Parallel Metaheuristics: A New Class of Algorithms
Edited by Enrique Alba

7 Parallel Evolution Strategies

GÜNTER RUDOLPH[†]
Parsytec Solutions GmbH, Erkelenz, Germany

7.1 INTRODUCTION

The term 'Evolution Strategy' (ES) denotes a branch of Evolutionary Algorithms (EAs) that emerged and initially developed independently [22, 23, 34] from other branches like Evolutionary Programming (EP) [11] and Genetic Algorithms (GAs) [16]. In their contemporary form the typical field of application of ES and EP focuses on parameter optimization with real variables [9, 10, 25, 35]. Therefore the scope of this chapter comprises parallel EAs, with individuals encoded by vectors of floating-point numbers. Not surprisingly, the architectural design of parallel ES/EP hardly differs from parallel GAs. But due to the characteristic feature of a self-adaptive control of the mutation strength, there are parallel designs working fine for GAs (with fixed mutation strength) for which parallel ES/EP fail. Such cases and their cure will be addressed in the course of this chapter.

Historically, probably the first non-sequential computer running an ES was the vector computer CYBER 205 [3] in 1984. This may be seen as the beginning of the area of parallel ES/EP. With the advent of affordable parallel hardware in the late 1980s the dream of exploiting the inherent parallelism of EAs came true. First parallel versions of ES were realized in a local area network [17], on Parsytec transputer systems under OCCAM [4] or under C/Helios [27, 28], and on the Connection Machine 2 (CM2) under C* [30]. Parallel versions of EP were implemented on an Intel Paragon iPSC/860 [6] and a MasPar MP-1 [20].

Remarkably, none of these early parallel versions was a pure parallelization of the sequential version despite the parallelism inherent to a population of individuals. Rather, the new parallel designs were closely aligned to the parallel target hardware. As a result, the availability of parallel hardware with different architectures had led to entirely new EAs!

[†] The views expressed in this chapter are those of the author and do not reflect the official policy or position of Parsytec Solutions GmbH.

According to Flynn's well-known classification of parallel hardware [8] the designs of parallel ES/EP were tailored to SIMD (MP-1, CM2) and MIMD (Paragon, transputer) computers. This is why the taxonomy of parallel EAs proposed in [15, 14] distinguished between parallelism at the level of subpopulations (MIMD) and at the level of individuals (SIMD) and between synchronous and asynchronous communication. Viewed from the biological angle, the parallel models can be categorized into migration (MIMD) and pollination (SIMD) models. Other terms commonly used for the latter model are neighborhood or diffusion model. The presentation of parallel ES/EP given in Section 7.4 is oriented at this classification, too. Section 7.2 is devoted to the question of under which circumstances a parallelization of randomized optimization algorithms is useful at all.

7.2 DEPLOYMENT SCENARIOS OF PARALLEL EVOLUTIONARY ALGORITHMS

The utility of a parallel deterministic optimization is evident: Since the deterministic algorithm is run only once, the parallel version delivers the solution more rapidly. In case of randomized optimization algorithms like EAs the situation changes. Moreover, our typical measures for speedup and efficiency of parallel versions must be changed [1]. Since randomized algorithms are run multiple times usually it is advisable to verify that the burden of developing a parallel randomized algorithm is worth the effort. This will be exemplified for two typical deployment scenarios.

7.2.1 Scenario: Run EA Multiple Times, Choose Best Solution Found

In practice, nobody runs a randomized algorithm like an EA only once. Rather, the EA is run multiple times and the best solution found within some time limit is used.

7.2.1.1 Fixed Generation Number. Let t be the running time of the sequential algorithm and $t_p = ct/p$ the running time of the parallelized sequential algorithm, where $c > 1$ aggregates the communication and other overhead costs of the parallelized version. Let n be the maximum number of times we can run the EA before we must use the best solution found and assume that $n = p$, where p is the number of processors.

Then $r = t$ is the total running time of running the sequential algorithm on p processors in parallel. Since the total running time of p sucessive runs of the parallelized version is $r_p = p \times t_p = ct$, we can see easily that nothing is gained by a parallelization. Even worse, every effort invested in this task is a waste of resources.

7.2.1.2 Random Generation Number. The situation changes if the running time of the EA is a random variable. Let T be the random running time of the sequential algorithm and $T_p = cT/p$ the running time of the parallelized sequential algorithm with $c > 1$. Again, assume $n = p$. Then the random total running time R of running

the sequential algorithm on p processors in parallel is

$$R = \max\{ T(1), T(2), \ldots, T(p) \} = T_{p:p}$$

where $T(i)$ is the running time at processor i. Clearly, the $T(i)$ are independent and identically distributed. Assume that $T(i)$ is normally distributed with mean $t > 0$ and variance σ^2. The expectation of R can be approximated [5] via

$$\mathsf{E}[\, R \,] = \mathsf{E}[\, T_{p:p} \,] \approx \mathsf{E}[\, T \,] + \mathsf{D}[\, T \,] \sqrt{2 \log p}.$$

The random total running time R_p of p sucessive runs of the parallelized version is given by

$$R_p = \sum_{i=1}^{p} T_p(i) = \frac{c}{p} \sum_{i=1}^{p} T(i)$$

with expectation

$$\mathsf{E}[\, R_p \,] = c\, \mathsf{E}[\, T \,].$$

Thus, the parallelized version is faster if

$$\mathsf{E}[\, R_p \,] < \mathsf{E}[\, R \,] \Leftrightarrow c < 1 + \frac{\mathsf{D}[\, T \,]}{\mathsf{E}[\, T \,]} \times \sqrt{2 \log p}.$$

In other words, the larger is the coefficient of variation $\nu = \mathsf{D}[\, T \,] / \mathsf{E}[\, T \,]$ the larger the benefit achieved by the parallelization of the sequential algorithm! As seen from this analysis, this scenario can be an appropriate field of deployment of parallelized EAs.

7.2.2 Run Until Satisfactory Solution

One might argue that the previous scenario is not always the case. For example, if we need only a satisfactory solution, then we can stop the EA as soon as such a solution has been detected. In principle, this can happen in a single run of the EA.

7.2.2.1 ***Fixed Generation Number.*** As in the previous scenario let t be the running time of the sequential algorithm and $t_p = ct/p$ the running time of the parallelized sequential algorithm with $c > 1$. Suppose there exists a success probability $s \in (0, 1)$ for each run of the EA such that the random variable G represents the number of runs until a successful run occurs. The random variable G has geometrical distribution with probability function

$$\mathsf{P}\{\, G = k \,\} = s\,(1 - s)^{k-1}$$

for $k = 1, 2, \ldots$ and $s \in (0, 1)$ with

$$\mathsf{E}[\, G \,] = \frac{1}{s} \quad \text{and} \quad \mathsf{D}^2[\, G \,] = \frac{1 - s}{s^2}.$$

The time until a successful run occurs on a single processor is $S = t\,G$. Therefore, the random total running time R of running the sequential algorithm on p processors in parallel is

$$R = \min\{\, S(1), S(2), \ldots, S(p) \,\} = S_{1:p} = t\,G_{1:p}$$

where $G_{1:p}$ denotes the minimum of p independent and identically distributed geometrical random variables. According to [45] we have

$$\mathsf{E}[\,G_{1:p}\,] = \frac{1}{1-(1-s)^p} \quad \text{and} \quad \mathsf{D}^2[\,G_{1:n}\,] = \frac{(1-s)^n}{[\,1-(1-s)^n\,]^2}$$

such that

$$\mathsf{E}[\,R\,] = t\,\mathsf{E}[\,G_{1:p}\,] = \frac{t}{1-(1-s)^p}\,.$$

The random total running time R_p of p sucessive runs of the parallelized version is given by

$$R_p = t_p\,S = \frac{c}{p}\,t\,S$$

with expectation

$$\mathsf{E}[\,R_p\,] = \frac{c}{p}\,t\,\mathsf{E}[\,S\,] = \frac{c\,t}{s\,p}\,.$$

Since

$$\mathsf{E}[\,R_p\,] < \mathsf{E}[\,R\,] \Longleftrightarrow c < \frac{s\,p}{1-(1-s)^p}$$

, there are constellations in which a parallelized version is useful.

7.2.2.2 ***Random Generation Number.*** Let $T(i)$ be the random running time of run i. Then

$$S = \sum_{i=1}^{G} T(i)$$

is the random time until the first successful run on a single processor. Since G is a stopping time we have $\mathsf{E}[\,S\,] = \mathsf{E}[\,G\,]\,\mathsf{E}[\,T\,]$. As a consequence, the random total running time R of running the sequential algorithm on p processors in parallel is

$$R = \min\{\, S(1), S(2), \ldots, S(p) \,\} = S_{1:p}$$

with

$$\mathsf{E}[\,R\,] = \mathsf{E}[\,S_{1:p}\,] < \mathsf{E}[\,S\,] = \mathsf{E}[\,T\,]\,\mathsf{E}[\,G\,]\,.$$

The random total running time R_p of p contiguous runs of the parallelized version is given by

$$R_p = \sum_{i=1}^{G} T_p(i) = \frac{c}{p} \sum_{i=1}^{G} T(i)$$

with

$$\mathsf{E}[R_p] = \frac{c}{p}\,\mathsf{E}[T]\,\mathsf{E}[G] = \frac{ct}{sp}.$$

Since certainly $\mathsf{E}[R] \geq \frac{1}{p}\mathsf{E}[S]$ a parallelization under this scenario is worth considering as well.

7.3 SEQUENTIAL EVOLUTIONARY ALGORITHMS

The similarities and 'historical' differences between ES and EP are described in [2]. Contemporary versions of ES and EP deploy the same self-adaptive control of the mutation strength and they differ only in the selection method and in the renunciation of recombination operators in case of EP. The general skeleton of ES/EP is given below:

Algorithm 1. ES/EP Algorithm

initialize population of μ individuals and evaluate them
  repeat
    generate λ offspring from μ parents
    evaluate offspring
    select μ new parents from offspring (and possibly parents)
  until stopping criterion fulfilled
output: best individual found

7.4 PARALLEL EVOLUTIONARY ALGORITHMS

7.4.1 Parallelization of Sequential Algorithm

The parallelism inherent in the sequential ES/EP can be found in the generation and evaluation of offspring. The nature of generational selection procedures requires that all offspring have been generated and evaluated before selection can begin. Therefore, an obvious approach to parallelize the sequential version on MIMD computers is given by the farmer/worker model: One distinguished processor is the farmer that runs the main logic of the algorithm and distributes the working packages to the worker processors. In case of ES/EP the farmer generates the initial population and evaluates them by sending the individual to an idle worker. The worker evaluates the individual and returns its fitness value to the farmer. After the initialization the farmer sequentially generates a new offspring by random variation (mutation and possibly recombination) and sends it to an idle worker. A concurrent thread of the farmer process is responsible for collecting the fitness values delivered by the workers. If

the farmer has sent all offspring to the worker, it waits until the concurrent thread signals that all offspring have been evaluated. Now the farmer applies a selection method in ES or EP fashion to determine the parents of the next iteration.

Needless to say, this approach leads to an efficient parallel ES/EP if the evaluation of the fitness function requires much time. In light of Section 7.2 a parallelization is useful if the evaluation of the objective/fitness function requires non-constant time. This may be the case if the evaluation requires an adaptive numerical integration procedure. In [7] a (μ, λ)-ES has been parallelized for optimizing a multibody system requiring numerical time integration of the equations of motion.

The PEPNet system [26] uses a parallel version of EP to optimize artificial neural networks. The farmer generates the initial population before sending groups of individuals to the worker. Now each worker runs a sequential EP for a prescribed number of generations and returns its group of evolved individuals. After all workers have delivered their evolved group of individuals the farmer selects the best solution for output and halts. Essentially, this is nothing more than running p instances of the sequential EP on p processors in parallel. Since the number of iterations is fixed and fitness evaluation needs almost constant time, this approach is the most efficient one actually (as can be seen from Section 7.2).

7.4.2 Migration Model

In the migration model the population is divided in p subpopulations, where p is the number of processors available. Each subpopulation runs a standard ES/EP and exchanges individuals from time to time. In the design phase you must make decisions about the

1. migration paths (which subpopulations exchange individuals?),
2. migration frequency (how often migration takes place?),
3. number of migrants (how many individuals are exchanged?),
4. selection policy for emigrants (which individuals leave the subpopulation?),
5. integration policy for immigrants (how to integrate the arriving individuals?).

In [28] the subpopulations were arranged in a bidirectional ring. Migrations took place every kth generation and a prefixed number of individuals are sent to each neighboring subpopulation. The individuals with rank 1, 3, 5, ... were sent to the left whereas the individuals with rank 2, 4, 6, ... were sent to the right neighboring subpopulation. The immigrants took the empty slots of the emigrants and the next generation began with recombination and mutation.

A variant of this ES was proposed in [33]: the immigrants remain unchanged until new immigrants arrive. The idea was to give them a chance to establish their genes in the new environment. Apparently, this idea proved useful for the application (mixed-integer optimization of the coating of optical multi-layer filters).

In the parallel EP described in [6] the subpopulations were placed at the vertices of a d-dimensional hypercube with $d = 6$. Migration took place after each generation as follows: Each subpopulation was divided into $d + 1$ groups of individuals by random selection. One group stayed at the processor whereas the other d groups were sent to the neighboring subpopulations. The immigrants took the place of the emigrants.

The subpopulations of the parallel ES proposed in [12] were arranged in a ring, but the migration paths were unidirectional. Every kth generation a *copy* of the best individual emigrates and the immigrant *replaces* the worst individual in the subpopulation. Notice that the emigrants in the previous versions actually left the processors. Here, only copies are sent.

The parallel ES presented in [21] divided the population in p subpopulations on p processors. One distinguished processor is the 'master' with additional responsibilities. Each subpopulation selects every kth generation the γ best individuals and sends copies of them to the master before starting the next iteration. A concurrent thread at the master asynchronously receives $(p - 1)\,\gamma$ individuals. As soon as all individuals have been collected the master selects the γ best of them and broadcasts them to all subpopulations. A concurrent thread at each subpopulation receives the γ individuals from the master and replaces the γ worst by those individuals just received.

A similar approach was realized in the parallel ES/GP hybrid developed in [44]. The blackboard architecture was used to 'publish' about 10% of the best individuals of each subpopulation. The topology of the migration paths of such approaches is the fully connected graph with p nodes. It seems plausible that the topology of the migration network has important impact on the diversity of the gene pool: It is expected that smaller diameters increase the risk of premature convergence but systematic studies are not available apparently.

The most recent migration model was given in [40]. As in [12] the population was arranged in a unidirectional ring. Each subpopulation ran a non-standard EP: Each parent generates two intermediate offspring. The first has normally-distributed mutations whereas the second has Cauchy-distributed mutations. Both intermediate offspring are evaluated and the best of them is the offspring. Thus, the variation step is already on generation of a $(1, 2)$-ES. After all offspring are generated the parents are selected in EP-style and the next generation starts. Every k generation the γ best or randomly selected individuals are sent to the other node. The immigrants either replace the worst or randomly selected individuals.

The EvA toolbox [41] also contains a parallel ES according to the migration model but no details are given. Migration models were also simulated on sequential computers. For example, experiments with 100 communicating $(1, 10)$-ES are reported in [18].

7.4.3 Pollination Model

The pollination model assumes that the individuals do not move but that the genetic information is spread by means of pollination. This situation is closely matched by a SIMD computer. Each individual is placed on a processor and the interaction

takes place in its neighborhood defined by an underlying graph structure. The most obvious neighborhood graph on a SIMD computer is the 2D torus. But the tight coupling between parallel ES/EP design and target hardware was broken in [37] by mapping a pollination model to a MIMD computer. The advantages are twofold: First, the population size on a SIMD computer is limited to multiples of the number of processors. Second, the population size can be scaled easily to achieve high efficiency on a MIMD computer.

This idea was adopted in the parallel pollination ES described in [29], where each individual selects a mate for recombination from its neighborhood. After recombination the offspring is mutated and evaluated. The offspring replaces its parent at the current location if it is better.

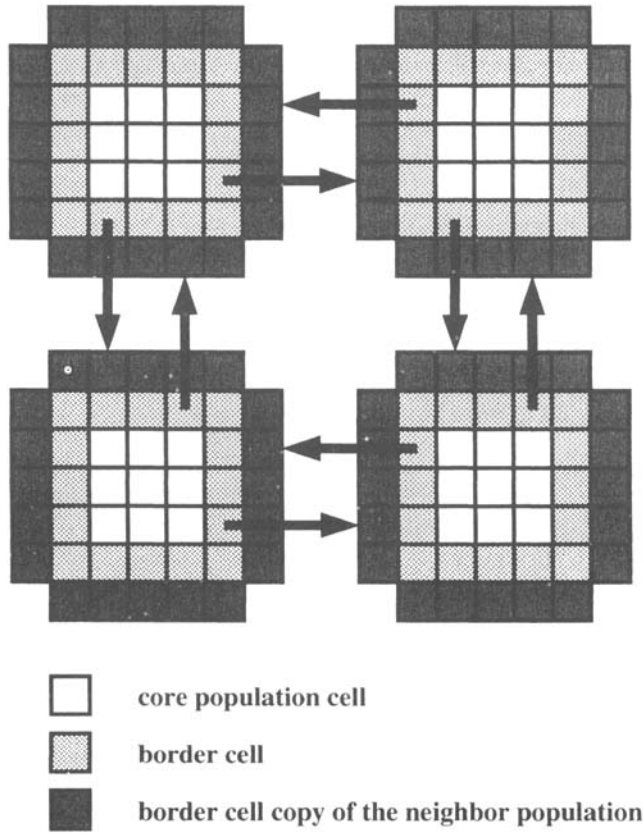

Fig. 7.1 Implementing a pollination model on a MIMD computer.

Figure 7.1 is intended to illustrate how the pollination model algorithm has been mapped to the MIMD machine. After initialization of the core and border population cells the border cells are sent to the neighboring subpopulations. Now all relevant information for one iteration is known on each processor and the generation and evaluation of new sample points for each cell placed on one processor can be computed

sequentially. If this is done, the border cells are sent to the neighboring processors again and the next iteration can be computed. This parallel ES was also realized on a true SIMD computer as reported in [30].

Another example of running the pollination model on MIMD computers is given in [38]. Here, the underlying topology is a ring, but the neighborhood of each individual has a certain radius, i.e., r individuals to the right and left. At each iteration and for each processor the two best individuals are taken from the processor's neighborhood. The selected individuals are recombined to an offspring that is mutated and evaluated. If it is better than the individual at the processor, it replaces that individual. Another version generates multiple offspring per processor and accepts only the best of them.

A parallel EP was realized on a MasPar MP-1 [20], which is also the target machine of the pollination model realized in the EvA toolbox [41]; unfortunately, no details are given.

An extension of the pollination model was presented in [43]. Here, the static neighborhood structure was replaced by a dynamic one. Now one needs rules for establishing (coupling) and cutting (decoupling) a connection. A connection to the neighbor is detached if recombination with that neighbor yields an offspring that is worse than the best offspring generated at the processor during the current generation. If the number of connections of an individual falls below a certain threshold, a certain number of new connections to randomly chosen individuals are established. Notice that individuals cannot become isolated, in contrast to groups of connected individuals. But this kind of isolation is finished as soon as a randomly generated connection is established by the coupling rule.

Needless to say, due to its lack of locality this algorithm is not well suited for SIMD machines. Rather, this approach is tailored for multiprocessor machines with shared memory.

7.4.4 Non-generational Models

Both the migration and pollination models represent generational ES/EP. All offspring must be generated and evaluated before selection can take place. Non-generational or steady-state or $(\mu + 1)$-ES do not have this potential synchronization bottleneck. Moreover, this model is well-suited for multi-processor computers with shared memory. But there is a drawback: The self-adaptive control of mutation strength does not work as desired. This problem was addressed by [42] and [31]. Here, we focus on the latter solution.

After the generation and evaluation of the initial population of size μ, each of the p processors selects the best and worst unblocked individual and blocks them. The best individual generates an offspring by mutation and is unblocked. The offspring is evaluated and replaces the worst individual, whose slot is unblocked.

The special handling of the mutation control was developed in [32]. A reproduction counter r_i is added to each individual. If an individual did not generate a better offspring in $R/2$ reproduction events, then the step size is considered wrong and it is adjusted via the weighted average with the mutation strength of the grandparent:

Something that was good in the past cannot be completely wrong in the near future. If this adjustment also does not lead to a better offspring within $R/2$ further reproduction events, then the individual will be discarded from the population. This kind of control of mutation strength seems to work for non-generational ES.

7.4.5 Nested Populations

The shorthand notation $(\mu \dot{+} \lambda)$–ES was extended in [24] to the expression

$$[\mu' \dot{+} \lambda' (\mu \dot{+} \lambda)^{\gamma}]^{\gamma'}\text{–ES}$$

with the following meaning: There are μ' populations of μ parents. These are used to generate (e.g., by merging) λ' initial populations of μ individuals each. For each of these λ' populations a $(\mu \dot{+} \lambda)$–ES is run for γ generations. The criterion to rank the λ' populations after termination might be the average fitness of the individuals in each population. This scheme is repeated γ' times. The obvious generalization to higher levels of nesting is described in [25], where it is also attempted to develop a short–hand notation to specify the parametrization completely.

This nesting technique is of course not limited to evolution strategies: other evolutionary algorithms and even mixtures of them can be used instead. In fact, the somewhat artificial distinction between ES, GA, and EP becomes more and more blurred when higher concepts enter the scene.

It is reported in [41] that the EvA toolbox provides a parallel implementation of nested populations. This opens the door to several fields of application:

1. **Alternative method to control internal parameters.** Herdy (1992) [13] used λ' subpopulations, each of them possessing its own different and fixed step size σ. Thus, there is no step size control at the level of individuals. After γ generations the improvements (in terms of fitness) achieved by each subpopulation is compared to each other and the best μ' subpopulations are selected. Then the process repeats with slightly modified values of σ. Since subpopulations with a near–optimal step size will achieve larger improvements, they will be selected (i.e., better step sizes will survive) resulting in an alternative method to control the step size.

2. **Mixed–Integer optimization.** Lohmann (1992) [19] considered optimization problems in which the decision variables are partially discrete and partially continuous. The nested approach worked as follows: The evolution strategies in the inner loop optimized over the continuous variables while the discrete variables were held fixed. After termination of the inner loop, the EA in the outer loop compared the fitness values achieved in the subpopulations, selected the best ones, mutated the discrete variables, and passed them as fixed parameters to the subpopulations in the inner loop.

   It should be noted that this approach to mixed integer optimization may cause some problems: In essence, a Gauß–Seidel–like optimization strategy

is realized, because the search alternates between the subspace of discrete variables and the subspace of continuous variables. Such a strategy must fail whenever simultaneous changes in discrete *and* continuous variables are necessary to achieve further improvements.

3. **Minimax optimization.** Sebald and Schlenzig (1994) [36] used nested optimization to tackle minimax problems of the type

$$\min_{x \in X}\{ \max_{y \in Y}\{ f(x,y) \} \}$$

where $X \subseteq \mathbb{R}^n$ and $Y \subseteq \mathbb{R}^m$. Equivalently, one may state the problem as follows:

$$\min\{ g(x) : x \in X\} \quad \text{where} \quad g(x) = \max\{ f(x,y) : y \in Y\}.$$

The evolutionary algorithm in the inner loop maximizes $f(x,y)$ with parameters x, while the outer loop is responsible to minimize $g(x)$ over the set X.

7.5 CONCLUSIONS

The most popular parallelization model for ES/EP in the past was the migration model. Today, migration and pollination models may be seen as extreme cases: as shown in [39] both models can be integrated in a general population model. More recent parallel realizations of ES/EP move into that direction. An important task for the future is the systematic test of the different parallel versions of ES and EP. Even if an analysis of the problem and deployment scenario support the use/development of a parallel ES/EP, one is unable to recommend a certain class. Summing up: there is the need of extending and founding our knowledge about parallel ES/EP.

REFERENCES

1. J. Aczél and W. Ertel. A new formula for speedup and its characterization. *Acta Informatica*, 34:637–652, 1997.

2. T. Bäck, G. Rudolph, and H.-P. Schwefel. Evolutionary programming and evolution strategies: Similarities and differences. In D. B. Fogel and W. Atmar, editors, *Proceedings of the 2nd Annual Conference on Evolutionary Programming*, pages 11–22. Evolutionary Programming Society, La Jolla (CA), 1993.

3. U. Bernutat-Buchmann and J. Krieger. Evolution strategies in numerical optimization on vector computers. In Feilheimer, Joubert, and Schendel, editors,

Proceedings of the Int'l Conf. on Parallel Computing '83, pages 42–51. Elsevier, Amsterdam, 1984.

4. A. Bormann. Parallelisierungsmöglichkeiten für direkte Optimierungsverfahren auf Transputersystemen. Diplomarbeit, University of Dortmund, Department of Computer Science, 1989.

5. H. A. David. *Order Statistics*. Wiley, New York, 1970.

6. B. S. Duncan. Parallel evolutionary programming. In D. B. Fogel and W. Atmar, editors, *Proceedings of the 2nd Annual Conference on Evolutionary Programming*, pages 202–209. Evolutionary Programming Society, La Jolla (CA), 1993.

7. P. Eberhard, F. Dignath, and L. Kübler. Parallel evolutionary optimization of multibody system with application to railway dynamics. *Multibody System Dynamics*, 9(2):143–164, 2003.

8. M. J. Flynn. Very high–speed computing systems. *Proceedings of IEEE*, 54:1901–1909, 1966.

9. D. B. Fogel. *Evolving Artificial Intelligence*. PhD thesis, University of California, San Diego, 1992.

10. D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, New York, 1995.

11. L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966.

12. H. Füger, G. Stein, and V. Stilla. Multi–population evolution strategies for structural image analysis. In *Proceedings of the First IEEE Conference on Evolutionary Computation (ICEC '94)*, pages 229–234. IEEE Press, Piscataway (NJ), 1994.

13. M. Herdy. Reproductive isolation as strategy parameter in hierachically organized evolution strategies. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature, 2*, pages 207–217. North Holland, Amsterdam, 1992.

14. F. Hoffmeister. Scalable parallelism by evolutionary algorithms. In M. Grauer and D. B. Pressmar, editors, *Applied Parallel and Distributed Optimization*, pages 175–198. Springer, Berlin, 1991.

15. F. Hoffmeister and H.-P. Schwefel. A taxonomy of evolutionary algorithms. In G. Wolf, T. Legendi, and U. Schendel, editors, *Proceedings of the V. International Workshop on Parallel Processing by Cellular Automata and Arrays (Parcella '90)*, pages 97–107. Akademie–Verlag, Berlin, 1990.

16. J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

17. R. Kottkamp. Nichtlineare Optimierung unter Verwendung verteilter, paralleler Prozesse in einem Local Area Network (LAN). Diploma thesis, University of Dortmund, Department of Computer Science, February 1989.

18. R. Lohmann. Application of evolution strategies in parallel populations. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, pages 198–208. Springer, Berlin and Heidelberg, 1991.

19. R. Lohmann. Structure evolution and incomplete induction. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature, 2*, pages 175–185. North Holland, Amsterdam, 1992.

20. K. M. Nelson. Function optimization and parallel evolutionary programming on the MasPar MP–1. In A. V. Sebald and L. J. Fogel, editors, *Proceedings of the 3rd Annual Conference on Evolutionary Programming*. World Scientific, River Edge (NJ), 1994.

21. R.S. Pereira, O. R. Saavedra, and O. A. Carmona. Parallel distributed evolution strategies. In *Proceedings of the 5th Brazilian Symposium on Intelligent Automation*, pages 1034–1040. Porto Alegre, 2001.

22. I. Rechenberg. Cybernetic solution path of an experimental problem, royal aircraft establishment, library translation no. 1122, farnborough, hants., uk, 1965.

23. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann–Holzboog Verlag, Stuttgart, 1973.

24. I. Rechenberg. Evolutionsstrategien. In B. Schneider and U. Ranft, editors, *Simulationsmethoden in der Medizin und Biologie*, pages 83–114. Springer, Berlin, 1978.

25. I. Rechenberg. *Evolutionsstrategie '94.* Frommann–Holzboog Verlag, Stuttgart, 1994.

26. G. A. Riessen, G. J. Williams, and X. Yao. PEPNet: Parallel evolutionary programming for constructing artificial neural networks. In *Proceedings of the Sixth Annual Conference on Evolutionary Programming (EP97)*, pages 35–45. Springer, Berlin, 1997.

27. G. Rudolph. Globale Optimierung mit parallelen Evolutionsstrategien. Diplomarbeit, University of Dortmund, Department of Computer Science, July 1990.

28. G. Rudolph. Global optimization by means of distributed evolution strategies. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, pages 209–213. Springer, Berlin and Heidelberg, 1991.

29. G. Rudolph. Parallel approaches to stochastic global optimization. In W. Joosen and E. Milgrom, editors, *Parallel Computing: From Theory to Sound Practice,*

Proceedings of the European Workshop on Parallel Computing (EWPC 92), pages 256–267. IOS Press, Amsterdam, 1992.

30. G. Rudolph. Massively parallel simulated annealing and its relation to evolutionary algorithms. *Evolutionary Computation*, 1(4):361–382, 1994.

31. T. P. Runarsson. An asynchronous parallel evolution strategy. *International Journal of Computational Intelligence & Applications*, 3(4):381–394, 2003.

32. T. P. Runarsson and X. Yao. Continuous selection and self-adaptive evolution strategies. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002)*, pages 279–284. IEEE Press, Piscataway (NJ), 2002.

33. M. Schütz and J. Sprave. Application of parallel mixed-integer evolution strategies with mutation rate pooling. In L. J. Fogel, P. J. Angeline, and T. Bäck, editors, *Proceedings of the 5th Annual Conference on Evolutionary Programming*, pages 345–354. MIT Press, Cambridge (MA), 1996.

34. H.-P. Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Birkhäuser, Basel, 1977.

35. H.-P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.

36. A. V. Sebald and J. Schlenzig. Minimax design of neural net controllers for highly uncertain plants. *IEEE Transactions on Neural Networks*, 5(1):73–82, 1994.

37. J. Sprave. Parallelisierung Genetischer Algorithmen zur Suche und Optimierung. Diplomarbeit, University of Dortmund, Department of Computer Science, 1990.

38. J. Sprave. Linear neighborhood evolution strategies. In A. V. Sebald and L. J. Fogel, editors, *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 42–51. World Scientific, River Edge (NJ), 1994.

39. J. Sprave. A unified model of non-panmictic population structures in evolutionary algorithms. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC 1999)*, volume 2, pages 1384–1391. IEEE Press, Piscataway (NJ), 1999.

40. S. Tongchim and X. Yao. Parallel evolutionary programming. In *Proceedings of the 2004 Congress on Evolutionary Computation (CEC 2004)*, pages 1362–1367. IEEE Press, Piscataway (NJ), 2004.

41. J. Wakunda and A. Zell. EvA – A tool for optimization with evolutionary algorithms. In *Proceedings of the 23rd EUROMICRO Conference*, pages 644–651. 1997.

42. J. Wakunda and A. Zell. Median-selection for parallel steady-state evolution strategies. In M. Schoenauer, J. J. Merelo, K. Deb, G. Rudolph, X. Yao, E. Lutton, and H.-P. Schwefel, editors, *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature (PPSN 2000)*, pages 405–414. Springer, Berlin, 2000.

43. K. Weinert, J. Mehnen, and G. Rudolph. Dynamic neighborhood structures in parallel evolution strategies. *Complex Systems*, 13(3):227–243, 2002.

44. K. Weinert, T. Surmann, and J. Mehnen. Parallel surface reconstruction. In J. A. Foster, E. Lutton, J. Miller, C. Ryan, and A. G. B. Tettamanzi, editors, *Proceedings of the Fifth European Conference on Genetic Programming (EuroGP 2002)*, pages 113–122. Springer, Berlin and Heidelberg, 2002.

45. D. H. Young. The order statistics of the negative binomial distribution. *Biometrika*, 57(1):181–186, 1970.