EvA
A Tool for Optimization with Evolutionary Algorithms

Jürgen Wakunda and Andreas Zell
Universität Tübingen
Wilhelm-Schickard-Institut für Informatik, Lehrstuhl Rechnerarchitektur
Köstlinstr. 6, D-72074 Tübingen, Germany
{zell,wakunda}@informatik.uni-tuebingen.de

Abstract

We describe the EvA software package which consists of parallel (and sequential) implementations of genetic algorithms (GAs) and evolution strategies (ESs) and a common graphical user interface. We concentrate on the descriptions of the two distributed implementations of GAs and ESs which are of most interest for the future.

We present comparisons of different kinds of genetic algorithms and evolution strategies that include implementations of distributed algorithms on the Intel Paragon, a large MIMD computer, and massively parallel algorithms on a 16384 processor MasPar MP-1, a large SIMD computer.

The results show that parallelization of evolution strategies not only achieves a speedup in execution time of the algorithm, but also a higher probability of convergence and an increase of quality of the achieved solutions. In the benchmark functions we tested, the distributed ESs have a better performance than the distributed GAs.

1. Introduction

Evolution strategies are stochastic, derivative-free optimization algorithms for mathematical and engineering optimization tasks. For genetic algorithms there exist several models of parallelization, which are also suitable for evolution strategies. Parallelization of evolution strategies promises not only increased computation speed, but also new aspects like the finding of better solutions.

Because a MIMD computer is normally built of widely available standard microprocessors, it can be easily programmed in a standard programming language and is a multi purpose computer suitable for almost all kinds of tasks. In particular, the aspect of evolutionary algorithms that there can be several relatively isolated populations (island model) matches the granularity of parallelization on a MIMD computer.

2. Overview of EvA

The EvA project (Evolutionary Algorithms), a system for optimization by means of evolutionary algorithms, was originally started at the University of Stuttgart, but has been continued at the University of Tübingen during the last two years. EvA consists of different parallel implementations of evolutionary algorithms – genetic algorithms as well as evolution strategies – on different parallel computer architectures – SIMD and MIMD.

There exist three implementations of parallel genetic algorithms and two implementations of Evolution Strategies:

- VEGA - Distributed Genetic Algorithms [2, 14] ('VE' stands for distributed - 'verteilt' in german) on the MIMD computer Intel Paragon and workstation clusters,
- MPGA - Massively Parallel Genetic Algorithms [9] on the SIMD computer MasPar MP-1 with 16384 processors,
- CNGA - CNAPS Genetic Algorithms [3] on the neurocomputer Adaptive Solutions CNAPS, a SIMD computer with 512 processors,
- VEES - Distributed Evolution Strategies [13] on the MIMD computer Intel Paragon and workstation clusters,
- MPES - Massively Parallel Evolution Strategies [4] on the SIMD computer MasPar MP-1 with 16384 processors.

Proceedings of the 23rd EUROMICRO Conference '97 New Frontiers of Information Technology
1089-6503/97 $10.00 © 1997 IEEE

All these programs can be started and controlled either stand-alone with a text based and menu driven user interface or by an easy-to-use graphical user interface for X11/Motif which is described in the next section.

As a special feature the script language Tcl[1] [10] is integrated in the EvA programs and the Tcl interpreter parses the command-line input of the text based user interface. Thus all the useful features of Tcl, like variables, loops, conditions, etc. are available. Instead of entering the commands interactively at the prompt, they can also be read from a batch file. In this case the Tcl commands are especially useful to write powerful batch scripts. The graphical user interface provides an edit window to create batch files which can be sent to the evolutionary algorithm and which are executed directly.

To get a feedback of how the evolutionary algorithm progresses, statistical data is produced during a run and written into several files. The output formats of all programs of the EvA project are the same and suitable for automated parsing and further processing by external analyzing tools. For the whole session a summary file is created, where an overview of all runs and the names of the additionally generated two protocol files are stored. In one protocol file the resulting best individuals of each processor are stored and in the other statistical data is stored, e. g. the three values of best, average and worst fitness that occur in any population. Also the measured time for each part of the algorithm is listed, e. g. the recombination-time, mutation-time, communication-time, and so on. For visualization of the results it is possible to follow the progression of fitness with a graphical gnuplot window. The three values of best, average and worst fitness are displayed in a customizable interval.

3. The graphical user interface UIEA

The graphical user interface UIEA (User Interface for Evolutionary Algorithms) is based on X11 and uses Motif 1.2. UIEA is a separate program, which starts the desired evolutionary algorithm ('EA module') via remote-shell on a remote computer or directly on the local workstation. The EA-module is then controlled with *remote procedure calls* (RPC). With this concept it is possible to run UIEA on the local workstation and the evolutionary algorithm on a remote parallel computer which therefore need not deal with graphics.

UIEA is not only a graphical frontend but provides additional functionality: for example it provides a script editor where batch scripts can be written and directly be sent to the EA module. Parameter settings can easily be loaded, stored and be inserted into a batch script. A hypertext help system is integrated, which uses HTML[2] help pages. Different configurations of the EA modules can be accessed, e. g. VEES on the Intel Paragon, VEGA on the local workstation, etc.

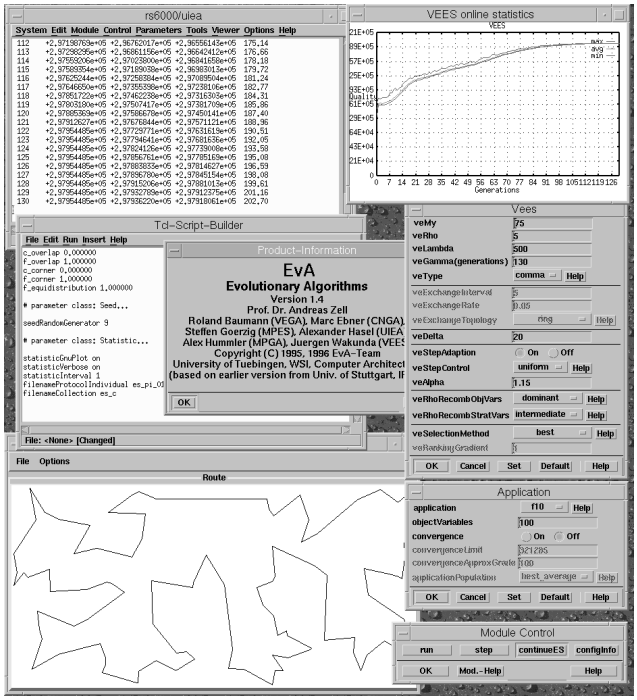

Figure 1. Screen of a session with VEES. The upper windows are the main window of UIEA (left) with the statistical output of the fitness values and the same data graphically displayed with gnuplot (right). In the middle left is the Tcl-Script-Builder and the information window of EvA. Some of the windows for changing the parameter settings of VEES are placed at the right side and at the lower left corner we see a viewer which shows the route for the TSP.

The most recent feature of UIEA is a software interface for data transportation from the EA module to external programs for visualization tasks. Data that emerges during the run of an evolutionary algorithm, e. g. fitness values or whole individuals, is sent to UIEA where it is transferred to one or more "viewers" which can process the data. In figure 1 there can be seen a viewer which shows the current best route of the Travelling Salesman Problem, which is optimized.

4. VEES – distributed evolution strategies

The program VEES (Verteilte Evolutionsstrategien = distributed ES) is built according to the master-slave model (Fig. 2). It uses one master node and a large number of slave

[1] Tool Command Language
[2] hypertext markup language

Proceedings of the 23rd EUROMICRO Conference '97 New Frontiers of Information Technology
1089-6503/97 $10.00 © 1997 IEEE

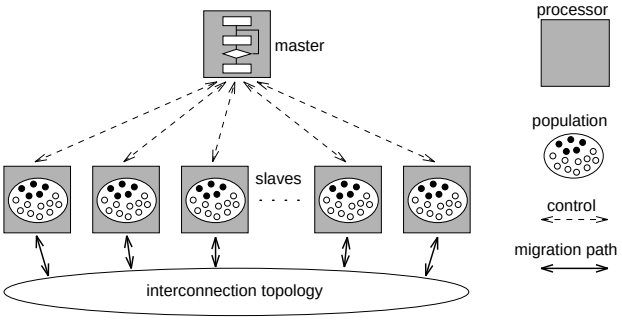

Figure 2. Master-slave architecture of the program. The master runs the control algorithm and on each slave is kept one population on which the ES is performed.

nodes. The master node deals with the complete program administration, it runs the text based user interface and coordinates the slave nodes. If only one node is available, this node performs both tasks, the administration task and the calculation work. Because VEES is able to run on only one node, it is also possible to run VEES on a single-processor workstation. The program is written in ANSI-C and hence is easily portable to other UNIX systems.

The slave nodes run the actual ES code. They wait in a receive loop for messages from the master. Depending on the received message, they perform special actions, like initialization with all necessary ES-data, computation of some generations of an ES in isolation, calculation of statistical data, and so on. On each slave node a separate population is kept. All populations have the same number of individuals. Each slave node performs one or more generations of the ES independently of all other nodes. The number of generations calculated independently is determined by the intervals for individual exchange and statistic calculation and is the same on all slave nodes.

Since all slave nodes are similar, perform similar computations and each node only runs one application program, a generation on each slave node takes approximately the same time. This holds at least for the complete set of standard benchmark functions and for many real world applications. Thus it is easy to synchronize all slave nodes at the end of a calculation cycle without long waiting periods. When all slaves have completed the isolated calculation, they communicate with the master node and get new commands. Slave node numbers start with 0, the master node is assigned the highest logical node number. All communication primitives of the program reside in a separate module. Thus, it is relatively easy to change the communication library, e. g. it is currently possible to switch from the custom NX library to the more portable MPI library.

4.1. Two Parallelization Methods on a MIMD–Computer

There exist several different ways of how to parallelize evolutionary algorithms on different computer architectures, which can be classified by the object of parallelization [6]. For MIMD machines it is best to parallelize on the level of populations, because of the coarse grained parallelism of these systems. The communication on MIMD machines normally has some latency, because it is asynchronous, but has a high throughput rate. Therefore it is best to communicate infrequently but with possibly big message sizes. This is suitable for sending whole populations of individuals after a relatively long calculation step like the computation of a generation.

In VEES two kinds of parallel, distributed evolution strategies are realized. A purely distributed strategy, based on the *island model* and a parallelized version of a nested (μ^+, λ)–strategy which was proposed by Rechenberg [11].

4.2. Distributed Strategy

The distributed evolution strategy is based on the island model. On an island a population can develop in isolation. But sometimes an individual of an other population is able to reach the island and some individuals are able to leave the island. This process is called *migration*. Migrated individuals increase the genetic diversity of a population which is especially important for the local operator *recombination* or *crossover* and thus for the rate of progress of the population. But diversity is also important for the whole evolutionary algorithm considered as dynamical system.

In a distributed evolution strategy (DES) each processor performs a (μ^+, λ)–strategy for some generations in isolation. Schwefel's notation [12] of such a strategy is:

$$(\mu/\rho^+, \lambda)^\gamma$$

where μ is the number parent individuals, ρ the number of individuals used for recombination, λ the number of offspring individuals, γ the number of generations to calculate and $+/,$ the choice of selection method: parent and offspring or only offspring individuals.

After the isolation an exchange of the best individuals between the processors is made. The number of generations which are calculated in isolation is called the *exchange interval*. During the exchange a certain number of individuals, determined by the *exchange rate*, is sent from each processor to some other processors. The destination processors are determined by the *exchange topology* which consists in VEES of bidirectional migration paths between two populations. Not necessarily all populations are connected with each other, so that an individual cannot reach every

other population during one exchange period. The longest distance between two populations is called the *diameter* of the topology. The three parameters diameter (given by the exchange topology), exchange interval and exchange rate should be attuned well to each other. If they lead to a short temporal-spacial distance between the populations, the advantage of the distributed model of not converging too soon into local optima is lost.

4.3. Nested Strategy

Rechenberg [11] introduced a type of evolution strategy on a higher level: the nested ES. The idea is that the same mechanisms duplication, recombination, mutation and selection can be performed on populations instead of individuals. One or more populations produce several offspring populations which then have a certain amount of time to develop in isolation in the same manner like in an (μ^+, λ)–strategy. After that they have to compete against each other to determine which populations will be selected for the next cycle. The notation of this kind of strategy is:

$$[\mu'/\rho'^+, \lambda'(\mu/\rho^+, \lambda)^\gamma]^{\gamma'}$$

Here we have the same parameters μ', ρ', λ' and γ' on population level indicated by a stroke. Mutation of a population means the relocation of the whole population to another place in the problem space. Recombination objects are now complete individuals of the parent populations. One cycle γ' on population level comprises γ generations of the inner strategy.

The advantage of the nested (μ^+, λ)–strategy over a normal (μ^+, λ)–strategy is the parallel search in different regions of the problem space, thus the probability of finding the global optimum is higher. The mutation and selection of populations also prevents the strategy from early convergence into a local optimum.

This model is very suitable for parallelization, because it includes already several independent populations. Since the main computation work is done on the λ' offspring populations, in VEES λ' is set to the number of slave nodes used. Thus each processor keeps one offspring population. To avoid multiple parent populations on any node, the number of parent populations μ' is restricted to a value less or equal than λ'. This would not be necessary in a strategy with selection type '+' on population level, but it simplifies the implementation and keeps the amount of communication low.

The coordination of the ES mechanisms on population level is the task of the master. It determines which populations are duplicated or used for recombination and finally which populations are selected to be parents in the next cycle.

5. VEGA – distributed genetic algorithms

The program VEGA (Verteilte Genetische Algorithmen = distributed GAs) has the same master-slave architecture as VEES. It implements a distributed genetic algorithm based on the island model with exchange of individuals over a defined exchange topology (analogous to section 4.2). The genetic algorithm can be varied in many points. There are several exchange topologies (ring, grid, x-net, hypercube, full connected), crossover methods (single/double point, uniform), individual codings (binary, gray code) and selection methods (roulette-wheel, ranking, stochastic remainder, deterministic sampling) available.

6. Comparison of different Implementations

The programs used for the comparisons are:

- VEES (described in section 4)
- VEGA (described in section 5)
- ESCAPADE
  ESCAPADE is an implementation of a $(\mu/\rho^+, \lambda)^\gamma$ strategy which is able to run on a single-processor workstation [7, 8]. For our comparison ESCAPADE version 1.2 was available.
- MPES
  MPES is also part of EvA and is a massively parallel version of ES on the SIMD computer MasPar MP-1 [4]. In MPES three kinds of ES are implemented: (μ^+, λ)-ES, nested (μ^+, λ)-ES and a special kind of ES adopted to the massively parallel architecture. The point of interest in this comparison is the difference between the SIMD and the MIMD implementation of ES.

In the evolutionary algorithms of EvA 24 standard benchmark functions are implemented, which we took from [1], because these functions are implemented in GENEsYs which was used for comparisons with the GA implementations of the EvA project. For the tests described here we chose four functions out of the 24, which have different properties.

The following names are used to denote the rows of the tables with the measured values:

n	number of compute nodes
n_{ind}	number of individuals (λ)
n_{gen}	number of generations (γ)
t_{tot}	total amount of compute time
t_{gen}	time per generation $t_{\mathrm{gen}} = t_{\mathrm{tot}}/n_{\mathrm{gen}}$
t_{ind}	time per individual $t_{\mathrm{ind}} = t_{\mathrm{gen}}/(n \cdot n_{\mathrm{ind}})$
r_{conv}	convergence rate $r_{\mathrm{conv}} = \frac{\#\text{ runs converged}}{\#\text{ runs}}$

Proceedings of the 23rd EUROMICRO Conference '97 New Frontiers of Information Technology
1089-6503/97 $10.00 © 1997 IEEE

As we see, the time per individual t_{ind} is not only divided by the number of individuals, but also by the number of compute nodes n. Therefore it is the effective time needed for the calculation of one individual. Thus the parallel calculations will benefit in lower values of the time t_{ind}.

The number of individuals n_{ind} is equal to the number of offspring individuals λ of an ES. This is because all the time intensive calculations like recombination, mutation and calculation of fitness are done λ times, for each offspring individual once.

The measured times of VEES are average values of 10 up to 20 measurements, depending on the effort which is given by the absolute time needed for one measurement and the number of nodes used. For each measurement of such a series a different initialization value of the random number generator was used. The detailed parameter settings can be found in [13, 2, 4]. The type of the used strategy is listed in the subtitle of the tables in Schwefel's notation.

6.1. Function f_1

$$f_1(\vec{x}) = \sum_{i=1}^{n} x_i^2 \quad , \quad \text{with} \quad n = 3, -5.12 \leq x_i \leq 5.12 \tag{1}$$

The function f_1 is a very simple square function with only one global optimum at the origin. The calculation of this function takes a negligible amount of time and therefore this function is good for comparison of the speed of the ES implementation. For the comparison the dimension $n = 3$ was used. Because VEES and VEGA are able to find maximas only, the function was inverted, so that it has a maximum with the value 78.644.

In Fig. 3 on the left side we can see the comparison values for finding the optimum of function f_1 using one processor. VEGA needs almost 3 times more generations to reach the optimum. The reason for this is that the evolution strategies can use self-adapting mutation step lengths which are better adapted to the shape of the objective function. VEES needs more time for the calculation of one individual than VEGA, but because of the fewer generations it needs also fewer time for the calculation overall. ESCAPADE is designed to run on one processor only and is much faster for this simple function than the other programs. That means that it has fewer overhead for internal administration tasks of the program. For more complex functions it is possible that the difference in calculation speed decreases in relation to the other programs, but in the used version of ESCAPADE the other test functions f_{10} and f_{22} were not implemented.

Considering the parallel runs of VEES and VEGA in Fig. 3 on the right side in comparison with the times for one processor on the left side, we see that some more generations

$f_1(3)$	n_{ind}	n_{gen}	t_{tot} [s]	t_{gen} [ms]	t_{ind} [μs]
VEGA (1+1)	1×350	69	6,2	85	257
VEES (1+1)	1×350	25	3,47	139	397
VEES Sparc10	1×350	25	4,1	164	469
ESCAPADE Sp10	1×350	28	0,9	32,1	91,7
VEGA (16+1)	16×22	92	1,6	17,4	49,4
VEES (16+1)	16×22	33	0,906	27,5	78
MPES 16K PEs	16384×1	8	0,33	41	2,5

Figure 3. Comparison of VEGA, VEES and ESCAPADE for function f_1, with 350 (resp. 352) individuals. VEES and ESCAPADE used a (95/2, 350)–strategy, VEES on 16 nodes a (7/2, 22)–strategy.

are needed to reach the optimum, when a similar number of individuals are used but distributed to 16 nodes (16×22 vs. 1×350). The reason for this effect is that a smaller population leads to a lower speed of progress per generation. The overall computation time, however, decreases much and compensates for the small increase in generations. Because of the parallel computation, the effectively needed time for the calculation of one individual decreases by a factor of 5. The reason for this sublinear speedup is the fact that the administration overhead of the program outbalances the calculation time of the objective function for this simple function.

The program MPES needs the shortest time for this function. With the large amount of 16384 individuals[3] it is able to make a nearly optimal step towards the optimum of this low dimensional function in every generation. This is indicated by the few generations needed. As expected, the calculation time for one individual is much less than with the other programs because of the large number of processors.

6.2. Function f_9

Ackley's function: $\min(f_9) = f_9(0, \ldots, 0) = 0$

$$f_9(\vec{x}) = -a \cdot \exp\left(-b\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right)$$

$$- \exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} \cos(c \cdot x_i)\right) + a + e$$

$$a = 20 \quad ; \quad b = 0.2 \quad ; \quad c = 2\pi \quad ; \quad e = 2.71828$$

[3] it would have taken additional time to prevent their use because of the SIMD architecture

$f_9(30)$	n_{ind}	n_{gen}	t_{tot} [s]	t_{gen} [ms]	t_{ind} [μs]	$r$conv
VEES P	1×320	85	42,6	501,1	1566	1,0
VEES P	1×20	254	8,2	32,3	1616	0,45
VEES P	16×20	118	6,0	50,6	158	1,0
VEES rs	1×20	267	2,8	10,5	524	0,59
VEES rs	4×20	145	3,5	23,9	299	1,0

Figure 4. Comparison for function f_9. Abbreviations: P - Intel Paragon, rs - IBM rs6000. For every column 20 independent measurements were made, for VEES on one node of a rs6000, 100 measurements were made. Convergence was reached when the optimum 0 was approximated with a difference less than $1 \cdot 10^{-6}$. We used a (45/2, 320) resp. a (5/2, 20) ES on every node and intermediate recombination for the object variables and the step size. Exchange parameters where: exchange interval = 5 generations, exchange rate = 0.2, exchange topology = full connected.

$$n = 30 \quad ; \quad -32.768 \leq x_i \leq 32.768$$

Ackley's function is a continuous and multimodal function. It consists of a base exponential function which is modulated by a cosine wave. We use it here in the generalized, scalable version of Schwefel [12] with dimension $n = 30$.

This function is high dimensional and multimodal, so we want to look at it under the aspect of what we can gain from parallelism. In the first column we have considerably high computation time on one node, but all runs converge to the global optimum. When we reduce the number of child individuals to 20, the computation time decreases, but only about half the runs converge. Using 16 nodes, we have both: low computation time and high convergence rate.

6.3. Function f_{10} (TSP)

The third test function we used is the *Traveling Salesman Problem*. The task is to find the shortest route that visits each of 100 cities once. For this special problem the shortest route is known and amounts to 21285 length units. The cities are given by their coordinates in the plane. We used here only a simple position encoding of the problem without special recombination operators etc., which is not able to find the optimal route of this NP-complete problem. So we can compare the quality of the found solution, too. To compare the calculation time for this function, in the first half of the table we calculated constantly 1000 generations, but the algorithms converged in about half of it to a final value.

$f_{10}(100)$	n_{ind}	n_{gen}	t_{tot} [s]	t_{gen} [ms]	t_{ind} [μs]
VEGA	64×256	1000	1320	1,32	80,56
VEES	64×256	1000	2063	2,064	125,9
VEES	36×100	320	200	0,625	174
VEES	64×256	360	664	1,844	112
MPES	16384×1	580	621	1,071	65,4

$f_{10}(100)$	n_{ind}	best route
VEGA	64×256	29498
VEES	64×256	23935
VEES	36×100	22280
VEES	64×256	22235
MPES	16384×1	28497

Figure 5. Comparison of calculation time for the TSP. The optimal route is 21285. The VEES strategy was of type (70/3, 256) resp. (27/3, 100).

The position encoding is as follows: All towns are numbered, each town number is assigned one object variable and each object variable is a ordinal number of its town. To calculate the town sequence, the object variables are sorted in ascending order, but they keep the implicit assignment to their town. After sorting, the sequence of assigned town numbers gives the route to visit the towns. In this algorithm an evolution strategy uses real-valued object variables and a genetic algorithm uses integer-valued variables for the ordinal numbers. So in genetic algorithms it is possible that two towns have the same ordinal numbers. In this case it depends on the ordering algorithm which town is visited first. With real valued ordinal numbers like in evolution strategies this case is very unlikely.

VEES needs about 50% more time for an individual than VEGA and thus for the whole run. This is due to the 64 bit *double* floating point values used in VEES for each of the 100 cities of an individual, in contrast to the 7 bit integer values used in VEGA. This may also be the reason for the much shorter route found by VEES. The floating point values leave more freedom for fine position changements of the towns relative to each other than the integer interval $[0, 127]$.

The table in Fig. 5 gives a comparison between VEES and MPES. With both programs 16384 individuals were calculated overall, in VEES they were distributed to 64 processors. The total time of VEES needed for finding the best

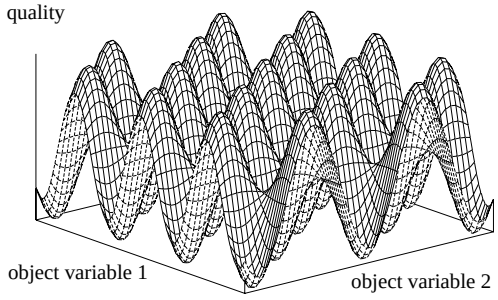

Figure 6. Function f_{22} for the 2-dimensional case, inverted.

$f_{22}(10)$	n_{ind}	n_{gen}	t_{tot} [s]	t_{gen} [ms]	t_{ind} [μs]
VEGA P	1×3200	290	570,4	1967	615
VEES P	1×3200	106	300,3	2833	885
VEGA S10	1×3200	164	407,2	2483	776
VEES S10	1×3200	106	268,4	2532	791
VEGA P	64×50	304	30,61	100,7	31,5
VEES P	64×50	108	13,6	126	39,4
MPES	16384×2	51	10,02	196,5	6

Figure 7. Comparison with function f_{22}. Abbreviations: P - Intel Paragon, S10 - Sparc 10. The used strategy was of type (864/2, 3200) resp. (14/2, 50).

solution with this configuration was only little longer than that needed by MPES. This shows that distributed evolution strategies can reach equally good results as the massively parallel ones. Additionally the shortest route found in this run by VEES was much shorter than that of MPES and it needed less generations as well.

The measurement in the left column of Fig. 5 shows that VEES is also able to find a comparable route with much less effort in time and number of individuals.

At this function a big influence of using recombination of individuals could be seen. Without recombination were found only routes that were about 10.000 units worse than those found with recombination $\rho = 2$. Increasing the number of recombinants to $\rho = 3$ brought again a visible improvement of the solutions. Values of $\rho > 3$ only increased calculation time but brought no considerable effect.

6.4. Function f_{22}

Function f_{22} (Fig. 6) is a 10-dimensional function with many suboptima that differ only a little from each other and from the global optimum. Therefore the difficulty of this function for the evolutionary algorithms is to find the global optimum and not to converge into a local one.

$$f_{22}(\vec{x}) = \frac{1}{d}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$$
$$\text{with} \quad n = 10, d = 4000, -600 \leq x_i \leq 600$$

The measurements with one processor (upper 4 rows of Fig. 7) brought the following results:

- VEES needs less generations and thus less total calculation time as VEGA,
- the time per generation and per individual of VEES is greater,
- with VEGA only about 15% of the runs converged to the global optimum, with VEES parameter settings were found, which had a convergence rate of 100%.

Using the same number of individuals distributed to 64 nodes brought a factor 23 speedup in both programs, VEGA and VEES. The number of generations needed did not increase, so that the time per individual t_{ind} decreased by the same factor. On 64 nodes the convergence rate of the single measurements of VEES was not as sensitive to little changes in the parameter settings as using only one node.

MPES needs about half of the generations of VEES, but because of the fewer individuals used in VEES, the overall computation time is only slightly greater. MPES also has no difficulties finding the global optimum.

7. Conclusions, Outlook

In summary, we obtained the following results:

- distributed ES execution on a MIMD computer is able to speedup the ES significantly,
- the global optimum is found with a higher probability even if there exist many similar suboptima,
- in our benchmarks the solutions found with the distributed ES were often much better than the ones found with a one-processor version.

The advantage of distributed evolution strategies over the massively parallel ones is that they are also able to run on

a single-processor workstation and not only on an expensive parallel computer. So it is possible to develop, test and debug new applications on a workstation and to use the parallel computer only for the optimization runs.

EvA is a user-friendly and powerful tool for optimization tasks with evolutionary algorithms. It provides menu-driven full implementations of distributed and massively parallel genetic algorithms and evolution strategies. Additionally the language Tcl is included, so that powerful batch scripts can be written that control the optimization runs.

Until now EvA was not applied to large real-world applications. We therefore are eager to get in contact with researchers who need such a powerful ES optimization system.

8. Acknowledgments

Some of this research was performed while both authors were at the University of Stuttgart, IPVR. We would like to thank the other members of the former EvA team: Roland Baumann [2] and Alex Hummler [9] for their distributed and massively parallel genetic algorithms (VEGA and MPGA), Marc Ebner [3] for his genetic algorithms on the neurocomputer Adaptive Solutions CNAPS called CNGA, Steffen Görzig [4] for his massively parallel evolution strategies, MPES, and Alexander Hasel [5] who combined all these with a powerful and easy to use graphical user interface under X11/Motif.

References

[1] T. Bäck. A User's Guide to GENEsYs 1.0. Technical report, University of Dortmund, Department of Computer Science, System Analysis Research Group, 1992.

[2] R. Baumann. Verteilte Genetische Algorithmen auf dem MIMD–Parallelrechner Intel Paragon. Diplomarbeit Nr. 1227, Universität Stuttgart, IPVR, 1995.

[3] M. Ebner. Parallele Genetische Algorithmen auf einem Neurocomputer. Studienarbeit Nr. 1478, Universität Stuttgart, IPVR, 1995.

[4] S. Görzig. Massiv Parallele Evolutionsstrategien auf dem Parallelrechner MasPar MP-1. Diplomarbeit Nr. 1291, Universität Stuttgart, IPVR, 1995.

[5] A. Hasel. Eine graphische Oberfläche für Parallele Genetische Algorithmen und Evolutionsstrategien. Diplomarbeit Nr. 1301, Universität Stuttgart, IPVR, 1995.

[6] F. Hoffmeister. *Scalable Parallelism by Evolutionary Algorithms*, pages 177–198. Number 367 in Lecture Notes in Economics and Mathematical Systems. Springer, Berlin, Heidelberg, 1991.

[7] F. Hoffmeister. *The User's Guide to* ESCAPADE *1.0 — A Runtime Environment for Evolution Strategies*. University of Dortmund, Department of Computer Science XI, P.O.Box 500 500, D-4600 Dortmund 50, Germany, October 22 1991.

[8] F. Hoffmeister and H.-P. Schwefel. KORR *2.1 — An Implementation of a* (μ^{+}, λ) *– Evolution Strategy*. University of Dortmund, Department of Computer Science XI, P.O.Box 500 500, D-4600 Dortmund 50, Germany, October 22 1991.

[9] A. Hummler. Massiv parallele Genetische Algorithmen auf dem Parallelrechner MasPar MP-1. Diplomarbeit Nr. 1240, Universität Stuttgart, IPVR, 1995.

[10] J. K. Ousterhout. *Tcl und Tk. Entwicklung grafischer Benutzerschnittstellen für das X Window System*. Professional Computing. Addison–Wesley, 1995.

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

[12] H.-P. Schwefel. *Evolution and optimum seeking*. Sixth-Generation Computer Technology Series. John Wiley & Sons, INC., 1995.

[13] J. Wakunda. Verteilte Evolutionsstrategien auf dem Supercomputer Intel Paragon. Diplomarbeit Nr. 1300, Universität Stuttgart, IPVR, 1995.

[14] A. Zell and R. Baumann. Distributed Genetic Algorithms on the Intel Paragon, a large MIMD Computer. In T. B. L. J. Fogel, P. J. Angeline, editor, *Proc. of the 5th Ann. Conf. on Evolutionary Programming*, San Diego, CA, Feb. 29-March 2 1966.