Parallel Approaches to Stochastic Global Optimization

Günter Rudolph
rudolph@ls11.informatik.uni-dortmund.de

University of Dortmund
Department of Computer Science
P.O. Box 50 05 00
W-4600 Dortmund 50 / FRG

Abstract

In this paper we review parallel implementations of some stochastic global optimization methods on MIMD computers. Moreover, we present a new parallel version of an Evolutionary Algorithm for global optimization, where the inherent parallelism can be scaled to obtain a reasonable processor utilization. For this algorithm the convergence to the global optimum with probability one can be assured. Test results concerning speed up and reliability are given.

1 Introduction

Many real world problems in engineering and economics can be formulated as optimization problems, in which the objective function is multimodal, i.e. the problem possesses many local minima. Compared to the number of methods designed to determine a local minimum, there are only a few methods which attempt to find the global minimum (see [52] for a survey). Although there are some special cases where the global optimum can be found (see [26]) the general case is unsolvable. This paper will be restricted to the more general case and the related stochastic optimization methods. These methods are very time consuming as soon as the dimensionality of the problem increases. Thus, the availability of parallel computers is a challenge in the field of global optimization.

In section 2 we state the problem formally and introduce some basic terms frequently used. A survey of some popular stochastic global optimization methods is given in section 3, where we discuss also the parallel versions for MIMD computers from the literature. In section 4 we consider so–called Evolutionary Algorithms as global minimization methods. For a special parallel version we discuss the reliability of the method itself and the efficiency of the parallel mapping onto a transputer system.

2 Global Optimization

DEFINITION 1:
The value $f(x^*)$ of a real–valued function $f : M \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ with $M \neq \emptyset$ and $x^* \in M$ is called the *global minimum* of f, iff

$$\forall x \in M \ : \ f(x^*) \leq f(x) \ .$$

Then, the point x^* is called a *global minimizer* of f. The task to determine the global minimum of f is said to be the *global optimization problem*:

$$\text{determine } f^* := \min\{f(x) : x \in M\} \ . \tag{1}$$

The function f is called the *objective function* and M the *feasible region.* □

Due to the limited accuracy of digital computers the global optimization problem (1) is considered as solved, if we can determine for some $\epsilon > 0$ an element of the level set

$$L_{f^*+\epsilon} := \{x \in M : f(x) \leq f(x^*) + \epsilon\} \ . \tag{2}$$

Moreover, we will assume that the global optimum is not isolated: $\mu(L_{f^*+\epsilon}) > 0$, where $\mu(.)$ denotes the Lebesgue measure, and that the feasible region is bounded.

Most methods developed so far in the field of optimization in $\mathbb{R}^n$ are not able to solve (1), because they usually determine a *local minimum*, that is closest to the starting point of the search.

DEFINITION 2:
The value $f(x^*)$ of a real–valued function $f : M \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ with $M \neq \emptyset$ and $x^* \in M$ is called a *local minimum* of f, if

$$\exists \epsilon > 0 : \forall x \in M : ||x - x^*|| < \epsilon \Rightarrow \ f(x^*) \leq f(x) \ .$$

Then, the point x^* is said to be a *local minimizer* of f. □

From this definition it is easy to see, that every global minimum is also a local minimum but not vice versa, which causes the difficulty of the global optimization problem, because there is no *constructive* criterion to decide whether a detected local minimum is the global minimum. This is only possible in special cases (see e.g. [26]), which will be not the problem of interest, here.

3 Stochastic Global Optimization Methods

In this section we will review those stochastic global optimization methods for which a parallel implementation on MIMD computers can be found in the literature. We start with the simplest method called *Monte Carlo Search* and continue in proposing those variants which improve and refine this simple method in order to achieve a better algorithmic performance. Not surprisingly these refinements make the parallelization more difficult, because they have an inherent sequential character. These difficulties can be bypassed by *Parallel Evolutionary Algorithms* described in section 4.

3.1 Monte Carlo Search

Monte Carlo Search can be considered as the stochastic pendant of deterministic Grid Search. The idea of the sequential version proposed by [12] is to sample N points uniformly distributed over the feasible region M and to evaluate them. Then, the sample point with the lowest objective value is considered as the candidate solution. It has been shown by [16], that this algorithm will converge to the global minimum under some conditions:

THEOREM 1:
Let $\tilde{f}_N$ be the best objective value found in a sample of size N. Then $\tilde{f}_N$ converges to the global minimum f^* with probability one with increasing N, if f is continuous and $\mu(L_{f^*+\epsilon}) > 0$:

$$Prob\left\{\lim_{N\to\infty} \tilde{f}_N = f^*\right\} = 1 \quad . \tag{3}$$

□

The algorithm can be parallelized very easily with $N = k \cdot P, k \in \mathbb{N}$, where P denotes the number of processors: Choose N/P sample points uniformly distributed over M and evaluate them. Then, the sample point with the lowest objective value over all processors will be the candidate solution. It is not hard to see that the speedup will be linear – or to be more precise: the efficiency of the parallel version will be close to 100 %. However, a short analysis demonstrates [44] that the method itself is very inefficient, because we need

$$N \approx -\ln(1-p_N)\left(\frac{R}{\epsilon}\right)^n \tag{4}$$

trials, such that at least one of N sample points falls within the ϵ–neighborhood of the global minimizer with probability p_N. Here, R denotes the radius of the hypersphere representing the feasible region. Thus, with $\epsilon << R$ the effort increases exponentially with the dimension.

3.2 Multistart

It is obvious that the global minimum can be easily determined if all local minima are known. This can be achieved by starting a local search technique again and again from starting points chosen at random over M. This technique is called the *Multistart Technique*. The idea is to sample a point from an uniform distribution over M and to apply a local search method with this sample point as starting point in order to obtain a local minimizer. If the termination criterion is not fulfilled the procedure is repeated.

The convergence to the global optimum follows directly from theorem 1. The problem is to provide an appropriate stopping rule. This work has been initiated by [53] and extended by [7], [8] and [6]. The idea of their work is to develop *Bayesian estimates* of the number of local minima not yet found and of the probability that a new local search will result in detecting a new minimum. The optimal stopping rule can be determined by weighing the costs and potential benefits of subsequent searches against each other in a Bayesian sense.

There are two possibilities for a parallel version of Multistart: The first version could use a parallel local search method in the above algorithm. Parallel versions of local search methods have been proposed by [28], [5] and [29]. The advantage is that all the theoretical results about the stopping rules remain valid for the parallel implementation. The efficiency mainly depends on the efficiency of the parallel local search method. A second version uses a master/slave approach with P slaves:

ALGORITHM PMS:
(1) On each slave:
    Repeat
        Choose a sample point from an uniform distribution over M
        Apply a local search method
        Send the local minimizer to the master
    Until there is a stop message from the master
(2) On the master: Whenever a local minimizer is received from a slave
        Update the Bayesian estimates for the stopping rule
        If the termination criterion is not fulfilled goto (2)
        Else terminate the slaves
(3) The best local minimizer is the candidate solution

Here, the theory for the stopping rule has to be taylored for the parallel case, because the outcome of the current local minimizations on the slaves are not known in advance.

Although the multistart technique appears to be promising there is still a lack of sufficient efficiency, because the same local minimum may be found several times. The ideal case would be achieved if a local search is started in every region of attraction only once. This is the aim of the so–called *Clustering methods*.

3.3 Clustering Methods

Clustering methods are used as a tool in global optimization, because they are able to group vectors into clusters such that the vectors within a cluster are as homogeneous as possible, whereas the clusters among each other are as heterogeneous as possible with respect to some similarity measure. The idea is to group the trial points into clusters by a clustering method so that the cluster can be considered as an approximation of the regions of attraction.

3.3.1 Controlled Random Search

Although this algorithm developed by Price [33] [34] [35] has nothing in common with the characterization of clustering methods given above some authors [52] have classified this algorithm as such. The algorithm combines some ideas of Nelder and Mead's [31] Simplex method and a lot of heuristics to construct only *one* cluster. Parallel versions (with modifications of the heuristics) have been proposed by Sutti [49] and Price [35] for MIMD computers. Since there is no theoretical background for this algorithm we omit the description here.

3.3.2 Density Clustering

The sequential algorithm is described in [9] and [38]. The main idea is to build clusters around local minimizers previously detected to prevent the algorithm to start local searches from points 'close' to a local minimum already found.

ALGORITHM DC:

Repeat

(1) Sample N points from an uniform distribution over M
Evaluate and assign them to $S := \{x_1, ..., x_N\}$

(2) Choose $x \in S$ with lowest objective value and start a local search
Build a cluster around the local minimizer by single linkage clustering
Remove clustered points from S
If $S \neq \emptyset$ goto (2)

Until no new local minimizer has been found in the last iteration

The best local minimum found is the candidate solution

Bertocchi [4] proposed a parallel version for a MIMD computer with shared memory. If we 'translate' the algorithm for the message passing architecture the algorithm would run as follows:

ALGORITHM PDC:

On each processor

(1) Run the sequential algorithm

(2) Whenever a new local minimum is found it will be broadcasted to the other processors which will update their set of local minimizers already detected

In order to do so, we have to check at each iteration of step (2) of the sequential code whether a new local optimum has been broadcasted. However, this is not the same algorithm as the sequential one. The reported speedup of about 4.5 on 6 processors is not only due to the parallelization but also to the fact, that more local searches are performed at the beginning.

3.3.3 Multi Level Single Linkage

This algorithm is described in [39] and [13], where some theoretical properties are presented, too.

ALGORITHM MLSL:

(1) k:=1

(2) Choose N points from an uniform distribution over M
Evaluate each point

(3) A sample point is selected as starting point for local minimization, if it has not been used as a starting point previously and if there is no other sample point within the critical distance $r(k)$ with a lower objective value

(4) Perform local minimizations from all selected starting points

(5) k:=k+1
If the termination criterion is not fulfilled goto (2)

(6) The best local minimum found is the candidate solution

The critical distance is given by $r(k) := \pi^{-\frac{1}{2}}\left[\sigma\frac{\log kN}{kN}\Gamma\left(\frac{n}{2}+1\right)\mu(M)\right]^{\frac{1}{n}}$. A parallel version running on a LAN (Local Area Network) can be found in [13].

ALGORITHM PMLSL:
(1) Divide M into P regularly shaped subregions M_i and assign one to each processor
(2) On each processor i do:
    (a) Choose N/P sample points from an uniform distribution over M_i
        Evaluate each point
    (b) Determine starting points as in the sequential version in region M_i
        Perform border resolutions between subregions
    (c) Start local minimizations
(3) If termination criterion is not fulfilled goto (2)
(4) The best local minimum found is the candidate solution

The authors discuss several variants how to perform the starting point selection. The initial approach used in the sequential version to compute the nearest neighbor relations requires $O(N^2)$ steps. For the parallel version they developed an one–stage *divide and conquer* algorithm which breaks down the previous runtime significantly, so that it is recommended to use it in the sequential version, too. The efficiency of the parallel version for the investigated standard test problems (see [17]) depends on the number of samples per iteration.

3.3.4 Topographical Clustering

Both the sequential and the parallel versions are described in [51]. The clustering is realized by constructing and evaluating a graph over the feasible region M.

ALGORITHM TC:
(1) Choose N points from a uniform distribution over M
    Evaluate each point
(2) Compute k nearest neighbors for each point
    Mark all those neighboring points with larger objective value
(4) Start a local search from each point with all reference points marked
(5) The best local minimum found is the candidate solution

Similar to the MLSL algorithm, the main problem is to compute the k–nearest neighbor relations. The approach taken there would improve the runtime of the TC algorithm, too. The parallel version described by [51] uses a ring topology with P workers and one master to generate and evaluate the sample points in parallel and to pipeline each point through the ring in order to calculate the nearest neighbor relations. The selected starting points will be distributed uniformly over all processors so that each processor performs approximately the same number of local searches. The authors report about reasonable speedup with up to 8 workers.

Considering all clustering methods described above the main problem is to design parallel cluster methods for MIMD computers. This would be valuable not only for global optimization algorithms but also in other cases where clustering methods are of use, e.g. pattern recognition or image segmentation. But that problem has not generally been solved up to now.

Another type of clustering is performed by so–called *Evolutionary Algorithms*: They do not cluster the points with respect to the distances of a seedpoint but with respect to their objective values.

4 Evolutionary Algorithms

Evolutionary Algorithms (EAs) can be separated into *Genetic Algorithms* (GAs) developed by [25] at Ann Arbor/Michigan and *Evolution Strategies* (ESs) developed by [37] and [44] [45] at Berlin. Although their major working scheme is the same, there are some significant differences pointed out by [22] [23] in detail. Here, we will propose only Evolution Strategies (for a description of Genetic Algorithms see e.g. [18]).

The starting point has been the so–called two–membered $(1+1)$ ES of [37]:

$$x^{(t+1)} = \begin{cases} x^{(t)} + \xi^{(t)} & \text{, if } f(x^{(t)} + \xi^{(t)}) < f(x^{(t)}) \text{ and } x^{(t)} + \xi^{(t)} \in M \\ x^{(t)} & \text{, otherwise} \quad , \end{cases} \tag{5}$$

where $\xi^{(t)}$ denotes a random vector with some probability measure. This algorithm converges to the global minimum with probability one under some conditions (see [46], [2], [32], [36]):

THEOREM 2:
Let p_t denote the probability that algorithm (5) will reach the region $L_{f^*+\epsilon}$ in step t. If f is bounded from below: $f > -\infty$, $\mu(L_{f^*+\epsilon}) > 0$, the density of ξ is continuous and $\sum\limits_{t=0}^{\infty} p_t = \infty$, then algorithm (5) will reach the region $L_{f^*+\epsilon}$ with probability one with increasing t:

$$Prob\left\{\lim_{t\to\infty} x^{(t)} \in L_{f^*+\epsilon}\right\} = 1 \quad .$$

□

Born [11] received the same result by using a normal distribution as probability measure for random vector ξ with $\sigma \geq \sigma_{min} > 0 : \xi \sim N(0, \sigma I)$. The expected error decreases with $O(t^{-\frac{2}{n}})$ in the quadratic convex case with fixed σ [36]. With an proper adaptation of the σ the decrease of the expected error can be accelerated to $O(\beta^t)$ with $\beta \in (0,1)$. Rechenberg [37] derived the so–called 1/5 – success rule to adapt σ, so that the convergence will be linear in the quadratic convex case. Another adaptation rule has been proposed by [42] but they used an uniform distribution on a hypersphere surface around the current point so that the last condition of theorem 2 is not fulfilled, indicating that this method may *not* converge with probability one [32].

Algorithm (5) is not well suited for parallelization. Schwefel [44] [45] proposed a so–called multi–membered version denoted by $(\mu+\lambda)$ ES or (μ, λ) ES, respectively. For the special case $\mu = 1$ he calculated the convergence rates for some test functions.

In the multi–membered version λ sample points are generated from μ points ($\mu < \lambda$). The best μ points are selected from the newly generated λ sample points for the (μ, λ)–version or from both the old and new sample points for the $(\mu + \lambda)$–version. The selected points then serve as new generation points for the next iteration. An obvious way to parallelize this algorithm is to use a master/slave–approach, where the function evaluations can be done in parallel. One can expect a good processor utilization assuming that the function evaluation is expensive.

In order to avoid this potential bottleneck many authors designed parallel algorithms by using additional principles from organic evolution as a pool of inspiration (see [24] for a survey). All approaches can be separated into two categories which are often called the *migration model* and the *diffusion model*.

4.1 *Migration Model*

The idea is to run the sequential version on each processor and exchange some information between the processors during the search. Imaginating that there are λ individuals (the trial points) forming a population on each processor, the exchange of information can be regarded as migration of individuals if e.g. the best individuals found so far will be exchanged. This parallel version has been implemented by [50], [10], [27], [30] and [40] for ESs as well as for GAs w.r.t. several types of optimization problems. Of course, this is an other algorithm as the sequential one, so that it is not surprising that the 'efficiency' of the parallel version can be above 100 %. As explained in [40] the better algorithmic performance of the parallel version occurs whenever the local optima have parameters in common with the global optimum.

4.2 *Diffusion Model*

The diffusion model represents a more fine grained approach: Each individual is placed on a single processor and selects another individual to share their information in order to generate a new trial point. Using a grid or torus topology each individual has four neighbors, if we are using a von–Neumann neighborhood. This approach seems to be well suited for SIMD machines and has been implemented by [47], [14] as well as [15] on that kind of hardware. Gorges–Schleuter [19] [20] used another topology and MIMD hardware to realize the algorithm. However, the number of individuals is limited by the number of processors, which are not available to such an extent as on a SIMD machine (e. g. connection machine). On the other hand the number of interacting individuals is important for a good algorithmic performance. Moreover, numerical results seem to indicate that algorithms using the diffusion model perform better on average. So it would be interesting for practical reasons to simulate the SIMD algorithm on a MIMD machine. First experimental results are given by [48], who implemented a parallel GA in such a manner on a transputer system with 30 transputers in order to solve combinatorial problems. In order to continue this work we have implemented a parallel ES of this kind and applied it to global optimization problems.

Figure 1 is intended to illustrate how the diffusion 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.

4.2.1 *Reliability*

The algorithm has been tested on standard test problems for global optimization given in [17] and [52] named *Goldstein (n=2), Shekel10 (n=4), Hartman6 (n=6) and Griewank10 (n=10)*, where n denotes the dimension of the problem. A function of Ackley [1] has been generalized for arbitrary dimensions. The algorithm has been run 20 times on each problem with a 10x10–population, or with a 32x32–population for Griewank's function, respectively. The global optimum has been found 13 out of 20 times for Griewank's function within 500 iterations and every time for the others in less than 100 iterations.

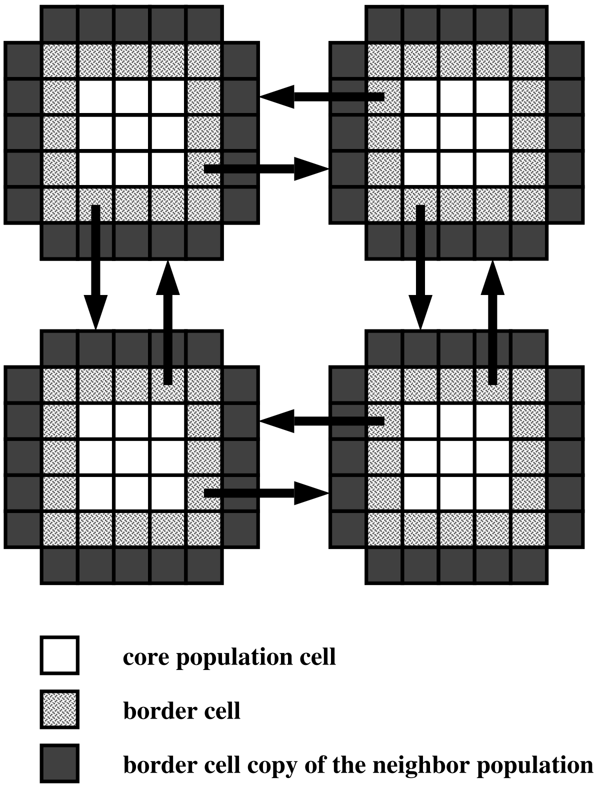

Figure 1: Diffusion model type Evolutionary Algorithm on a MIMD computer

4.2.2 Efficiency

As illustrated by figure 2 the speedup obtained is unsatisfactory for low dimensional problems. Clearly, efficiency improves as soon as the population size increases or function evaluations become more expensive, so that more time must be spent upon one iteration. This is demonstrated by the speedup on Griewank's problem. This kind of algorithm normally runs on SIMD machines with some thousands PE's (processing elements). Thus, one can expect that the current implementation will achieve a reasonable efficiency for large populations on MIMD machines with a moderate number of processors.

5 Conclusion

Considering the few contributions about parallel global optimization summarized in section 3 we can conclude that only little work has been done on this topic. Often there arise algorithmic problems which have nothing to do with global optimization itself but with other areas such as clustering. Moreover, most methods apply a combination of global and local search and as soon as the dimensionality increases much more points must be sampled in the global phase. We expect that the performance of these methods will decrease rapidly in that case.

Such problems can be bypassed by parallel Evolutionary Algorithms. These algorithms do not make any assumption about the objective function and can be used on SIMD as well as on MIMD machines. As demonstrated by [40] Evolution Strategies can be applied to combinatorial problems, too. This generality in application is achieved

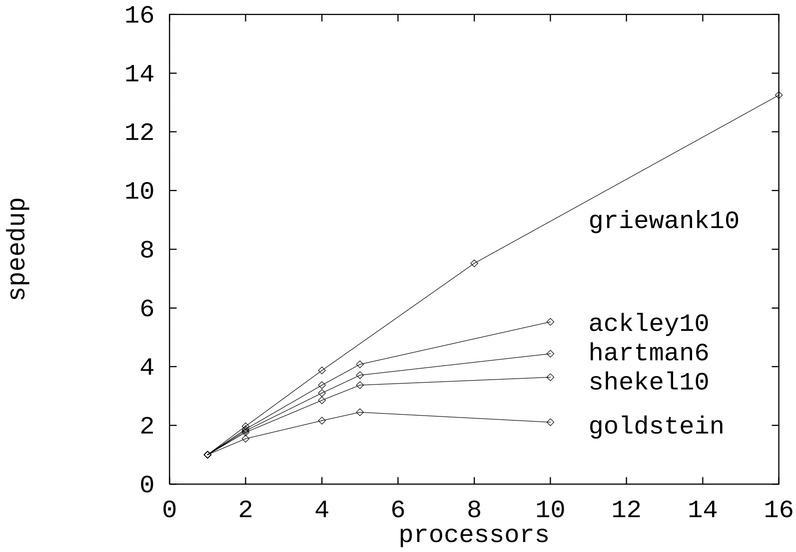

Figure 2: Speedup for some test problems

at the expense of a rather large number of function evaluations compared to other specialized and more or less inherently sequential methods. However, the more processors are available the more Evolutionary Algorithms become competitive to other methods.

6 Acknowledgements

This work has been partially funded under project 107 004 91 by the Research Initiative *Parallel Computing* of Nordrhein–Westfalen, Land of the Federal Republic of Germany.

References

[1] D.H. Ackley. *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers, Boston, 1987.

[2] N. Baba. Convergence of random optimization methods for constrained optimization methods. *Journal of Optimization Theory and Aplications*, 33:451–461, 1981.

[3] R.K. Belew and L.B. Booker, editors. *Proceedings of the fourth Conference of Genetic Algorithms.* Morgan Kaufmann, San Mateo, 1991.

[4] M. Bertocchi. A parallel algorithm for global optimization. *Optimization*, 21(3):379–386, 1990.

[5] D.P. Bertsekas and J.N. Tsitsiklis, editors. *Parallel and Distributed Computation.* Prentice Hall, Englewood Cliffs, 1989.

[6] B. Betro. Bayesian methods in global optimization. *Journal of Global Optimization*, 1(1):1–14, 1991.

[7] B. Betro and F. Schoen. Sequental stopping rules for the multistart algorithm in global optimization. *Mathematical Programming*, 38:271–286, 1987.

[8] C.G.E. Boender and A.H.G. Rinnooy Kan. Bayesian stopping rules for multistart global optimization methods. *Mathematical Programming*, 37:59–80, 1987.

[9] C.G.E. Boender, A.H.G. Rinnooy Kan, G.T. Timmer, and L. Stougie. A stochastic method for global optimization. *Mathematical Programming*, 22:125–140, 1982.

[10] A. Bormann. Parallelisierungsmöglichkeiten für direkte Optimierungsverfahren auf Transputersystemen. Master's thesis, University of Dortmund, Chair of System Analysis, 1989.

[11] Joachim Born. *Evolutionsstrategien zur numerischen Lösung von Adaptationsaufgaben.* Dissertation A, Humboldt-Universität, Berlin, GDR, 1978.

[12] S.H. Brooks. A discussion of random methods for seeking maxima. *Operations Research*, 6:244–251, 1958.

[13] R.H. Byrd, C.L. Dert, A.H.G. Rinnooy Kan, and R.B Schnabel. Concurrent stochastic methods for global optimization. *Mathematical Programming*, 46(1):1–29, 1990.

[14] R.J. Collins and D.R. Jefferson. Selection in massively parallel genetic algorithms. In Belew and Booker [3], pages 249–256.

[15] Y. Davidor. A naturally occuring niche and species phenomenon: The model and first results. In Belew and Booker [3], pages 257–263.

[16] L. Devroye. Progressive global random search of continuous functions. *Mathematical Programming*, 15:330–342, 1978.

[17] L.C.W. Dixon and G.P. Szegö, editors. *Towards Global Optimization 2.* North Holland, Amsterdam, 1978.

[18] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, Reading/Mass., 1989.

[19] M. Gorges-Schleuter. ASPARAGOS: an asynchronous parallel genetic optimization strategy. In Schaffer [41], pages 422–427.

[20] M. Gorges-Schleuter. Explicit parallelism of genetic algorithms through population structures. In Schwefel and Männer [43], pages 150–159.

[21] M. Grauer and D. B. Pressmar, editors. *Applied Parallel and Distributed Optimization.* Lecture Notes in Mathematical Systems and Economics. Springer, 1991.

[22] F. Hoffmeister and Th. Bäck. Genetic Algorithms and Evolution Strategies: Similarities and Differences. Technical Report "Grüne Reihe" No. 365, Department of Computer Science, University of Dortmund, 1990.

[23] F. Hoffmeister and Th. Bäck. Genetic Algorithms and Evolution Strategies: Similarities and Differences. In Schwefel and Männer [43], pages 455–469.

[24] Frank Hoffmeister. Scalable parallelism by evolutionary algorithms. In Grauer and Pressmar [21].

[25] John H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, 1975.

[26] R. Horst and H. Tuy. *Global Optimization - Deterministic Approaches.* Springer, Berlin and Heidelberg, 1990.

[27] R. Lohmann. Application of evolution strategies in parallel populations. In Schwefel and Männer [43], pages 198–208.

[28] F.A. Lootsma and K.M. Ragsdell. State-of-the-art in parallel nonlinear optimization. *Parallel Computing*, 6:133–155, 1988.

[29] X. Miao, P.L. Luh, and S.-C. Chang. A parallel conjugate direction method for unconstrained optimization. *International Journal of Modelling and Simulation*, 10(1):6–12, 1990.

[30] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. In Belew and Booker [3], pages 271–278.

[31] J.A. Nelder and R. Mead. A simplex method for function minimization. *Comp. Journal*, 7:308–313, 1965.

[32] J. Pinter. Convergence properties of stochastic optimization procedures. *Math. Operat. Stat. , Ser. Optimization*, 15:405–427, 1984.

[33] W.L. Price. A controlled random search procedure for global optimization. In Dixon and Szegö [17], pages 71–84.

[34] W.L. Price. Global optimization by controlled random search. *Journal of Optimization Theory and Aplications*, 40(3):333–348, 1983.

[35] W.L. Price. Global Optimization Algorithms for a CAD Workstation. *Journal of Optimization Theory and Aplications*, 55(1):133–146, 1987.

[36] G. Rappl. *Konvergenzraten von Random Search Verfahren zur globalen Optimierung.* PhD thesis, HSBw München, Germany, 1984.

[37] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann–Holzboog Verlag, Stuttgart, 1973.

[38] A.H.G. Rinnooy Kan and G.T. Timmer. Stochastic global optimization methods, Part I: Clustering methods. *Mathematical Programming*, 39:27–56, 1987.

[39] A.H.G. Rinnooy Kan and G.T. Timmer. Stochastic global optimization methods, Part II: Multi level methods. *Mathematical Programming*, 39:57–78, 1987.

[40] G. Rudolph. Global optimization by means of distributed evolution strategies. In Schwefel and Männer [43], pages 209–213.

[41] J.D. Schaffer, editor. *Genetic Algorithms, Proceedings of the 3rd International Conference on Genetic Algorithms.* Morgan Kaufman, San Mateo, 1989.

[42] M.A. Schumer and K. Steiglitz. Adaptive step size random search. *IEEE Transactions on Automatic Control*, 13:270–276, 1968.

[43] H.-P. Schwefel and R. Männer, editors. *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*. Springer, Berlin and Heidelberg, 1991.

[44] Hans-Paul Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie.* Interdisciplinary systems research; 26. Birkhäuser, Basel, 1977.

[45] Hans-Paul Schwefel. *Numerical Optimization of Computer Models.* Wiley, Chichester, 1981.

[46] F.J. Solis and R.J.-B. Wets. Minimization by random search techniques. *Math. Operations Research*, 6:19–30, 1981.

[47] P. Spiessens and B. Manderick. A massively parallel genetic algorithm: Implementation and first analysis. In Belew and Booker [3], pages 279–286.

[48] J. Sprave. Parallelisierung Genetischer Algorithmen zur Suche und Optimierung. Master's thesis, University of Dortmund, Chair of System Analysis, December 1990.

[49] C. Sutti. Local and global optimization by parallel algorithms for MIMD machines. *Annals of Operations Research*, 1:151–164, 1984.

[50] R. Tanese. Distributed genetic algorithms. In Schaffer [41], pages 434–439.

[51] A. Törn and S. Viitanen. Topographical global optimization. *Journal of Global Optimization*, to appear.

[52] A. Törn and A. Zilinskas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer, Berlin and Heidelberg, 1989.

[53] R. Zielinski. A statistical estimate of the structure of multiextremal problems. *Mathematical Programming*, 21:348–356, 1981.