Application Issues for Multiobjective Evolutionary Algorithms

Thomas Hanne
Fraunhofer Institute for Industrial Mathematics (ITWM)
Department of Optimization
Gottlieb-Daimler-Str. 49
67633 Kaiserslautern
Germany
Email: hanne@itwm.fhg.de

Abstract: Various issues of the design and application of multiobjective evolutionary algorithms to real-life optimization problems are discussed. In particular, questions on problem-specific data structures and evolutionary operators and the determination of method parameters are treated. Three application examples in the areas of constrained global optimization (electronic circuit design), semi-infinite programming (design centering problems), and discrete optimization (project scheduling) are discussed.

Keywords: multiobjective optimization, Pareto set, evolutionary algorithm, discrete optimization, continuous optimization, electronic circuit design, semi-infinite programming, scheduling

1.Introduction

Let us start with introducing some basic terminology. Usually, we consider an optimization problem defined by

$$\min_{a \in A} f(a)$$

where the objective function f is defined by

$$f : R^n \to R^q, q \geq 1.$$

In the case q > 2, we talk about multiobjective optimization while q=1 corresponds to a usual (scalar) optimization problem.
We assume that the set of feasible solutions A is defined by restrictions as follows:

$$A = \left\{ a \in R^n : g_j(a) \leq 0,\ j \in \{1,..,\ m\} \right\}$$

Each restriction function g is defined as

$$g_j : R^n \to R$$

In Evolutionary Algorithms (EAs) we are dealing with populations of “entities” which correspond to solutions. Let us assume for simplicity that parent and offspring solutions are given as follows:

$$M^t = \left\{ a_1^t,..., a_\mu^t \right\} \subseteq A$$
$$N^t = \left\{ b_1^t,..., b_\lambda^t \right\} \subseteq A$$

M^t is the parent population in generation t which is assumed to consist of μ entities.
N^t is the offspring population in generation t which is assumed to consist of λ entities.

In actual implementations of EAs, entities are representations of solutions but possibly include other data additionally. With respect to the actual usage in computers, Genetic Algorithms (GA), for instance, use fixed-size bit strings for encoding the entities (see, e.g., Holland (1975)). In Evolution Strategies (ES) fixed numbers of floating-point variables are used (see, e.g., Schwefel (1981), Schwefel (1995)). In Genetic Programming (GP) the entities are programs of variable size (typically in LISP). The term Evolutionary Algorithm (EA) is used as a general expression for describing any kind of algorithm simulating natural evolution and using arbitrary (problem-specific) data structures (see also Heitkötter and Beasley (2000), Michalewicz (1998), Bäck, Fogel, and Michalewicz (1997)).

The general algorithmic framework of EAs is usually similar to the following pseudo code:

1: Initialize starting population M^0 .
2: Initialize control parameters; t:=0.
3: Copy & mutate N^t from M^t.
4: Recombine N^t.
5: Evaluate fitness of N^t and M^t .
6: Select M^{t+1} from $N^t \cup M^t$.
7: If stopping criterion fulfilled then stop.
8: t:=t+1; goto 3.

Thus, an EA basically consists of a generational loop producing offspring solutions from parent solutions using some variation principles and selecting new parent solutions according to their fitness.

2. What's Special in Evolutionary Multiobjective Optimization?

From the viewpoint of traditional evolutionary algorithms, the vector-valued nature of the objective function requires some special attention. Since the objective function is usually evaluated only for the fitness calculation in the selection step of an EA, only this step requires some adaptation when several objectives are to be considered. We will come back to these modifications below.

From the viewpoint of traditional Multiple Criteria Decision Making (MCDM), the particularities of evolutionary multiobjective optimization require a more comprehensive discussion (see, e.g., Hanne (2001a) for further references). The main question here is not, how the algorithm works in details but what the result of the algorithm should be. In the huge research field of MCDM, most of the considered methods aim at selecting a "compromise solution" from the set of feasible ones. Usually, this solution should be efficient (Pareto-optimal or nondominated) or fulfill some other axioms of rationality.

Multiobjective Evolutionary Algorithms (MOEAs) on the other hand try to calculate a good approximation and representation of the efficient set, typically for hard-to-solve

combinatorial or nonlinear optimization problems. For some multiobjective optimization problems such as multiobjective linear optimization or some kinds of discrete optimization problems, effective algorithms for calculation the complete and accurate efficient set are well-known in the MCDM community.

Since usually decision makers do not care much about an approximation of the efficient set (or other complex solution sets) but want to select a single solution at the end, a typical scenario for applying MOEAs together with traditional approaches would be as follows: A traditional MCDM method such as a reference point approach, a utility function-based method, an outranking approach, etc. is applied *after* using the MOEA (aposteriori approach). For a comprehensive survey on MOEAs, we refer to the recent monographs by Coello Coello et al. (2003) and Deb (2001).

2.1 Multiobjective Selection

There are various possibilities for considering multiple objectives in the selection step: A straightforward idea, intuitively used long before MCDM was invented, is to aggregate the several objective values to a single one. This proceeding is also known as scalarization and the simplest way of doing so is by building a (possibly weighted) sum.
Frequently, the scalarization leads to problems with representing nonconvex efficient sets (see, e.g., Fonseca, and Fleming, 1995). Therefore, the idea came up that the Pareto order only should be used for selection.
For instance, in the dominance level or rank approach, all solutions from a population set, which are nondominated within that set, are assigned the value 0 (and treated equally for the selection). For the remaining solutions, the nondominated ones are assigned the value 1 and so on. The dominance grade approach works similar. In that approach each alternative is assigned for fitness evaluation the number of solutions, which dominate it. Thus, also here all solution being efficient with respect to the current population are assigned the value 0.

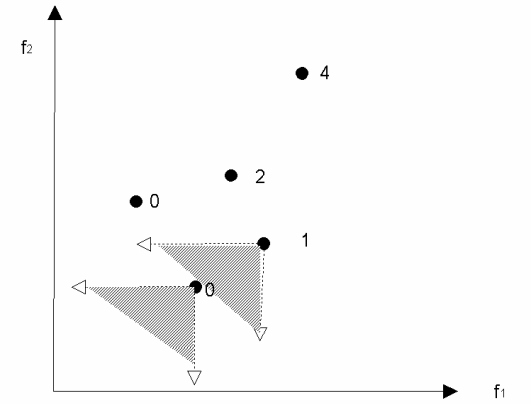

Fig. 1: Dominance grades of some solutions in the biobjective case.

Both approaches show a low discrimination among alternatives (i.e. many alternatives are efficient with respect to a particular population) when the MOEA reaches a mature state (see, e.g., Hanne (2001c)). This is, however, a problem, which can hardly be avoided since usually in continuous multiobjective optimization problems and also in many discrete ones, the set of Pareto-optimal solutions is large (or even infinite).

3. Why Evolutionary Algorithms?

A first question before implementing or using evolutionary algorithms for a given multiobjective optimization problem is to ask why this class of methods should be applied and not one of the many other available or proposed methods. As mentioned above, for some classes of problems there are efficient algorithms available which calculate the exact solution (i.e. not just some approximation) within a usually acceptable amount of time. On the other hand, there are many other methods available today which may be used for calculating approximate solutions, for instance methods from the field called metaheuristics. In general, it is not possible to say which method may be best for given unstudied optimization problem, especially for "non-standard" optimization problems, which are frequent in real-life applications. Considering the fact that in daily life, there is not enough time to comprehensively analyze the effectiveness (time consumption, exactness of solutions, etc.) of a method, the question of method choice remains ad-hoc up to a certain degree (see Hanne (2001a) for a deeper treatment of this issue).

Therefore let us just discuss a few characteristics of evolutionary algorithms, which let them appear to be attractive for being used for multiobjective optimization problems. Of course, these features may not be valid for any kind of optimization problem while, on the other hand, also other methods remain competitive.

3.1 Robustness

Robustness is usually considered as the most important reason for using evolutionary algorithms. There are two interpretations of robustness: On the one hand, it means that for a large class of problems rather good solutions are calculated. On the other hand, it is assumed that the obtained solutions are rather stable with respect to minor modifications (or perturbations) of the problem. This aspect concerns also the sensitivity of the problem, i.e. the question which kind of perturbations of the problem lead to what changes of the solution set.

One of the reference studies with respect to the robustness of evolutionary algorithms (evolution strategies in that case) is the computational comparison by Schwefel (1981), which showed that Evolution Strategies performed best from a set of nonlinear optimization methods using a diverse sample of test problems.

3.2 Speed

Usually, evolutionary algorithms are not considered to be particularly fast. For some classes of problems with established specific optimization procedures, this is certainly true. On the other hand, evolution strategies showed an average performance with respect to quadratic optimization problems (see Schwefel (1981)) in a comparison with more specialized methods, in particular nonlinear optimization methods which take advantage of using first and/or second derivatives of the objective function. This result should put into perspective the opinion that

evolutionary algorithms should only be used where information on derivatives etc. is not available.

3.3. Ease of use

Compared with many other optimization methods, EAs are rather easy to implement and to use. There is not much knowledge required about the handling of derivative information, optimality conditions, or numerical issues. It is possible to start implementation with a rough prototype (see below) that does not apply sophisticated mutation, recombination, or selection routines. On the other hand, refinement of the algorithm may be challenging and time consuming. The degree of re-use may be smaller than in the case of some traditional optimization methods.

4. How to Start?

4.1. The Application-Specific Development Process for EAs

The implementation and application of evolutionary algorithms to a given optimization problem can be considered as a regular software development process. This type of process can be described by the so-called waterfall model. This model assumes various stages of the process through which the product, the algorithm, streams towards its application. Usually these stages are as follows: requirements analysis, design, implementation, validation & verification, operation & maintenance (see Fig. 2).

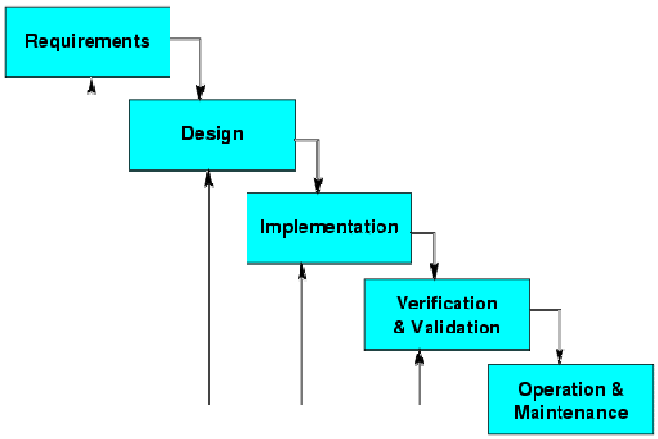

Fig. 2: Waterfall model of a software development process.

Applied to evolutionary algorithms, the development process is typically similar to the following:

Phase I (Prototype)

- Find an appropriate representation (data structures) of solutions
- Implement objective function(s) and deal with infeasibility (see below)
- Implement the handling of a population
- Implement simple variation operator(s)
- Implement selection

Phase II (Refinement)

- Refine evolutionary operators
- Implement problem-specific adaptations
- Experiment with parameter(s)
- Use specialized subroutines

In the following, three specific questions on the application-specific development and operation of EAs are discussed, the choice of parameter values, the refinement of evolutionary operators, and the handling of infeasible solutions.

4.2. How to Set the Parameters?

One of the most prominent questions in using EAs for a specific problem is the following one: How should the parameter values of the algorithm as, for instance, the population size, the mutation rates, or the recombination probability be set? If nothing is known about good parameter values (i.e. values leading better solutions or reaching them in less time), one may start using typical values for parameters. Suggestions for parameter values can be found in the literature, for instance:

- Mutation probability (GA): 0.001 - 0.01
- Mutation step sizes initialized with step sizes (sigma values) being 10% of the starting point
- Mutation step sizes vary by 10% on average per generation (see, e.g., Hanne (2001b))
- Probability of recombination: 0.25 - 0.9
- Number of parents: 10-200
- Number of offspring: 10-200

Frequently, such default parameters do not lead to the desired success: Either the obtained solutions are not good enough or the algorithms take too much time. In that case, experimenting with the parameter values in a trial and error fashion might be a simple but effective way for improving the performance. Systematic experiments may be another way for finding better parameter values. Last but not least let us mention that the problem of determining parameter values may be defined as an optimization problem itself (meta optimization problem) that may be solved, for instance, by another evolutionary algorithm (meta EA). This idea is explained in more details in Hanne (2001a).

4.3. Advanced Evolutionary Operators

There are various reasons why more advanced evolutionary operators should be used. On the one hand, such operators may make the evolution process more realistic, more similar to the natural evolution. On the other hand, and this aspect is more relevant for optimization application, the solution process may be improved. In particular, the process may be sped up and/or the quality of solutions (e.g. diversity) may be improved. A frequent goal is to allow for a better adaptation to particularities of a problem. In the next subsection, we discuss a specific reason for problem-specific adaptations of evolutionary operators. Further below, in Section 5, some examples of adaptations are discussed.

4.4 The Problem of Infeasibility

During the “data variation steps”, mutation and recombination, it may occur that generated offspring solutions are not feasible, i.e. either the data does not correspond to variable values with respect to a given encoding or the variable values do not belong to the feasible set A.

In that case, there are various possibilities to react. A very simple one is to redo the variation step, i.e. to generate new solutions until enough feasible ones are obtained. Since the probability of obtaining infeasible solutions may be high, in particular when A is defined by many restrictions, this strategy may be costly with respect to time consumption.

Another frequent approach is that of using a repair operator. Here, the idea is to continue with the generated infeasible solution and to map it to a feasible one. Sometimes, there is no canonical way for doing so. In such cases, one may think about using a punishment function. That function punishes the generation of infeasible (or almost infeasible) solutions by deteriorating their fitness values. The stronger the restrictions are violated by a solution, the more the objective values are to be deteriorated. In that case, it may be possible to force subsequently generated offspring (offspring of the infeasible offspring) back to the feasible domain.

However, the best general advice with respect to infeasibility might be, that one should avoid it by using an appropriate encoding. Occasionally, more intelligent data structures may avoid the infeasibility of solutions at all.

5.Three Examples from Recent Projects

In the following, we would like to discuss concisely three real-life application examples of multiobjective evolutionary algorithms. These application examples are conducted in the context of real-life problems studied during the past two years. Each example shows different particularities of the MOEA design, implementation, and usage.

5.1 Design of Electronic Circuits

During a recent project we studied the problem of determining parameters for an electronic circuit model (as given by a circuit simulation software such as Cadence) for approximating the behavior of a real-life circuit. Depending on the frequency of input, a real-life circuit deviated more or less from its idealistic model described by a small number of elements. The behavior of the circuit can be described by a complex-valued matrix y. For analyzed problems the dimension of y was 2x2. For comparing the result values for the real circuit (given as data file) and the circuit model the deviations of the y-values were considered separately for the real and the imaginary part of the matrix coefficients. The deviations were summed up over a range of frequency values. In this way, 8 or 6 (using some symmetry in y) objective values were calculated for a given setting of parameters.

For the considered problem it was assumed that there were feasible intervals for each of the parameters while no other restrictions were to be observed. This made the feasibility check for new solutions (obtained by mutations) rather simple. For each new component, a comparison with the lower and upper bounds had to be performed. In case of violation, using the bounds as truncation values repaired the values. Recombination among the offspring entities could not lead to infeasible solutions.

Another main advantage of knowing the bounds was that they could be used for scaling parameter-specific mutation rates. If not knowing such intervals it would be hard to find mutation rates, which for instance work for intervals between 10^{-3} and 10^{-2} on the one hand, and 10^{-11} and 10^{-12} on the other hand.

Another major advantage of the bounds was that they could be used for scaling the parameter components. Random starting solutions could easily be generated by using parameter values uniformly distributed between the lower and upper bounds. The knowledge of the feasible parameter ranges could additionally be used for an enforced convergence towards the efficient frontier. For that reason, a modified 1/5 rule (see Schwefel, 1981) was implemented which adapted the mutation rates periodically during the run of the EA. Usually, these modifications led to decreases of the parameter-specific mutation rates such that a rapid convergence towards locally efficient solutions was supported.

5.2. Design centering problems

Another continuous multiobjective optimization problem analyzed by evolutionary algorithms belongs to the class of design centering problems. The considered problems are special generalized semi-infinite optimization problems. In contrast to “usual” nonlinear optimization problem, there is an infinite set of restrictions, which can be described by a finite set of restrictions on the restrictions:

$$M = \{x \in R^n \mid g_i(x, y) \leq 0 \forall y \in Y(x), i = 1,.., q\}$$

$$Y(x) = \{y \in R^m \mid v_l(x, y) \leq 0, l = 1,.., s\}$$

In our case we considered the problem of volume maximization of a gemstone cut for a given raw stone. Alternative objective functions on M result from various measures considering other criteria than the volume (e.g. shape of the design, deviation from ideal proportions) relating to the expected price of the gemstone. The restrictions can be interpreted as follows: A body (cut gemstone) described by some interdependent restrictions Y should be embedded in another body (the raw stone) described by measurement date, e.g. in the form of an STL file.

For treating the problem by an EA we can distinguish two types of restrictions: Design restriction, which can be treated by a repair mechanism and container restrictions, which are considered by a punishment approach. Design restrictions are usually in the form of lower and upper bounds for specific proportions (e.g. the proportion between pavilion height and girdle diameter) of the cut gemstone. Mutations violating these proportions can be repaired by using the bounds as cut-off

values. For the container restriction there is no simple repair mechanism available. A punishment function approach showed good performance in the given problem and allowed the handling of a large number of these restrictions (depending on the resolution of measurement data, there may be some 100000 of these restrictions).

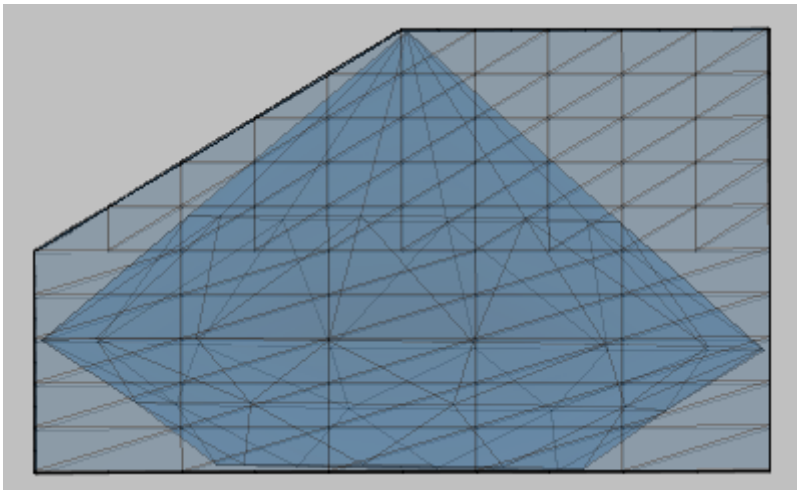

Fig. 3: Gemstone design embedded in a container.

5.3. Project Scheduling

In contrast to the two application examples discussed above, scheduling problems belong to the class of combinatorial optimization problems, i.e. significant decision variables $a \in A$ are discrete instead of continuous. Evolutionary algorithms have been originally developed according to a strict typing of the supported decision variables, for instance genetic algorithms for bit string encoding and evolution strategies for floating point numbers. Therefore, project scheduling is a good example to demonstrate that such a focus on specific data structures is inappropriate for many real-life problems.

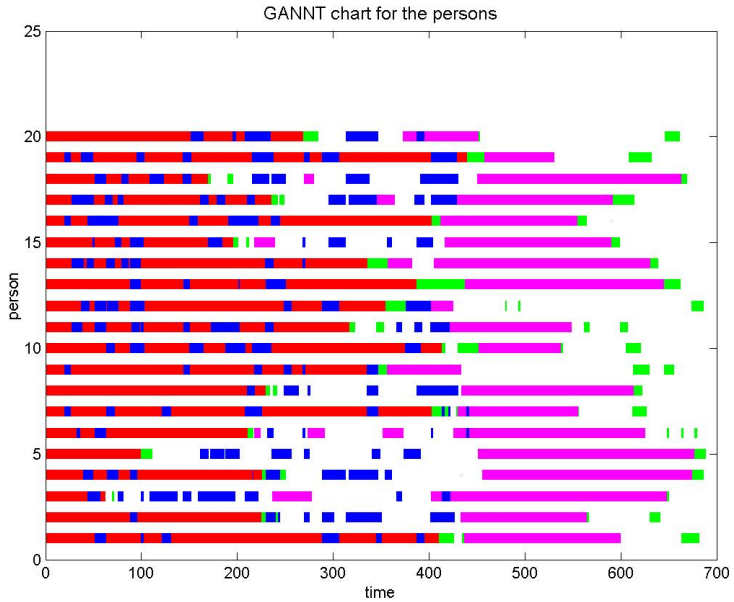

Fig. 4: Person schedule for various interdependent activities leading to waiting times.

Based on our experiences with the modeling and simulation of complex projects such as software development processes, data structures supporting both discrete and continuous variables have been used together with subordinate procedures for scheduling, for instance based on priorities. Some of the decision variables are continuous by nature, for instance the durations of specific activities that influence the quality of the outcome while other decision variables are “artificially” continuous. For instance, real-valued priorities are used to determine the sequence of equally possibly activities. Other decision variables of the problem (e.g. task assignments) are discrete.

A subordinate scheduling heuristics is used to evaluate the priority values and generates a concrete schedule. Some types of activities are scheduled without using any information subject to the evolutionary algorithms but applying a simple first come-first served rule. The mixture of various scheduling concepts led to a significant improvement of the speed of the MOEA and the obtained solutions. Three optimization criteria had been considered in these studies: the makespan, the costs, and the quality of project results (expected no. of defects of the software artifacts). For more details on this application see Hanne and Nickel (2004). Another application of MOEAs to the scheduling of construction projects is discussed in Hanne (2005a).

6. Conclusions

In this paper, we wanted to emphasize that frequently the usage of standard multiobjective evolutionary algorithms is not possible or is insufficient for solving real-life problems. The usage of problem-specific data structures and evolutionary operators is a major issue when developing MOEAs for a specific application.

Additional methods are often required for improving the performance of MOEAs. In particular, the speed of the algorithm is often a problem in practice, especially when some on-line work with a decision support system is required. For instance, clustering methods, hybridizations with other (meta) heuristics and optimization techniques, or databases may be used in such cases.

A main question is however: What happens after having found an approximation of the Pareto set? The implementation of adequate techniques for supporting a decision maker in selecting an efficient solution is often as important as generating suitable candidate solutions (see, e.g., Trinkaus and Hanne (2005)). A novel approach for combining an evolutionary generation of the efficient set and interactive decision support is discussed in Hanne (2005b).

7. References

Bäck, T., D.B. Fogel, Z. Michalewicz (Eds) (1997): Handbook of Evolutionary Computation, Oxford University Press.

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

Deb, K. (2001): Multi-Objective Optimization Using Evolutionary Algorithms.

Wiley, Chichester.

Fonseca, C.M., Fleming, P.J. (1995). An overview of evolutionary algorithms in multiobjective optimization, Evolutionary Computation 3 (1) 1-16.

Hanne, T. (2001a): Intelligent Strategies for Meta Multiple Criteria Decision Making. Kluwer, Boston.

Hanne, T (2001b): Selection and mutation strategies in evolutionary algorithms for global multiobjective optimization. Evolutionary Optimization 3, 1, 27-40

Hanne, T. (2001c): Global multiobjective optimization with evolutionary algorithms: Selection mechanisms and mutation control. Zitzler, E. et al. (Eds.). Evolutionary Multi-Criterion Optimization, First International Conference, EMO 2001, Zurich, Switzerland, March 2001, Proceedings. Springer, Berlin, 197-212.

Hanne, T. (2005a): On the scheduling of construction sites using single- and multiobjective evolutionary algorithms. Proceedings of MIC2005: The Sixth Metaheuristics International Conference. Wien.

Hanne, T. (2005b): Interactive decision support based on multiobjective evolutionary algorithms. Proceedings of Operations Research 2005, International Scientific Annual Conference, Bremen, 7.-9. September 2005

Hanne, T., S. Nickel (2004): A multi-objective evolutionary algorithm for scheduling and inspection planning in software development projects. Feature Issue of European Journal of Operational Research on Scheduling with Multiple Objectives. In Press, Corrected Proof available online 1 September 2004.

Heitkötter, J., D. Beasley (Eds.) (2000): The Hitch-Hiker's Guide to Evolutionary Computation, Issue 8.1, 29 March 2000, http://www.cs.bham.ac.uk/Mirrors/ftp.de.uu.net/EC/clife/www/

Holland, J.H. (1975): Adaptation in natural and artificial systems. University of Michigan Press 1975

Michalewicz, Z. (1998): Genetic Algorithms + Data Structures = Evolution Programs, Third revised and extended edition. Springer

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

Schwefel, H.-P. (1995): Evolution and Optimum Seeking. Wiley,

Trinkaus, H. L., T. Hanne (2005): knowCube: a Visual and Interactive Support for Multicriteria Decision Making. Computers & Operations Research 32, 1289-1309.