising parent individuals and writes their IDs to another file. The variator generates the offspring by variating these parents. The IDs and objective vectors of the offspring are again communicated to the selector. This loop continues until some termination criterion is met.

The synchronization of the two programs is achieved through a handshake protocol. A common state variable is written to a text file which both processes regularly read. When one process is finished with an optimization step it updates the state variable, thus signaling to the other module that the latter can perform the next step. This simple scheme ensures that only one process is active at the time and data files are only read after they have been completely written.

6.4 Benefits and Limitations

Since all communication between the selector and the variator is established through text files the only requirement for combining two modules is that both of them have access to the same file system. The two modules can be programmed in different programming languages and can run on different machines and even on different platforms. Nevertheless it is easy to add the interface functionality to

an existing algorithm or application since the whole communication only consists of a few text file operations.

As a negative consequence, the data transfer introduces an additional overhead into the optimization. Since only IDs and objective values are exchanged the amount of data is minimal. In practically relevant applications this overhead is small compared to the intrinsic run-time of the optimization [3]. Another drawback is that the interface poses certain limitations on the structure of the optimization process. Most but not all evolutionary algorithms and many others (e.g. simulated annealing, tabu search) fit into the proposed scheme. Furthermore, the file format leaves room for extensions so that particular details such as diversity measures in decision space can be implemented on the basis of PISA.

As mentioned, PISA allows to use pre-compiled, ready-to-use executable files, which, in turn, minimizes the implementation overhead and avoids the problem of implementation errors. As a result, an application engineer can easily exchange the optimization method and try different variants, while an algorithm designer has the opportunity to test a search algorithm on various problems without additional programming effort (cf. Fig. 15). Certainly, this concept is not meant to replace programming libraries. It represents a complementary approach that allows to build collections of optimizers and applications, all of them freely combinable across computing platforms.

Crucial, though, for the success of the proposed approach is the availability of optimization algorithms and applications compliant with the interface. To this end, the authors maintain a website at `http://www.tik.ee.ethz.ch/pisa/` which contains example implementations for download.

7 Conclusions

Optimization problems involving multiple objectives are common. In this context, evolutionary computation represents a valuable tool, in particular

- if we would like to be flexible with respect to the problem formulation,
- if we are interested in approximating the Pareto set, and
- if the problem complexity prevents exacts methods from being applicable.

Flexibility is important if the underlying model is not fixed and may change or needs further refinement. The advantage of evolutionary algorithms is that they have minimum requirements regarding the problem formulation; objectives can be easily added, removed, or modified. Moreover, due the fact that they operate on a set of solution candidates, evolutionary algorithms are well-suited to generate Pareto set approximations. This is reflected by the rapidly increasing interest in the field of evolutionary multiobjective optimization. Finally, it has been demonstrated in various applications that evolutionary algorithms are able to tackle highly complex problems and therefore they can be seen as an approach complementary to traditional methods such as integer linear programming.

Acknowledgment

This work has been supported by the Swiss National Science Foundation (SNF) under the ArOMA project 2100-057156.99/1 and the SEP program at ETH Zurich under the poly project TH-8/02-2.

References

1. T. Bäck, U. Hammel, and H.-P. Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, 1997.
2. S. Bleuler, M. Brack, L. Thiele, and E. Zitzler. Multiobjective genetic programming: Reducing bloat by using SPEA2. In *Congress on Evolutionary Computation (CEC-2001)*, pages 536–543, Piscataway, NJ, 2001. IEEE.
3. S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA - a platform and programming language independent interface for search algorithms. In C. M. Fonseca et al., editors, *Second International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)*, pages 494–508, Berlin, Germany, 2003. Springer.
4. C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer, New York, 2002.
5. D. W. Corne, J. D. Knowles, and M. J. Oates. The pareto envelope-based selection algorithm for multiobjective optimisation. In M. Schoenauer et al., editors, *Parallel Problem Solving from Nature (PPSN VI)*, pages 839–848, Berlin, 2000. Springer.
6. K. Deb. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation*, 7(3):205–230, 1999.
7. K. Deb. *Multi-objective optimization using evolutionary algorithms*. Wiley, Chichester, UK, 2001.
8. K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In M. Schoenauer et al., editors, *Parallel Problem Solving from Nature (PPSN VI)*, pages 849–858, Berlin, 2000. Springer.
9. K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization test problems. In D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, editors, *Congress on Evolutionary Computation (CEC)*, pages 825–830, 2002.
10. S. Droste, T. Jansen, and I. Wegener. A rigorous complexity analysis of the (1+1) evolutionary algorithm for separable functions with Boolean inputs. *Evolutionary Computation*, 6(2):185–196, 1998.
11. S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276(1–2):51–81, 2002.
12. T. Erlebach, H. Kellerer, and U. Pferschy. Approximating multi-objective knapsack problems. In F. K. H. A. Dehne et al., editors, *Workshop on Algorithms and Data Structures (WADS 2001)*, pages 210–221, Berlin, Germany, 2001. Springer.
13. C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, San Mateo, California, 1993. Morgan Kaufmann.

14. M. P. Fourman. Compaction of symbolic layout using genetic algorithms. In J. J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 141–153, Pittsburgh, PA, 1985. sponsored by Texas Instruments and U.S. Navy Center for Applied Research in Artificial Intelligence (NCARAI).
15. D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.
16. P. Hajela and C.-Y. Lin. Genetic search strategies in multicriterion optimal design. *Structural Optimization*, 4:99–107, 1992.
17. T. Hanne. On the convergence of multiobjective evolutionary algorithms. *European Journal Of Operational Research*, 117(3):553–564, 1999.
18. T. Hanne. Global multiobjective optimization with evolutionary algorithms: Selection mechanisms and mutation control. In E. Zitzler et al., editors, *Evolutionary Multi-criterion Optimization (EMO 2001), Proc.*, Lecture Notes in Computer Science Vol. 1993, pages 197–212, Berlin, 2001. Springer.
19. S. Helbig and D. Pateva. On several concepts for ϵ-efficiency. *OR Spektrum*, 16(3):179–186, 1994.
20. J. Horn, N. Nafpliotis, and D. E. Goldberg. 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, pages 82–87, Piscataway, NJ, 1994. IEEE Press.
21. H. Ishibuchi and T. Murata. Multi-objective genetic local search algorithm. In *Proceedings of 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, pages 119–124, Piscataway, NJ, 1996. IEEE Press.
22. A. Jaszkiewicz. On the performance of multiple-objective genetic local search on the 0/1 knapsack problem - a comparative experiment.
23. J. D. Knowles. *Local-Search and Hybrid Evolutionary Algorithms for Pareto Optimization*. PhD thesis, Department of Computer Science, University of Reading, UK, 2002.
24. J. D. Knowles and D. W. Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Congress on Evolutionary Computation (CEC99)*, volume 1, pages 98–105, Piscataway, NJ, 1999. IEEE Press.
25. F. Kursawe. A variant of evolution strategies for vector optimization. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, pages 193–197, Berlin, 1991. Springer.
26. M. Lahanas, N. Milickovic, D. Baltas, and N. Zamboglou. Application of multiobjective evolutionary algorithms for dose optimization problems in brachytherapy. In E. Zitzler, K. Deb, L. Thiele, C. A. C. Coello, and D. Corne, editors, *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001)*, volume 1993 of *Lecture Notes in Computer Science*, pages 574–587, Berlin, 2001. Springer-Verlag.
27. M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multi-objective optimization. *Evolutionary Computation*, 10(3):263–282, 2002.
28. M. Laumanns, L. Thiele, E. Zitzler, and K. Deb. 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, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 439–447, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
29. M. Laumanns, L. Thiele, E. Zitzler, E. Welzl, and K. Deb. Running time analysis of multi-objective evolutionary algorithms on a simple discrete optimization problem. In *Parallel Problem Solving From Nature — PPSN VII*, 2002.
30. S. Mostaghim, J. Teich, and A. Tyagi. Comparison of data structures for storing pareto-sets in moeas. In *IEEE Proceedings, World Congress on Computational Intelligence (CEC 2002)*, pages 843–849, Honolulu, USA, May 12-17 2002.
31. G. T. Parks and I. Miller. Selective breeding in a multiobjective genetic algorithm. In A. E. Eiben et al., editors, *Parallel Problem Solving from Nature – PPSN V*, pages 250–259, Berlin, 1998. Springer.
32. G. Rudolph. *Convergence Properties of Evolutionary Algorithms.* Verlag Dr. Kovač, Hamburg, 1997.
33. G. Rudolph. Evolutionary search for minimal elements in partially ordered sets. In *Evolutionary Programming VII – Proc. Seventh Annual Conf. on Evolutionary Programming (EP-98)*, San Diego CA, 1998. The MIT Press, Cambridge MA.
34. G. Rudolph and A. Agapie. Convergence properties of some multi-objective evolutionary algorithms. In A. Zalzala and R. Eberhart, editors, *Congress on Evolutionary Computation (CEC 2000)*, volume 2, pages 1010–1016, Piscataway, NJ, 2000. IEEE Press.
35. J. D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 93–100, Pittsburgh, PA, 1985. sponsored by Texas Instruments and U.S. Navy Center for Applied Research in Artificial Intelligence (NCARAI).
36. J. Scharnow, K. Tinnefeld, and I. Wegener. Fitness landscapes based on sorting and shortest paths problems. In *Parallel Problem Solving From Nature — PPSN VII*, 2002.
37. B. W. Silverman. *Density estimation for statistics and data analysis.* Chapman and Hall, London, 1986.
38. N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.
39. D. A. Van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations.* PhD thesis, Graduate School of Engineering of the Air Force Institute of Technology, Air University, June 1999.
40. D. A. Van Veldhuizen and G. B. Lamont. On measuring multiobjective evolutionary algorithm performance. In A. Zalzala and R. Eberhart, editors, *Congress on Evolutionary Computation (CEC 2000)*, volume 1, pages 204–211, Piscataway, NJ, 2000. IEEE Press.
41. I. Wegener. Methods for the analysis of evolutionary algorithms on pseudo-boolean functions. Technical Report CI-99/00, SFB 531, Universität Dortmund, 2000.
42. I. Wegener. Theoretical aspects of evolutionary algorithms. In *ICALP 2001*, volume 2076 of *LNCS*, pages 64–78. Springer-Verlag, 2001.
43. E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications.* Ph.D. thesis, Shaker Verlag, Aachen, Germany, 1999.
44. E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.
45. E. Zitzler, K. Deb, L. Thiele, C. A. C. Coello, and D. Corne, editors. *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*

(EMO 2001), volume 1993 of *Lecture Notes in Computer Science*, Berlin, Germany, March 2001. Springer.

46. E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In K. Giannakoglou, D. Tsahalis, J. Periaux, K. Papailiou, and T. Fogarty, editors, *Evolutionary Methods for Design, Optimisation, and Control*, pages 19–26, Barcelona, Spain, 2002. CIMNE.
47. E. Zitzler, J. Teich, and S. S. Bhattacharyya. Multidimensional exploration of software implementations for DSP algorithms. *Journal of VLSI Signal Processing*, 24(1):83–98, February 2000.
48. E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.
49. E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.