A new Selection Scheme for Steady-State Evolution Strategies

Jürgen Wakunda and **Andreas Zell**
University of Tübingen, Computer Science Dept.
Köstlinstr. 6, D-72074 Tübingen, Germany
{wakunda,zell}@informatik.uni-tuebingen.de

Abstract

In this paper, a new selection scheme for a $(\mu + 1)$ steady-state evolution strategy is described: the so-called median selection. In contrast to generational algorithms, only one individual is generated and evaluated in one step of the algorithm and is immediately integrated into the population. Previous steady-state algorithms are similar to the $(\mu+\lambda)$ selection scheme in evolution strategies, which has a disadvantage in the fast self-adaptation of mutation step-length. This is compensated by the presented *median selection*, which is oriented at the (μ, λ) selection. The median selection is compared with other steady-state selection schemes and with (μ, λ) selection. As a result, median selection achieves better or equally good results as the other selection schemes for a large number of benchmark functions. Additionally, it is shown that the use of a sequential steady-state evolution strategy is advantageous even on one-processor computers.

1 INTRODUCTION

Evolution Strategies (ES) were developed at the end of the sixties by Ingo Rechenberg and Hans-Paul Schwefel at the Technical University of Berlin [Rechenberg, 1973, Schwefel, 1977]. Since then, they have been improved enormously in theory and practice and have been established in the line of Evolutionary Algorithms (EA), beside Genetic Algorithms (GA), Genetic Programming (GP), Evolutionary Programming (EP) and similar methods.

The main applications of evolution strategies are multi-dimensional, real-valued parameter optimization problems. The most important property of ES is the ability of self-adaptation. With self-adaptation the size and distribution of mutations on the object variables are adapted at runtime and thus the optimization speed is maximized. 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 the powerful 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, (μ, λ) selection is superior to $(\mu + \lambda)$ selection (see next section). The $(\mu + 1)$ Evolution Strategy was proposed early by Rechenberg [Rechenberg, 1994], but nowadays is no longer used because there is missing a scheme for realizing self-adaptation [Rudolph, 1997]. The "median selection", which is presented in this paper, eliminates this disadvantage.

This paper is organized as follows: in section 2 existing selection-methods are presented and in section 3 the new median selection is described; then in section 4 the test functions are given; in section 5 the simulation results are presented, which are discussed in section 6. Finally, there are the conclusions in section 7.

2 PLUS-, COMMA- AND STEADY-STATE SELECTION

In the following sections some selection methods for standard evolution strategies and for steady-state algorithms are presented.

2.1 COMMA- AND PLUS-SELECTION FOR EVOLUTION STRATEGIES

In evolution strategies generally the comma- or plus selection is used, denoted as (μ, λ) and $(\mu + \lambda)$. Here μ is the size of the parent population and λ is the size of the offspring population. This notation is motivated by the fact that in plus selection, the μ parent individuals *plus* the resulting λ offspring individuals form the selection pool for selection of the parents of the next generation. This causes the best individual to be always contained in the next generation. Therefore it is an "elitist" selection.

However, in the (μ, λ) selection, only the λ offspring individuals form the selection pool. To be able to select μ new parents at all, $\lambda \geq \mu$ has to hold. In comma selection, it is possible that the best offspring individual is worse than the best parent individual and hence a regression happens. Anyhow, this selection is better suited in the long run for adaptation of the step-lengths of the individuals [Schwefel, 1992], because in every generation the possibility of changing the strategy parameters exists. Additionally it allows to escape from local minima.

2.2 SELECTION IN STEADY-STATE ALGORITHMS

In contrast to so-called generational evolutionary algorithms, where a whole offspring population is created in every generation, in steady-state EAs 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 (only one step is shown, the surrounding loop-code is left out):

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, a steady-state evolution strategy with these replacement parameters is used for comparison and is denoted as "standard steady-state".

The algorithm for the "standard steady-state ES" is as follows (I - individual; P - population; Index μ - parent population; $P_\mu(t)$ - population of generation t):

```
t = 0;
initialize($P_\mu(0)$); evaluate($P_\mu(0)$);
while (not termination) do
    $I$ = recombine($P_\mu(t)$);
    $I'$ = mutate($I$);
    evaluate($I'$);
    if ($I'$ is better than worst($P_\mu(t)$)) then
        replace worst($P_\mu(t)$) by $I'$;
    endif
    t = t + 1;
endwhile
```

Because in a steady-state algorithm the parent individuals participate in the selection process just as in the case of a plus ES, the same notation for them is used in this paper. The kind of algorithm is given additionally: $(\mu + 1)$ steady-state

2.3 STEADY-STATE ALGORITHM WITH LOCAL TOURNAMENT-SELECTION

Another steady-state algorithm compared here, was inspired by Smith and Fogarty [Smith and Fogarty, 1996]. They modified Genetic Algorithms and added a method for self-adaptation of the bit mutation rate, which is coded into every individual. In their work the bit string of an individual consists of two parts: 1. the encoded mutation rate (gray, binary or exponential) 2. the encoded problem representation. Creation of an offspring individual works as follows:

1. create a new individual by crossover,
2. copy this individual λ times and perform steps 3 and 4 on every copy:
3. mutate the coded mutation rate,
4. mutate the problem encoding with the new mutation rate,
5. evaluate all λ new individuals,
6. select one of the λ individuals and integrate it into the population.

The idea is to generate only a small number λ of offspring individuals with different mutation rates and select one of them (e.g. the best) to integrate it into the main parent population. Integration is performed applying one of the replacement strategies and replacement conditions mentioned earlier. This is a kind of local tournament selection: $(1, \lambda)$. It has a high selection pressure and is distinctive like the normal (μ, λ) selection.

For this paper, the algorithm of Smith and Fogarty was transferred to an Evolution Strategy. The best individual was selected from the offspring individuals – which is standard in evolution strategies – analogous to the $(1, \lambda)$ selection. In contrast to the algorithm of Smith and Fogarty, the λ offspring individuals are not created from the same parent individual with different mutation rates, but the parents are chosen anew for each offspring individual. So the only difference between the (μ, λ) ES and the new algorithm lies in the selection method and not in the way the offspring individuals are created.

The algorithm used in this paper is denoted as "steady-state with local tournament selection" and the pseudocode is given here:

```
$t$ = 0;
initialize($P_\mu(0)$); evaluate($P_\mu(0)$);
while (not termination) do
    $P_\lambda$ = recombine($P_\mu(t)$);
    $P'_\lambda$ = mutate($P_\lambda$);
    evaluate($P'_\lambda$);
    $I$ = selectBest($P'_\lambda$);
    select Individual to replace ($I_{repl}$)
    if (replacement condition) then
        replace $I_{repl}$ by $I$;
    endif
    t = t + 1;
endwhile
```

For this special variant of a steady-state ES, the following notation was chosen: $(\mu + (1, \lambda))$

This shall emphasize the $(1, \lambda)$ local selection, but it does *not* mean, that the λ offspring individuals are generated from one parent individual.

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, wether an individual is integrated into the population or not, is made by a decision function without the context of other individuals. The use of such a function, which is able to decide for one single individual, if it is accepted or not, makes it easy to realize a steady-state selection. The function does not use the fitness values of λ newly generated individuals, instead it 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.

The model of figure 1 was assumed for the (μ, λ) selection.

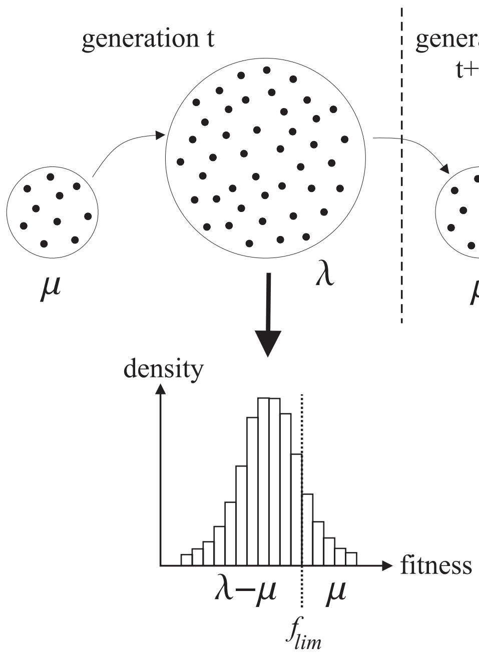

Figure 1: (μ, λ) selection

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.

One could look at this from another point of view: With the fitness value f_{lim} of the worst but still selected individual, the decision for each of the λ offspring individuals (fitness $f(ind)$), wether it is selected as parent of the next generation or not, can be made by simply comparing the fitness values:

```
if (f(ind) >= f_lim) then integrate(ind);
```

The "acceptance limit" f_{lim} is determined by the distribution of the fitness values of all offspring individuals and is the μ-smallest value (for minimization) or the μ-median of the set of fitness values. 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 increasing permanently, it is desired that relatively old fitness values are removed from the model. No particular distribution (e. g. Gaussian normal distribution or similar) was assumed. Instead, the distribution is recorded with a sample of the last n_p fitness values of the created offspring individuals. For this recording a fitness buffer is used, which is organized according to the FIFO-principle (First In First Out) (figure 2).

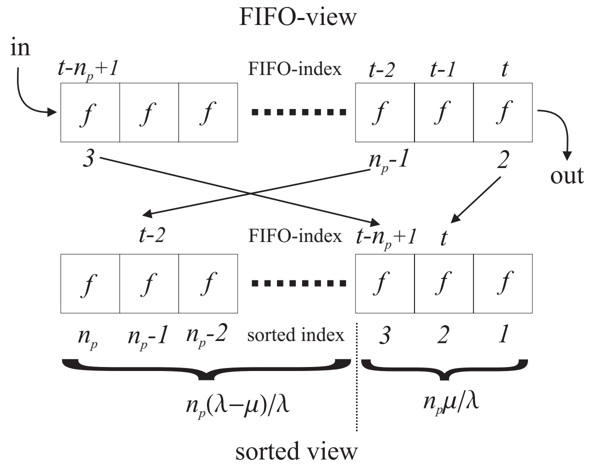

Figure 2: The buffer with the fitness values of the last n_p created offspring individuals. On the top the FIFO-view is shown, below the access according to sorted fitness values.

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.

The buffer is realized as an array because the number of values remains constant. The insertion index is moved cyclically when a new fitness value is inserted. To efficiently access the k-smallest element a doubly linked list of the elements in sorted order is maintained. On insertion of an element, these links are updated. The insertion operation is denoted in the following text 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.

Because the algorithm with median selection is also a steady-state algorithm, the same notation is used. The selection method has to be given additionally:

$$(\mu + 1) \text{ median}$$

Additional parameters for the median selection are the length of the FIFO-buffer n_p (it can be different from the number of offspring individuals of a corresponding (μ, λ) ES) and the relative rate of acceptance $r_p \hat{=} \frac{\mu}{\lambda}$ which determines the acceptance limit in this way: $f_{lim} = \text{fifo\_getSorted}(r_p \cdot n_p)$

Smith and Fogarty [Smith and Fogarty, 1996] use a ratio of $\frac{\mu}{\lambda} = \frac{1}{5} = 0.2$. Bäck [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 [Ostermeier et al., 1993].

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

```
t = 0;
initialize($P_\mu(0)$); evaluate($P_\mu(0)$);
fifo_init();
while (not termination) do
    $I$ = recombine($P_\mu(t)$);
    $I'$ = mutate($I$);
    evaluate($I'$);
    $f_{lim}$ = fifo_getSorted($r_p \cdot n_p$);
    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 section 2.2: replacement of the worst, oldest or a randomly chosen individual.

4 TEST FUNCTIONS

The following functions numbered according to [Bäck, 1992b] were used as test functions:

- f_2 Generalized Rosenbrock's Function (unimodal with interdependencies between variables)
- f_6 Schwefel's Function 1.2 (unimodal)
- f_9 Ackley's Function (multimodal)
- f_{15} Weighted Sphere Model (unimodal, extension of the "Sphere Model" function f_1 with different weights for each variable)
- f_{24} Kowalik (multimodal)

5 SIMULATIONS

All simulations were done with the EvA-system for *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. For the unimodal functions f_2, f_6 and f_{15}, $\mu = 1$ would be sufficient, but the problems for which steady-state algorithms are used, are normally multimodal "real-world" problems.

The simulations were performed on a uniprocessor computer. They serve at first for comparing the different methods. The test functions are relatively fast to compute in comparison with the communication time over a network. So a distributed computation would not be efficient. But an implementation with parallel, asynchronous evaluation of multiple individuals to be used for bigger optimization problems exists already as logical continuation of this work.

For all simulations the Covariance-Matrix Adaptation (CMA) was used for adaptation of the strategy parameters, because this method is the most powerful of the existing adaptation methods [Hansen and Ostermeier, 1996].

The compared strategies are:

1. $(20, \lambda)$ Evolution Strategy (comma),
2. $(20 + 1)$ ES (plus),
3. $(20 + 1)$ steady-state ES with *replace worst, if better*; the "standard steady-state"–algorithm,
4. $(20 + (1, \lambda))$ steady-state ES with local tournament selection, replacement strategy *replace oldest* and replacement condition *always* (selection takes already place in local tournament),
5. $(20 + 1)$ 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. Local Tournament selection and median selection bring both their own replacement condition: the local tournament and the comparison with the acceptance limit f_{lim}.

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.

The actual chosen values are given in the simulations section.

In the following sections 30 runs have been evaluated for each strategy with different values for the random number generator, except for f_{24}, where 100 runs were used (see 5.5). For each test function, a diagram is presented, which shows for each strategy the average number of function evaluations needed and the range of the standard deviation.

5.1 F_2 GENERALIZED ROSENBROCK'S FUNCTION

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).

In figure 3 for each strategy the average value and the range of the standard deviation of the number of function evaluations is given.

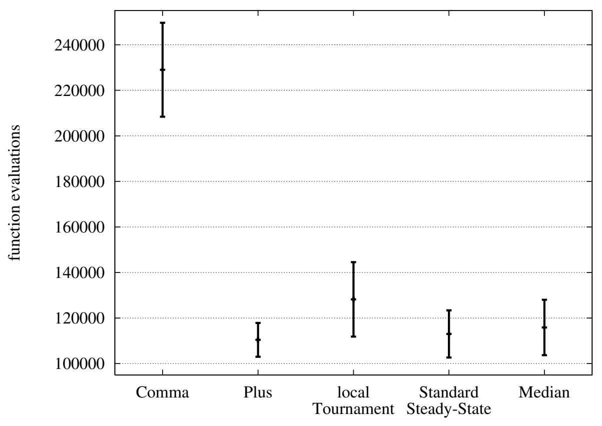

Figure 3: Average and standard deviation of the number of function evaluations until reaching a fitness value less than 10^{-20} for function f_2.

For function f_2 median selection needs on average approx. 5% more function evaluations than the plus strategy. The difference to the standard steady-state algorithm is even smaller. For the strategy with local tournament selection, however, approx. 16% more function evaluations than for the plus strategy are needed. The comma strategy needs even twice as much as plus. This point will be discussed in more detail in section 6.

5.2 F_6 SCHWEFEL'S FUNCTION 1.2

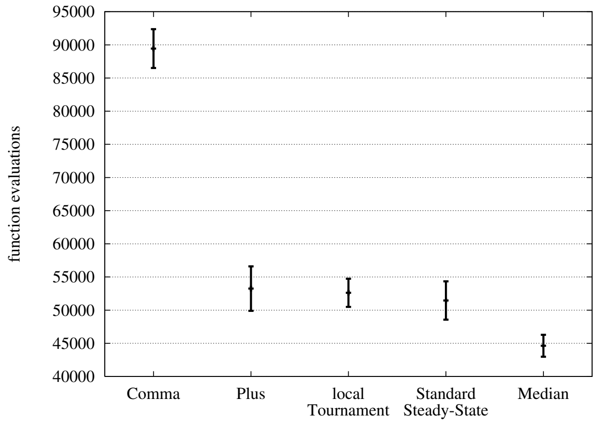

Figure 4: Average and standard deviation of the number of function evaluations until reaching a fitness value less than 10^{-20} for function f_6.

Function f_6 was calculated with dimension $n = 20$, termination criterion was reaching a fitness value less than $\Delta = 10^{-20}$, $t_{max} = 100.000$.

The following free parameters were chosen: $\lambda = 70$ (comma-ES); $\lambda = 5$ (local tournament); $n_p = 70$ (median).

For this function the three other steady-state algorithms need between 15% and 19% more function evaluations than steady-state with median selection.

5.3 F_9 ACKLEY'S FUNCTION

Function f_9 was calculated with dimension $n = 20$, termination criterion was reaching a fitness value less than $\Delta = 10^{-10}$ (due to limited computing precision), $t_{max} = 150.000$.

The following free parameters were chosen: $\lambda = 60$ (comma-ES); $\lambda = 5$ (local tournament); $n_p = 40$ (median).

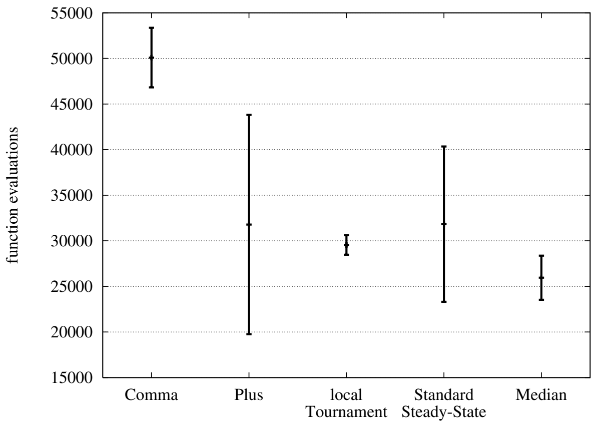

Figure 5: Average and standard deviation of the number of function evaluations until reaching a fitness value less than 10^{-10} for function f_9.

Here the steady-state algorithm with median selection needs on average the lowest number of function evaluations and has a relatively small standard deviation. The tournament method needs approx. 14% and plus and standard steady-state approx. 23% more function evaluations on average. The tournament method shows here an extremely small standard deviation. In contrast, standard steady-state and especially plus have a several times greater one.

5.4 F_{15} WEIGHTED SPHERE MODEL

Function f_{15} was calculated with dimension $n = 20$, termination criterion was reaching a fitness value less than $\Delta = 10^{-20}$, $t_{max} = 160.000$.

The free parameters were optimized to: $\lambda = 65$ (comma); $\lambda = 4$ (loc. tournament); $n_p = 40$ (median).

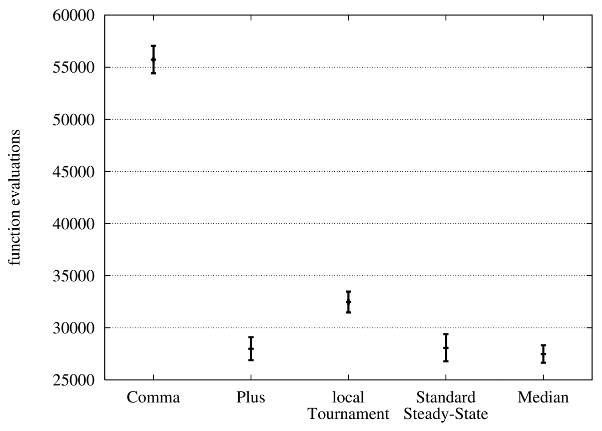

Figure 6: Average and standard deviation of the number of function evaluations until reaching a fitness value less than 10^{-20} for function f_{15}.

For plus, standard steady-state and the median selection, equally good results are obtained for this function, with negligible differences. Local tournament selection needs approx. 18% more function evaluations. The standard deviations are very small for this unimodal function.

5.5 F_{24} KOWALIK

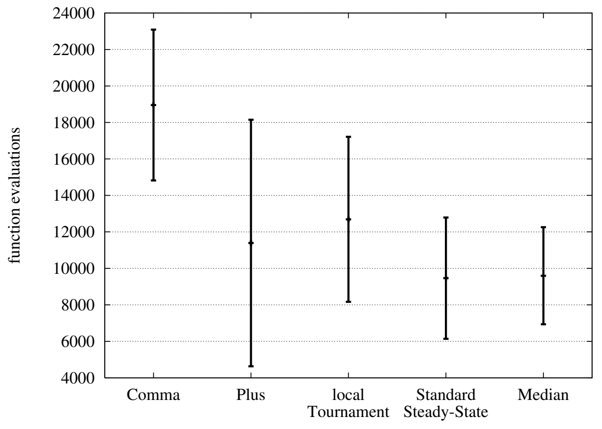

Figure 7: Average and standard deviation of the number of function evaluations until reaching a fitness value less than $3.07486 \cdot 10^{-4}$ for function f_{24}.

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 $\Delta = 3.07486 \cdot 10^{-4}$ with a maximum of $t_{max} = 200.000$ function evaluations.

The following free parameters were chosen: $\lambda = 100$ (comma-ES); $\lambda = 6$ (local tournament); $n_p = 40$ (median).

The optimum of f_{24} is not at 0, here such a good approximation of the optimum like in the other functions was not demanded. All compared strategies reached the global optimum only in half of the runs. Hence we used 100 runs per strategy to obtain more significant results.

Here, standard steady-state and the median selection have approximately the same average values, plus needs approx. 20%, local tournament selection approx. 34% more function evaluations. But the relevance of this result becomes smaller when we consider the large standard deviations.

6 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 $(20 + 1)$-ES and the other 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 the ratio of μ/λ. Consequently, for a fixed μ and fixed selection pressure, an accordingly large λ has to be chosen. But in one generation, no arbitrarily big optimization progress can be achieved. Above a certain value, an increase of λ has no noteworthy effect, regarding the possible progress. In this case, it is more efficient to take several smaller steps with a reduced size offspring population which leads to a bigger progress altogether.

The $(20+1)$-ES always behaves approximately like the standard steady-state evolution strategy with *replace worst*, *if better* replacement strategy and condition. This result was expected, but differences in implementation are possible and probable. In plus strategy, the parent and offspring populations are mixed, sorted and then the $\mu = 20$ best are selected for the next generation. This is necessary because λ can be an arbitrary value bigger or equal to 1. In contrast to this, for the standard steady-state strategy λ is always 1, so with the replacement strategy *replace worst* not the whole population has to be sorted, but only the worst individual has to be found and eventually replaced.

The steady-state evolution strategy with median selection obtains very good results for all five test functions used here. For the functions f_2, f_{15} and f_{24} approximately the same number of function evaluations are needed as with the best other strategies. For the functions f_6 and f_9 the median selection is even better than the other strategies tested here.

The strategy with local tournament selection seems to be not so suitable for evolution strategies, in most of the cases it needs more function evaluations than other strategies. Only at function f_9 it is better than the plus- and standard steady-state strategy and even shows an extremely small standard deviation. But at this function the median selection strategy is best.

Further investigation of the median selection is planned, e. g. measuring the actual rate of acceptance, that means the number of individuals which are accepted in relation to the total number of offspring individuals generated. This rate has not to be identical with the acceptance rate parameter r_p. This parameter is only used for accessing the FIFO-buffer to get the acceptance limit f_{lim}. Also, tests with the parallelized version of the algorithm with asynchronous evaluation and integration of the individuals will be made. The impact on the effectiveness is to be analyzed.

7 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 showed that the non-elitist median selection enables self-adaptation as well as or even better than all other selection methods.

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, where the number of parent individuals μ has to be larger than 1 to ensure a reasonable convergence probability to find the global optimum.

References

[Bäck, 1992a] 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, 1992b] 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 et al., 1997] Bäck, T., Fogel, D. B., and Michalewicz, Z., editors (1997). *Handbook of Evolutionary Computation.* IOP Publishing and Oxford University Press, New York, Bristol.

[Hansen and Ostermeier, 1996] 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 et al., 1993] Ostermeier, A., Gawelczyk, A., and Hansen, N. (1993). A derandomized approach to self adaptation of evolution strategies. Technical report, Technische Universität Berlin.

[Rechenberg, 1973] 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, 1994] Rechenberg, I. (1994). *Evolutionsstrategie '94*, volume 1 of *Werkstatt Bionik und Evolutionstechnik.* frommann–holzboog, Stuttgart.

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

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

[Schwefel, 1992] 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, 1998] 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 and Fogarty, 1996] 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 and Zell, 1997] Wakunda, J. and Zell, A. (1997). EvA - a tool for optimization with evolutionary algorithms. In *Proceedings of the 23rd EUROMICRO Conference*, Budapest, Hungary.