Median-Selection for Parallel Steady-State Evolution Strategies

Jürgen Wakunda and Andreas Zell

University of Tübingen, Computer Science Dept.
Köstlinstr. 6, D-72074 Tübingen
{wakunda,zell}@informatik.uni-tuebingen.de

Abstract. We describe a new selection scheme for steady-state evolution strategies, *median selection*. In steady-state algorithms, only one individual is generated and evaluated at each step and is immediately integrated into the population. This is especially well suited for parallel fitness evaluation in a multiprocessor environment. Previous steady-state selection schemes resembled $(\mu + \lambda)$ selection, which has a disadvantage in self-adaptation of the mutation step length. Median selection is similar to (μ, λ) selection. Median selection is compared with other steady-state selection schemes and with (μ, λ) selection on a uniprocessor and on a multiprocessor. It achieves equally good or better results as the best other selection scheme for a number of benchmark functions.

1 Introduction

Evolution Strategies (ES) were developed by Ingo Rechenberg and Hans-Paul Schwefel (Rechenberg, 1973, Schwefel, 1977) for multi-dimensional, real-valued parameter optimization problems. The most important property of ES is the ability of self-adaptation. With self-adaptation the variance of mutations on the object variables are adapted at runtime and thus the optimization progress is improved. Methods for self-adaptation are for example the 1/5-success-rule, mutative step control (Rechenberg, 1994), derandomized step control (Ostermeier et al., 1993) or covariance matrix adaptation (CMA) (Hansen and Ostermeier, 1996). The selection method plays an important role for self-adaptation. In (Schwefel, 1992) it is shown, that regarding speed of self-adaptation, comma selection is superior to plus selection (see next section). The $(\mu + 1)$ ES was proposed early by (Rechenberg, 1994), but nowadays is no longer used because it is missing a scheme for realizing self-adaptation (Rudolph, 1997). *Median selection*, which is presented in this paper, eliminates this disadvantage.

This paper is organized as follows: in Sect. 2 existing selection methods are presented and in Sect. 3 the new median selection is described; then in Sect. 4 the optimization results on a number of benchmark functions and are presented, which are discussed in Sect. 5. Finally, there are the conclusions in Sect. 6.

2 Selection in Evolution Strategies

2.1 Comma and Plus Selection

In evolution strategies generally the *comma* or *plus* selection is used, denoted as (μ, λ) and $(\mu + \lambda)$. In *plus* selection, the μ parent individuals plus the resulting λ offspring individuals form the selection pool for the parents of the next generation. This causes an elitist selection where the best individual is always contained in the next generation.

In the (μ, λ) selection, only the λ offspring individuals form the selection pool. Thus it is possible that the best offspring individual is worse than the best parent individual and hence a regression happens. Nevertheless, this selection is better suited for adaptation of the step-lengths of the individuals (Schwefel, 1992), because in every generation the possibility of changing the strategy parameters exists.

2.2 Selection in Steady-State Algorithms

In contrast to generational evolutionary algorithms, where a whole offspring population is created in every generation, in steady-state ES only one or a few individuals are created per step and immediately integrated back into the parent population. The term "steady-state" expresses that in one step only a small change takes place and not the whole population changes. The basic algorithm step of steady-state ES is the following:

1. create a new individual and evaluate it with the fitness function,
2. (a) select an old individual which may be replaced by the new one,
   (b) decide, if the old individual will be replaced.

In step 2a one can chose the *replacement strategy*, e. g. replacement of the worst, the oldest or a randomly chosen individual. In step 2b one can chose the *replacement condition*, e. g. replacement if the new individual is better, or unconditional replacement. A widely used combination is to replace the worst individual only if the new individual is better (Bäck et al., 1997, Glossary, Smith, 1998, p. 8). This is an elitist selection and corresponds to the $(\mu + 1)$ strategy. In our simulations, this is denoted as "standard steady-state" selection.

2.3 Steady-State Algorithm with Local Tournament-Selection

Another steady-state algorithm compared here, was inspired by (Smith and Fogarty, 1996). The idea is to generate only a small number λ of offspring individuals and select one of them (e.g. the best) to integrate it into the main parent population. This is a kind of local tournament selection: $(1, \lambda)$. It has a high selection pressure and is distinctive like the normal comma selection. Parallelization of this algorithm is easy: instead of immediately integrating an evaluated offspring individual, λ offspring individuals are collected in a buffer and only the best of them is then integrated in the parent population. In the experiments this algorithm is denoted as "steady-state with local tournament selection".

3 Median Selection

The motivation for the design of the median selection was to get a selection scheme with the following properties:

- it should evaluate and integrate only one individual per step,
- it should be a non-elitist selection, which facilitates self-adaptation; a temporary worsening of the average fitness should be allowed, like in the (μ, λ) selection.

The idea behind the median selection is, that the decision, whether an individual is integrated into the population or not, is made by a decision function without the context of other individuals. This makes it easy to realize a steady-state selection. The function uses data about the fitness distribution of formerly generated individuals which have already passed the selection process. Using this data, the behavior of a (μ, λ) selection is modeled by determining the fitness limit, which separates the μ best individuals from the $\lambda - \mu$ remaining individuals. No further replacement condition is needed.

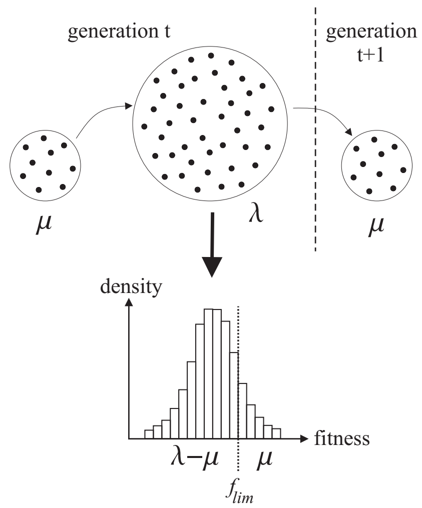


Fig. 1. (μ, λ) selection

The model of Fig. 1 was assumed for the (μ, λ) selection. Out of μ parent individuals λ offspring individuals are generated. Thereof the μ best are selected as parents for the next generation. To do this, it is useful to sort the offspring individuals according to their fitness. If f_{lim} is the acceptance limit fitness value, a new offspring individual i is integrated if $f_i >= f_{lim}$. f_{lim} is is the μ-smallest value (for minimization) or the μ-median of the fitness values of the offspring. Hence the name.

A model is used for the distribution of the fitness values and f_{lim} is determined from it. With every newly created and evaluated individual the model is updated. Because the average fitness of the population should be improving permanently, it is desired that relatively old fitness values are removed from the model. Instead of modeling a particular distribution, the distribution is represented by a sample of the last n_p fitness values of the created offspring individuals. For this a FIFO fitness buffer whose elements additionally are linked in sorting order, is used (Fig. 2). This is not a priority queue, as elements leave the FIFO after n_p steps, regardless of their fitness. This buffer can be accessed in two ways:

1. in FIFO organization, for insertion of a new fitness value. It remains n_p steps in the buffer and then drops out.

2. in sorted order according to the fitness value, for determining the μ-median.

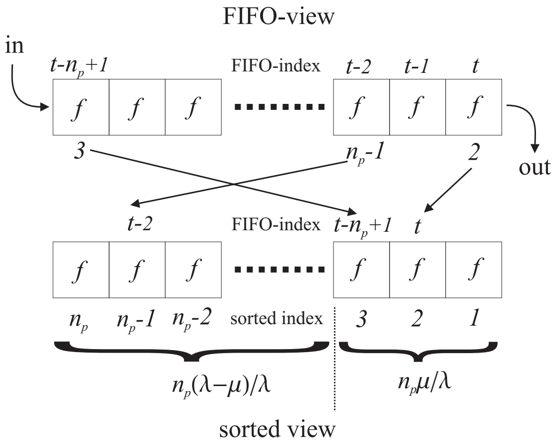


Fig. 2. The buffer with the fitness values of the last n_p created offspring individuals. Above the FIFO view is shown, below the access according to sorted fitness values.

The operation of inserting an element into the FIFO buffer is denoted in the algorithm below by `fifo_insert()`. Access to the k-smallest element is realized by the function `fifo_getSorted(`k`)`. For determining the μ-median, which represents the acceptance limit f_{lim}, the fitness value at index $n_p \cdot \frac{\mu}{\lambda}$ has to be accessed in the sorted buffer.

Additional parameters for the median selection are the length of the FIFO-buffer n_p and the relative rate of acceptance $r_p \hat{=} \frac{\mu}{\lambda}$ which determines the acceptance limit in this way: $f_{lim} = \text{fifo\_getSorted}\,(r_p \cdot n_p)$. The buffer length n_p corresponds to the number of offspring individuals of a corresponding (μ, λ) ES. But now we have the advantage to chose n_p lower than we would chose λ, because this does not primarily affect the selection pressure like λ and it speeds up the adaptation of the fitness acceptance limit. (Smith and Fogarty, 1996) use a ratio of $\frac{\mu}{\lambda} = \frac{1}{5} = 0.2$. (Bäck, 1992a) uses a ratio of $\frac{\mu}{\lambda} \approx \frac{1}{6} \approx 0.16667$. In evolution strategies the ratio $\frac{\mu}{\lambda} = \frac{15}{100} = 0.15$ is often used (Östermeier et al., 1993).

The algorithm for the steady-state ES with median selection is:

```
t = 0;
initialize(P_μ(0)); evaluate(P_μ(0));
fifo_init();
while (not termination) do
    I = recombine(P_μ(t));
    I' = mutate(I);
    evaluate(I');
    f_lim = fifo_getSorted(r_p · n_p);
    if (f(I') better than f_lim) then
        select Individual to replace (I_repl)
        replace I_repl by I';
    endif
    fifo_insert(f(I'));
    t = t + 1;
endwhile
```

The selection of the individual to replace I_{repl} can be performed by one of the replacement methods mentioned in Sect. 2.2: replacement of the worst, oldest or a randomly chosen individual.

4 Evaluation

The following functions numbered according to (Bäck, 1992b) were used as test functions (formulas are not presented here due to space limitations):

- f_2 Generalized Rosenbrock's Function (unimodal, correlated variables),
- f_6 Schwefel's Function 1.2 (unimodal),
- f_9 Ackley's Function (multimodal)
- f_{15} Weighted Sphere (unimodal, different weights for each variable),
- f_{24} Kowalik (multimodal).

All simulations were done with EvA (*Ev*olutionary *A*lgorithms) (Wakunda and Zell, 1997), our own system which contains a large number of variants of genetic algorithms and evolution strategies.

In the simulations a population size of $\mu = 20$ was used consistently to ensure comparability. This is especially necessary for the multimodal functions f_9 and f_{24} in order not to converge into a local optimum.

The simulations on one processor were performed on PCs, the multiprocessor simulations were performed on a Hewlett Packard V2200, a 16 processor shared memory machine, using the MPI library. The parallel version of the program runs on 2 or more processors and consists of one master processor for the core ES algorithm and one or more worker processors for (asynchronous) fitness function evaluations. So, the lowest useful number of processors in the parallel version is 3.

For all simulations covariance matrix adaptation (CMA) was used for adaptation of the strategy parameters, because it is currently the most powerful adaptation method (Hansen and Ostermeier, 1996). The compared strategies are:

1. $(20, \lambda)$ Evolution Strategy (comma) (only sequential simulations),
2. $(20 + 1)$ steady-state ES with *replace worst, if better*; the "standard steady-state"-algorithm,
3. $(20 + (1, \lambda))$ steady-state ES with local tournament selection, replacement strategy *replace oldest* and replacement condition *always* (selection takes already place in local tournament),
4. $(20 + 1)$ steady-state ES with median selection, also with replacement strategy *replace oldest* and replacement condition *always*.

In simulations prior to the tests listed here, it was shown that the replacement strategy *replace oldest* is advantageous in evolution strategies: it causes a non-elitist selection (in contrast to *replace worst*), which is also the case in (μ, λ) selection.

For the different parameters to set for these strategies, no static standard values were used, but for every function the optimal values were determined separately by an extra experiment. These are the following parameters:

- $(20, \lambda)$ ES: optimal λ,
- $(20 + (1, \lambda))$ ES with local tournament selection: optimal tournament size λ,
- $(20 + 1)$ ES with median selection: optimal buffer size n_p, the acceptance limit $r_p = 0.15$ turned out to be good for all simulations.

These values were determined only for the sequential version and were then used also for the parallel version. The actual values are given with each function. For the (μ, λ) ES only sequential results are shown, because it is not very well suited for this kind of parallelization and the other algorithms were more promising.

In all experiments with the sequential algorithm 30 runs were evaluated for each strategy with different values for the random number generator. With the parallel algorithm, 20 runs were evaluated for each strategy and every number of processors. Function f_{24} demanded lower computation resources and had lower convergence rates, so 100 runs were made with all numbers of processors.

4.1 Generalized Rosenbrock's Function f_2

Function f_2 was calculated with dimension $n = 20$, termination criterion was reaching a fitness value less than $\Delta = 10^{-20}$ with a maximum of $t_{max} = 270.000$ function evaluations. For the comma-ES $\lambda = 80$ was chosen, for the steady-state ES with local tournament selection $\lambda = 5$ was chosen and the buffer size of the median-ES was $n_p = 40$ (the acceptance limit is $r_p = 0.15$ for all tested functions).

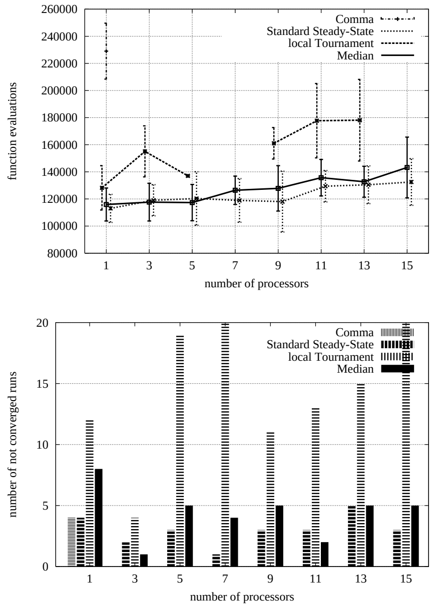
The results are shown in Fig. 3(a). Standard Steady-State and Median selection need nearly the same number of function evaluations, at 9 processors Median needs about 8% more, this is the maximum difference. The difference to local tournament selection is significantly greater: about 10% to 35% in relation to standard steady-state. For local tournament with 7 and 15 processors none of the 20 runs did converge to 10^{-20}. In general the convergence ratio was significantly worse than for the other algorithms.

The comma strategy needs nearly twice as much function evaluations as the steady-state algorithms on one processor. This is similar for the other functions and will be discussed in more detail in Sect. 5.

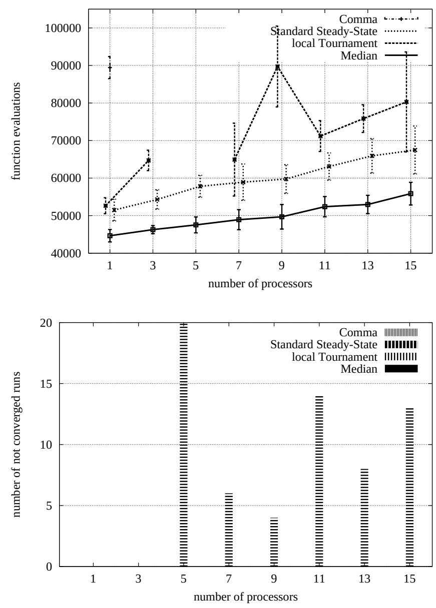
4.2 Schwefel's Function 1.2 f_6

Function f_6 was calculated with dimension $n = 20$, $\Delta = 10^{-20}$, $t_{max} = 100.000$. The free parameters optimized were $\lambda = 70$ (comma-ES); $\lambda = 5$ (local tournament); $n_p = 70$ (median).

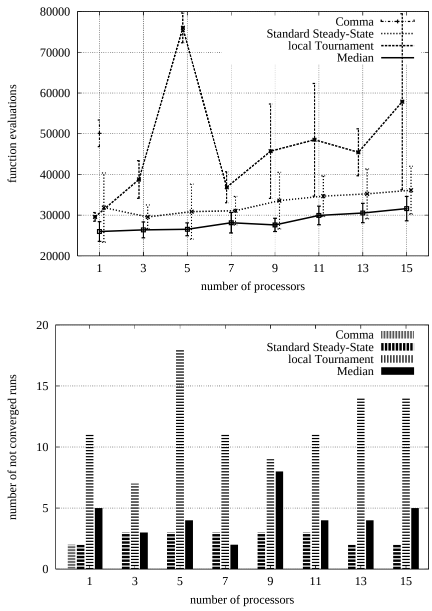
The results are shown in Fig. 3(b). Here Median selection needs only between 80% and 87% of the function evaluations of the standard steady-state algorithm. This means that Median selection is able to adapt the step sizes better. Local tournament selection is here for one processor as good as standard steady-state, but for other number of processors, it needs clearly more function evaluations. The convergence rates for local tournament are quite surprising. With one or three processors, all 20 runs converged. This is the same for the other methods,



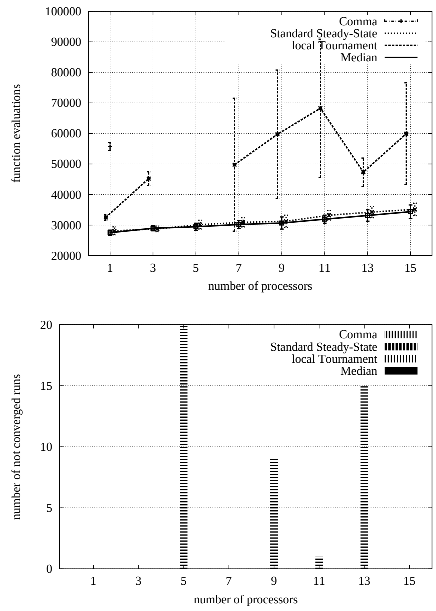
(a) function f_2



(b) function f_6



(c) function f_9



(d) function f_{15}

Fig. 3. Line graphs: comparison of numbers of function evaluations until the termination criterion is reached. Bar graphs: comparison of the number of not converged runs. Details see text.

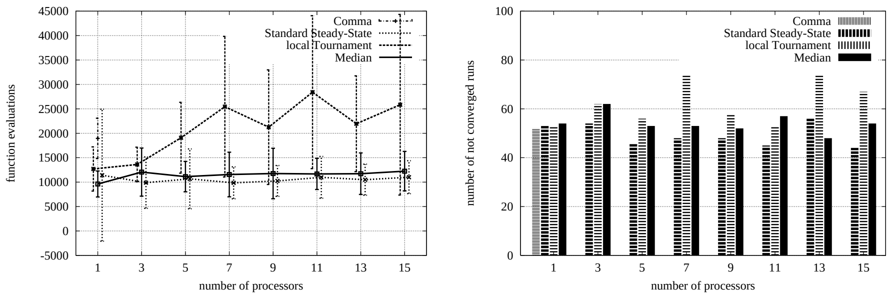


Fig. 4. Comparison for function f_{24}.

where this was the case for all number of processors. But with 5 processors and local tournament selection, none of the 20 runs converged within the 100.000 evaluations.

4.3 Ackley's Function f_9

Function f_9 was calculated with dimension $n = 20$, $\Delta = 10^{-10}$ (due to limited computing precision), $t_{max} = 150.000$. The free parameters were optimized to: $\lambda = 60$ (comma-ES); $\lambda = 5$ (local tournament); $n_p = 40$ (median).

The results are shown in Fig. 3(c). The results are similar to function f_6: Median selection constantly needs less evaluations than standard steady-state. Local tournament is here a little better for one processor, but this is not the case for more than one processors, where it needs clearly more evaluations than the other two parallel methods. The convergence rate for Median were always a little worse than for standard steady-state. Local tournament selection failed to adapt in many runs.

4.4 Weighted Sphere Model f_{15}

Function f_{15} was calculated with dimension $n = 20$, $\Delta = 10^{-20}$, $t_{max} = 160.000$. The free parameters were optimized to: $\lambda = 65$ (comma-ES); $\lambda = 4$ (local tournament); $n_p = 40$ (median).

The results are shown in Fig. 3(d). Standard steady-state and the Median method show almost equal behavior, the convergence rates are 100% for both methods. The local tournament method needs up to two times more function evaluations than the other two, convergence rates are between 0% and 100%.

4.5 Kowalik f_{24}

The dimension of function f_{24} is fixed at $n = 4$. The optimum of f_{24} is given in literature (Bäck, 1992b) with $\min(f_{24}) \approx f_{24}(0.1928, 0.1908, 0.1231, 0.1358) \approx 3.07485988 \cdot 10^{-4}$. Termination criterion was reaching a fitness value less than

$3.07486 \cdot 10^{-4}$ with a maximum of 200.000 function evaluations. The following free parameters were chosen: $\lambda = 100$ (comma-ES); $\lambda = 6$ (local tournament); $n_p = 40$ (median).

Because all compared strategies reached the global optimum at maximum only in half of the runs and this function needed clearly less evaluations than the others, we used 100 runs per strategy to obtain more significant results. The results are shown in Fig. 4(a). The best results are achieved by the standard steady-state algorithm, which needs about 10.000 function evaluations for all numbers of processors. For some numbers of processors, Median selection achieves roughly the same results, but for others, it needs up to 1.5 times more evaluations. Local tournament constantly needs more evaluations than the other two steady-state methods, the convergence rate is worse for some number of processors.

5 Discussion of the Results

The comparisons were all performed with the same number of parent individuals $\mu = 20$. Thereby the $(20, \lambda)$-ES needs more function evaluations than the steady-state algorithms. The reason for this probably lies in the interdependence of the population size and the selection pressure of the comma strategy, which is given by μ/λ. For a fixed μ and fixed selection pressure, a large λ has to be chosen. But above a certain value, an increase of λ has no significant effect towards progress. In this case, it is more efficient to take several smaller steps with a reduced size offspring population.

The strategy with local tournament selection seems to be not so suitable for evolution strategies. It fails to adapt step sizes correctly in many cases and needs more function evaluations than the standard steady-state methods or even is not able to make significant progress at all and the optimization stagnates.

The new method Median selection shows a better performance than the other tested methods for the functions f_6 and f_9. For the other three test functions, Median selection shows similar performance as the standard steady-state method or is only slightly worse for some numbers of processors. The high computing resources needed for the parallel measurements together prevented a still higher number of runs per data point.

Median selection introduces the two new parameters n_p and r_p, but makes the parameter λ obsolete. It seems to be very robust for a fixed setting of these parameters for all numbers of processors.

Regarding the number of function evaluations needed with increasing number of processors, there is only a relatively small increase. This is due to the overlapping of the asynchronous handled fitness evaluations. This promises a near linear speedup and is very good to reduce the computation time for real applications, which need a high amount of computing power.

6 Conclusions

The new selection method *median selection* for steady-state evolution strategies was presented and compared for a number of test functions with other steady-state selection methods and the generational (μ, λ) ES. It indicated that median selection enables self-adaptation as well as or even better than all other selection methods. The algorithm is very well suited for asynchronous, parallel fitness evaluation, which is the preferred parallelization method for optimization problems with the need for high computing resources. Furthermore it turned out that the use of a steady-state evolution strategy is valuable even on a single processor computer without parallel evaluation of the individuals. This is true especially for multimodal functions.

References

Bäck, T. (1992a). The interaction of mutation rate, selection and self-adaptation within a genetic algorithm. In Männer, R. and Manderick, B., editors, *Parallel Problem Solving from Nature – PPSN II*, volume 2, pages 85–94, Amsterdam, Netherlands. Elsevier Science Publishers.

Bäck, T. (1992b). A user's guide to genesys 1.0. Technical report, University of Dortmund, Department of Computer Science, System Analysis Research Group.

Bäck, T., Fogel, D. B., and Michalewicz, Z., editors (1997). *Handbook of Evolutionary Computation*. IOP Publishing and Oxford University Press, New York, Bristol.

Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC '96)*, pages 312–317, Nagoya, Japan. IEEE.

Ostermeier, A., Gawelczyk, A., and Hansen, N. (1993). A derandomized approach to self adaptation of evolution strategies. Technical report, Technische Universität Berlin.

Rechenberg, I. (1973). *Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, TU Berlin, F. f. Maschinenwesen. Published also in: Schriften zur Informatik 1971.

Rechenberg, I. (1994). *Evolutionsstrategie '94*, volume 1 of *Werkstatt Bionik und Evolutionstechnik*. frommann–holzboog, Stuttgart.

Rudolph, G. (1997). Evolution strategies. In Bäck et al., 1997, pages B1.3:1–B1.3:6.

Schwefel, H.-P. (1977). *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, volume 26 of *Interdisciplinary systems research*. Birkhäuser, Basel.

Schwefel, H.-P. (1992). Natural evolution and collective optimum seeking. In Sydow, A., editor, *Computational Systems Analysis — Topics and Trends*, pages 5–14. Elsevier, Amsterdam.

Smith, J. E. (1998). *Self Adaptation in Evolutionary Algorithms*. PhD thesis, Faculty of Computer Studies and Mathematics, University of the West of England, Bristol.

Smith, J. E. and Fogarty, T. C. (1996). Self adaptation of mutation rates in a steady state genetic algorithm. In *Proceedings of the 1996 IEEE Conference on Evolutionary Computation*, pages 318–323, New York. IEEE Press.

Wakunda, J. and Zell, A. (1997). EvA - a tool for optimization with evolutionary algorithms. In *Proceedings of the 23rd EUROMICRO Conference*, Budapest, Hungary.