Multi-Objective Particle Swarm Optimization on Computer Grids

Sanaz Mostaghim, Jürgen Branke and Hartmut Schmeck

Institute AIFB, University of Karlsruhe, Germany
{mostaghim|branke|schmeck}@aifb.uni-karlsruhe.de

Abstract. In recent years, a number of authors have successfully extended particle swarm optimization to problem domains with multiple objectives. This paper addresses the issue of parallelizing multi-objective particle swarms. We propose and empirically compare two parallel versions which differ in the way they divide the swarm into subswarms that can be processed independently on different processors. One of the variants works asynchronously and is thus particularly suitable for heterogeneous computer clusters as occurring e.g. in modern grid computing platforms.

1 Introduction

Particle Swarm Optimization (PSO) is now established as an efficient optimization algorithm for static functions in a variety of contexts [21]. PSO is a population based technique, similar in some respects to evolutionary algorithms, except that potential solutions (particles) move, rather than evolve, through the search space. The rules, or particle dynamics, which govern this movement, are inspired by models of swarming and flocking [15]. Each particle has a position and a velocity, and experiences linear spring-like attractions towards two attractors:

1. The best position attained by that particle so far (local attractor), and
2. The best position found by the swarm as a whole (global attractor),

where best is in relation to evaluation of an objective function at that position. The global attractor therefore enables information sharing between particles, whilst the local attractors serve as individual particle memories.

The optimization process is iterative. In each iteration, the acceleration vectors of all the particles are calculated based on the positions of the corresponding attractors. Then, this acceleration is added to the velocity vector, the updated velocity is constricted so that the particles progressively slow down, and this new velocity is used to move the individual from the current to the new position. The details of our implementation are provided in Section 3.

Due to the success of particle swarm optimization (PSO) in single objective optimization, in recent years, more and more attempts have been made to extend PSO to the domain of multi-objective problems, see e.g. [18, 3]. The main challenge in multi-objective particle swarm optimization (MOPSO) is to select the global and local attractors such that the swarm is guided towards the Pareto optimal front and maintains sufficient diversity. Our paper simply adopts one of the proposed strategies, namely the sigma-method [18]. The paper's focus is on parallelization strategies, and is largely independent from the multi-objectivization technique used.

The motivation for parallelization is that for many practical optimization problems, evaluating a single solution already requires a significant computational effort, e.g. if the evaluation involves a computationally demanding fluid dynamics simulation. On the other hand PSO and also other nature-inspired optimization techniques like evolutionary computation or simulated annealing usually require the evaluation of a large number of solutions before producing good results. One way to resolve this predicament is to employ parallel processing. While only few people have access to a dedicated parallel computer, recently, it also became possible to distribute an algorithm over any bunch of networked computers, using a paradigm called "grid computing". Grid Computing enables the sharing, selection, and aggregation of a wide variety of geographically distributed computational resources (such as supercomputers, compute clusters, storage systems, etc.) and presents them as a single, unified resource for solving large-scale and data intensive computing applications [13]. This idea is analogous to the electric power network (grid) where power generators are distributed, but the users are able to access electric power without bothering about the source of energy and its location. As such, grid computing promises to make high performance computing available to almost everyone. On the other hand, it makes parallelization more challenging, as one can no longer assume a homogeneous set of processors. Also, grid computers usually do not allow the direct communication between processors.

In this paper, we propose two parallel variants of MOPSO. In both cases, the basic idea is to repeatedly divide the population into a number of subswarms which can be processed in parallel. The subswarms run for a limited number of iterations and then return their result (i.e. all non-dominated solutions found) to a central server. In the next iteration, they are provided with a "guide", a non-dominated particle, and re-start search with this guide and all other particles re-initialized within the local search-space neighborhood of the guide. The two proposed variants differ in the way they select the guides. The cluster-based subswarm MOPSO (C-MOPSO) waits for all subswarms to return their results, and then performs a clustering step to identify a number of well-distributed guides along the non-dominated front. The hypervolume-based subswarm MOPSO (H-MOPSO) selects guides one at a time based on their marginal hypervolume, i.e. the hypervolume covered by a particle that is not covered by any other particle. Note that the latter version works asynchronously, which makes it particularly suitable for heterogeneous computer clusters such as the grid.

The paper is structured as follows. In the next section, we briefly survey related work. Our parallel PSO variants are presented in Section 3. Some preliminary empirical results on homogeneous as well as heterogeneous computer clusters are provided in Section 4. The paper concludes with a summary and several ideas for future work.

2 Related Work

In PSO, parallelization has been largely neglected so far. We are aware of only one paper [9] on parallel single-objective PSO, and no paper on parallel MOPSO. Thus, in the following, we focus on parallel multi-objective evolutionary algorithms (MOEAs), where parallelization has been studied extensively. The parallel MOEA approaches can be grouped into three categories:

1. Master-slave: Here, a single processor maintains control over selection, and uses the other processors only for crossover, mutation and evaluation of individuals. It is useful only for few processors or very large evaluation times, as otherwise the strong communication overhead outweighs the benefit of parallelization.
2. Island model: In this model, every processor runs an independent EA, using a separate sub-population. In regular intervals, *migration* takes place: The processors cooperate by exchanging good individuals. The island model is particularly suitable for computer clusters, as communication is limited, and thus most related to our work.
3. Diffusion model: Here, the individuals are spatially arranged, and mate with other individuals from their local neighborhood. When parallelized, there is a lot of interprocessor communication (every individual has to communicate with its neighbors in every iteration), but the communication is only local. Thus this paradigm is particularly suitable for massively parallel computers with a fast local intercommunication network.

A detailed discussion of parallel evolutionary algorithms is out of the scope of this paper. The interested reader is referred to e.g. [22, 8, 2]. One of the few papers considering heterogeneous processors is [6], which assumes an island model and examines different migration policies and neighborhood structures.

Also, there are quite a few papers on parallelizing multi-objective evolutionary algorithms. Most of these methods are based on the island model, and attempt to divide the objective space into several regions which are then assigned to different processors. Deb [12] uses ideas from the Guided-MOEA [7] to focus the different subpopulations on different trade-off ranges between the objectives. Branke et al. [5] proposed to explicitly divide up the objective space into "cones". Streichert et al. [23] study a clustering method and apply a divide and conquer method. These approaches assume communication between processors, and tested only a rather small number of parallel processors (up to six). Heterogeneity of the processors has not been considered in either of these papers.

Subswarms in MOPSO have already been used in [19] in order to obtain a better covering of the front, and in [14] to maintain the spread of solutions.

Optimization in Grid Computing is an established field, e.g., Nimrod/O [1]. In Nimrod/O, several heuristics are provided like Simulated Annealing, Evolutionary Programming and Genetic Algorithms which are being used to solve many real-world applications. However, they involve only single objective optimization techniques, and to our knowledge, no PSO has been proposed yet.

3 Parallel Multi-Objective Particle Swarm Optimization

3.1 Multi-Objective Particle Swarm Optimization

A MOPSO starts with a set of uniformly distributed random initial particles defined in the search space S. A set of M particles are considered as a population P_t at the generation t. Each particle i has a position defined by $\boldsymbol{x}^i = (x_1^i, x_2^i, \cdots, x_n^i)$ and a velocity defined by $\boldsymbol{v}^i = (v_1^i, v_2^i, \cdots, v_n^i)$ in the search space S. Beside the population, another set (called Archive) A_t can be defined in order to store the obtained non-dominated solutions. Due to the presence of an archive, good solutions are preserved during generations. The particles are evaluated and the non-dominated solutions are added to the archive in every generation, while dominated solutions are pruned. In the next step, the particles are moved to a new positions in the space. The velocity and position of each particle i is updated as below:

$$\begin{aligned} v_{j,t+1}^i &= wv_{j,t}^i + c_1R_1(p_{j,t}^i - x_{j,t}^i) + c_2R_2(p_{j,t}^{i,g} - x_{j,t}^i) \\ x_{j,t+1}^i &= x_{j,t}^i + v_{j,t+1}^i \end{aligned} \tag{1}$$

where $j = 1, \ldots, n$, $i = 1, \ldots, M$, c_1 and c_2 are two positive constants, R_1 and R_2 are random values in the range $[0, 1]$ and w is the inertia weight which is employed to control the impact of the previous history of velocities.

$\boldsymbol{p}_t^{i,g}$ is the position of the **global best particle** in the population, which guides the particles to move towards the optimum. The important part in MOPSO is to determine the best global particle $\boldsymbol{p}_t^{i,g}$ for each particle i of the population. In single-objective PSO, the global best particle is determined easily by selecting the particle that has the best position. But in MOPSO, $\boldsymbol{p}_t^{i,g}$ must be selected from the updated set of non-dominated solutions stored in the archive. We use the sigma-method for this purpose [18]. $\boldsymbol{p}_t^i$ is the best position that particle i could find so far and keeps the non-dominated (best) position of the particle by comparing the new position $\boldsymbol{x}_{t+1}^i$ in the objective space with $\boldsymbol{p}_t^i$ ($\boldsymbol{p}_t^i$ is the last non-dominated (best) position of the particle i). The steps of a MOPSO are iteratively repeated until a termination criterion is met.

3.2 Parallelization

Here, we propose two parallelization techniques. We assume that there is a central server maintaining the archive, distributing work to different processors and collecting the results. This assumption is in line with the usual grid architecture. The parallel processors are assigned sub-swarms, and run independent MOPSOs based on these sub-swarms for a prespecified number of iterations. All found non-dominated solutions are returned as result to the central processor.

Intuitively, it makes sense to have different subswarms each in different regions of the non-dominated front. This is also the underlying assumption of the parallel MOEAs discussed in Section 2. In our case, focus on different areas of the non-dominated front is achieved by assigning each subswarm a "guide". A guide is a non-dominated solution from the archive. The subswarm is then randomly initialized in a search-space neighborhood of the guide, and the guide itself becomes part of the swarm. Because of the guide and the local initialization, the search of the sub-swarm is likely to focus on a region of the non-dominated front close to the location of the guide.

The two parallelization approaches differ in the way they select guides from the archive, which will be explained in more detail in the following subsections.

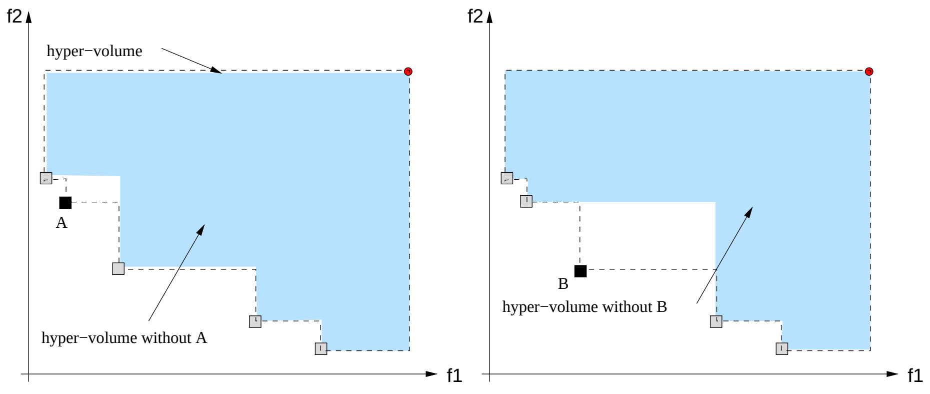

Fig. 1. Selection of the global guide. Point B has a larger impact on the hyper-volume and therefore must be selected first. (□: solutions, ○: reference point)

Cluster-based subswarm MOPSO The idea of the cluster-based subswarm MOPSO (C-MOPSO) is to pick a set of guides which represents the current non-dominated front as well as possible. This is achieved by performing a clustering operation on the archive, with the goal to find N cluster representatives if N is the number of processors available.

Note that this assumes synchronization after every iteration: the central processor performs the clustering, sends out the different guides to the different processors, waits for them to return their results, updates the archive, and then starts the next iteration.

Hypervolume-based subswarm MOPSO In the hypervolume-based subswarm MOPSO (H-MOPSO), guides are selected one by one according to their marginal hypervolume. The hypervolume is the area dominated by all solutions stored in the archive [25]. The *marginal* hypervolume of a particle is the area dominated by the particle that is not dominated by any other particle. As guide, the particle from the archive is selected which has not been selected before and which has the largest marginal hypervolume. Only if all archive solutions have been used as guides before, they are allowed to be re-used. Figure 1 illustrates this. Point A has a smaller contribution to the whole hyper-volume value than Point B. Therefore, Point B must be selected first.

Note that this guide selection process is asynchronous. Whenever a processor returns its results, they can be immediately integrated into the archive, a new guide can be selected and the processor can be assigned a subswarm based on a new guide. This makes the approach particularly suitable for heterogeneous computer clusters such as grids, where very fast processors are used along with rather slow ones. It is not necessary to wait for the slowest processor to return its results before spawning the next set of subswarms as is the case with C-MOPSO.

For both variants, the region used to initialize a subswarm around the guide can be chosen in different ways. In preliminary experiments, we tested the following three approaches:

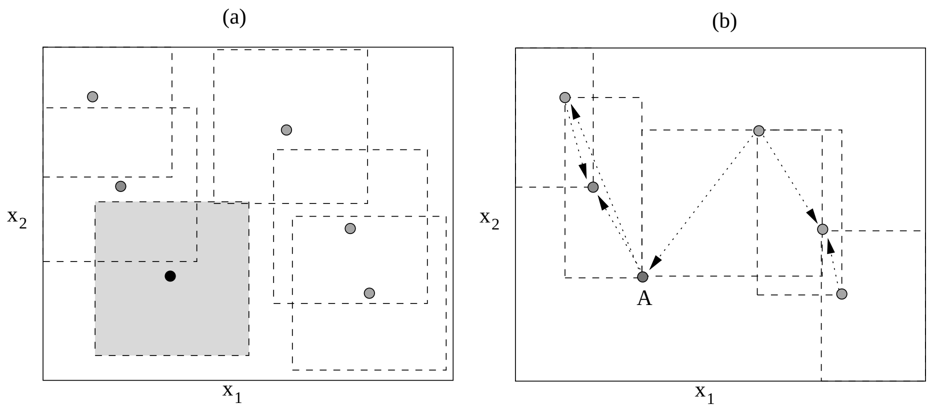

Fig. 2. (a) A fixed sized area around each guide is selected to be the search space of the corresponding MOPSO for that guide. (b) The area in search space between surrounding neighbors in the objective space is selected to be the search space for the MOPSO. The neighbors are shown with arrows. For point A, both neighbors are not surrounding it in the search space, therefore the maximum and minimum coordinates of A and its neighbors are selected to define the search space.

– A subswarm can use the whole search space of the problem. This is not efficient, if the search space is large, as it takes a while until the subswarm converges to the interesting area around the current non-dominated front.
– A fixed, but smaller search area can be defined around the input guide. We use $\pm 10\%$ of the whole search space range in each dimension (capped by the search space boundaries, of course).
– The area between the m neighbors (in the m-objective space) of the guide which the guide is in between, is selected as the search space of the subswarm. The neighbors are non-dominated solutions selected from the archive. The selection of neighbors is based on the distances in the objective space.

Figure 2 shows the last two scenarios in an example with 2 parameters. $\circ$ denotes the non-dominated solutions stored in Archive.dat. In (a), a selected area around each guide is set to be the search space for the corresponding MOPSO. Depending on the defined size, the main search space can be easily covered by all of them. In (b) the area between the two neighbors is selected to be the search space. Since the neighbors of the guide are selected in the objective space, it can happen that the area defined by their positions in the search space does not contain the guide, e.g., point A. Therefore, the maximums and minimum values of their corresponding decision vectors are set to be the high and low ranges for the defined search space. In our preliminary experiments, the second range definition performed best and will be used for the remainder of this study.

4 Experiments

In this section, we test C-MOPSO and H-MOPSO on two scenarios, one consisting of homogeneous processors, one consisting of heterogeneous processors.

Parameter Setting In both approaches, for a subswarm, a MOPSO method as proposed in [18] is used with 20 particles and an internal archive size of 20 and is run for 20 generations. We select standard values for turbulence factor and inertia weights as 0.1 and 0.4.

The hyper-volume values in H-MOPSO are being computed by building a grid between a reference point and the origin in the objective space. The grid points dominated by the solutions are counted as the hyper-volume. High values of hyper-volume show a high quality in terms of the diversity and convergence of the obtained solutions.

The selected test function from literature is a 2-objective test containing 10 parameters [10]:

$$\begin{aligned} f_1(\boldsymbol{x}) &= 1 - exp(-\sum_i (x_i - \frac{1}{\sqrt{n}})^2) \\ f_2(\boldsymbol{x}) &= 1 - exp(-\sum_i (x_i + \frac{1}{\sqrt{n}})^2) \end{aligned} \tag{2}$$

where $x_i \in [-4,\ 4]$.

We examine both of the methodologies in homogenous and heterogeneous environments. In order to simulate the heterogeneous environment, we consider 20 processors as shown in Figure 3. Three processors (1-3) are fast and can finish their optimization tasks (i.e. running a subswarm for 20 iterations) in time T_1, Four processors (4-7) are slower and require time $T_2 = 2T_1$ for this task. The rest of the processors (8-20) are even slower, requiring $T_3 = 3T_1$. In Figure 3, a total time of $6T_1$ is depicted. After this time, all processors are synchronized again, and we call this a “cycle”. We run our tests for 6 such cycles (i.e. the slowest processors can process 12 subswarms in this time, the fastest processors can run 36 subswarms).

The homogenous environment uses 20 processors with the same speed. In this case, we run the program until each processor has processed 12 subswarms. Note that for C-MOPSO, as it always waits for the slowest processor before spawning the next set of subswarms, there is no difference between the heterogeneous and the homogeneous environment. H-MOPSO, on the other hand, can run more subswarms on the faster processors and thus can make better use of the available processing power in a heterogeneous environment.

Evaluations Table 1 shows the average hyper-volume measures obtained for both methods in heterogeneous and homogenous environments (for C-MOPSO, heterogeneous and homogeneous are identical as explained above). Comparing C-MOPSO and H-MOPSO on the homogeneous environment, C-MOPSO performs better. Because both use exactly the same processing power, this difference has to be due to the strategy

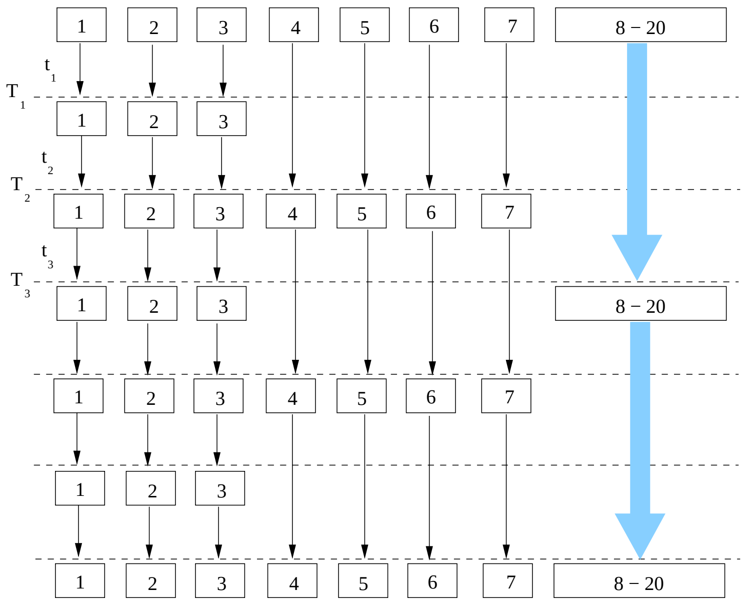

Fig. 3. 20 heterogeneous processors are illustrated. t_1, t_2 and t_3 are equal.

of selecting the guide. We conclude that selecting guides according to hypervolume is better than trying to obtain an even distribution along the front by using clustering.

In the heterogeneous environment, H-MOPSO works even better, because it can make better use of the processing power available. Figure 4 shows the results in Table 1 over iterations graphically.

Altogether, H-MOPSO is able to find more non-dominated solutions with higher convergence and diversity rates than C-MOPSO.

5 Conclusion and Future Work

In this paper, we proposed two new parallel methodologies of Multi-Objective Particle Swarm Optimization (MOPSO). To our knowledge, these approaches are the first parallel MOPSO methods. One of the methods is particularly designed to also work in parallel environments containing an arbitrary amount of heterogeneous processors as is a typical setting for modern grid-computing environments.

The basic idea behind these methods is to divide the population into subswarms which can be processed in parallel. Cluster-based MOPSO (C-MOPSO) is designed to work on a fixed number of processors where hypervolume-based MOPSO (H-MOPSO) is flexible to work on a heterogeneous set of processors. As the results show, H-MOPSO outperformed C-MOPSO also in homogeneous environments, showing that the way to generate subswarms according to marginal hypervolume performs better than trying to distribute subswarms evenly along the current non-dominated front by clustering.

Table 1. Average values and std. error of Hyper-volume values computed for Heterogeneous H-MOPSO (Hetero.H-MOPSO), C-MOPSO and Homogenous H-Mopso (Homo. H-MOPSO) methods after every cycle.

cycle	Hetero. H-MOPSO	stderr	C-MOPSO	stderr	Homo.H-MOPSO	stderr
1	13015	63.56	11969	44.98	12813	40.82
2	13186	58.515	12610	41.227	12950	42.73
3	13269	46.40	12758	35.79	13142	21.44
4	13322	40.93	12782	22.87	13239	9.391
5	13360	31.31	12863	35.577	13261	9.168
6	13639	13.57	12885	32.99	13273	8.9106

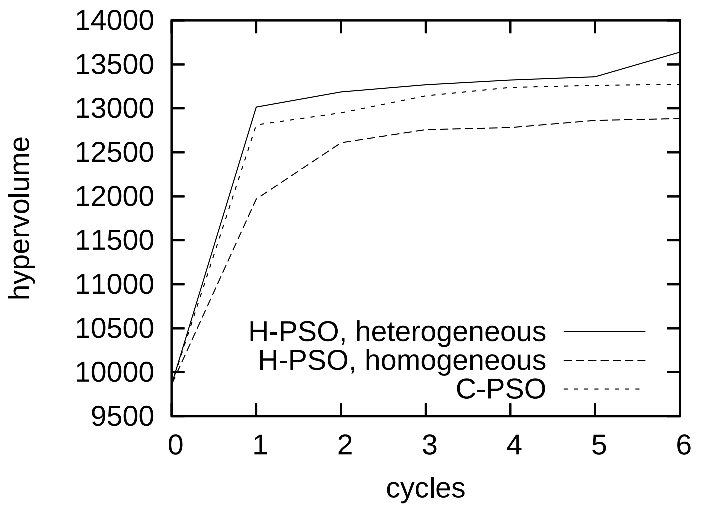

Fig. 4. Hypervolume values computed for homogenous and heterogeneous H-MOPSO and C-MOPSO methods over 10 iterations(the shown cycles are recorded for two iterations).

In future, we will test the approaches on a larger set of test problems and parallel environments. We will also test a number of possible improvements to the current approach, e.g. by allowing solutions with high marginal hypervolume to be re-selected after some time, even if some other non-dominated solutions with low marginal hypervolume have not yet been selected. Finally, we will test the above approached on a real grid system called JOSCHKA (Job Scheduling Karlsruhe) [4] with a real world application.

References

1. D. Abramson, A. Lewis, and T. Peachy. Nimrod/o: A tool for automatic design optimization. In *The 4th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2000)*, 2000.

2. E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–461, 2002.
3. J. E. Alvarez-Benitez, R. M. Everson, and J. E. Fieldsend. A MOPSO algorithm based exclusively on pareto dominance concepts. In C. Coello-Coello et al., editors, *Evolutionary Multi-Criterion Optimization*, volume 3410, pages 459–73. Springer, 2005.
4. Matthias Bonn, Frederic Toussaint, and Hartmut Schmeck. Joschka: Job-scheduling in heterogenen systemen. In Erik Maehle, editor, *PARS Mitteilungen 2005*, pages 99–106. 20. PARS Workshop, Gesellschaft für Informatik, JUN 2005.
5. J. Branke, H. Schmeck, K. Deb, and M. Reddy. Parallelizing multi-objective evolutionary algorithms: cone separation. In *IEEE Proceedings, World Congress on Computational Intelligence (CEC'04)*, pages 1952–1957, Portland, USA, June 2004.
6. Jürgen Branke, Andreas Kamper, and Hartmut Schmeck. Distribution of evolutionary algorithms in heterogeneous networks. In *Genetic and Evolutionary Computation Conference*, volume 3102 of *LNCS*, pages 923–934. Springer, 2004.
7. Jürgen Branke, Thomas Kaußler, and Hartmut Schmeck. Guidance in evolutionary multi-objective optimization. *Advances in Engineering Software*, 32:499–507, 2001.
8. E. Cantu-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer, 2000.
9. J.-F. Chang, S.-C. Chu, F. F. Roddick, and J.-S. Pan. A parallel particle swarm optimization algorithm with communication strategies. *Journal of Information Science and Engineering*, 21(4):809–818, 2005.
10. C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, 2002.
11. K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization test problems. In *Congress on Evolutionary Computation*, pages 825–830. IEEE, 2002.
12. K. Deb, P. Zope, and A. Jain. Distributed computing of pareto-optimal solutions with evolutionary algorithms. In *Proceedings of Second International Conference on Evolutionary Multi-Criterion Optimization (EMO03)*, pages 534–549, 2003.
13. Anthony Hey, Geoffrey Fox, and Fran Berman. *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley and Sons, 2003.
14. S. Janson and D. Merkle. A new multi-objective particle swarm optimization algorithm using clustering applied to automated docking. In *Hybrid Metaheuristics*, pages 128–141, Springer-Verlag, 2005.
15. J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.
16. M. Khabzaoui, C. Dhaenens, and E.-G. Talbi. Parallel genetic algorithms for multi-objective rule mining. In *Metaheuristic International Conference (MIC'2005)*, pages 571–576, 2005.
17. Jörn Mehnen, Thomas Michelitsch, Karlheinz Schmitt, and Torsten Kohlen. pMOHypEA: Parallel evolutionary multiobjective optimization using hypergraphs. Interner Bericht des Sonderforschungsbereichs 531 *Computational Intelligence* CI–189/04, Universität Dortmund, Dezember 2004.
18. S. Mostaghim and J. Teich. Strategies for finding good local guides in multi-objective particle swarm optimization. In *IEEE Swarm Intelligence Symposium*, pages 26–33, Indianapolis, USA, 2003.
19. S. Mostaghim and J. Teich. Covering pareto-optimal fronts by subswarms in multi-objective particle swarm optimization. In *IEEE Proceedings, World Congress on Computational Intelligence (CEC'04)*, pages 1404–1411, Portland, USA, June 2004.
20. T. Okabe, Y. Jin, M. Olhofer, and B. Sendhoff. On test functions for evolutionary multi-objective optimization. In *Parallel Problem Solving from Nature*, volume 3242 of *LNCS*, pages 792–802. Springer, 2004.
21. K.E. Parsopoulos and M.N. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, 1(2-3):235–306, 2002.

22. H. Schmeck, U. Kohlmorgen, and J. Branke. Parallel implementations of evolutionary algorithms. In A. Zomaya, F. Ercal, and S. Olariu, editors, *Solutions to Parallel and Distributed Computing Problems*, pages 47–66. Wiley, 2001.
23. Felix Streichert, Holger Ulmer, and Andreas Zell. Parallelization of multi-objective evolutionary algorithms using clustering algorithms. In *Proceedings of Third International Conference on Evolutionary Multi-Criterion Optimization (EMO05)*, pages 92–107, 2005.
24. D. A. Van Veldhuizen, J.B. Zydallis, and G. B. Lamont. Considerations in engineering parallel multiobjective evolutionary algorithms. In *IEEE Transactions on Evolutionary Computation, Vol. 7, No. 2*, pages 144–173, April 2003.
25. E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. Shaker, 1999.
26. Eckart Zitzler and Simon Künzli. Indicator-based selection in multiobjective search. In *8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*, pages 832–842, Birmingham, UK, September 2004. Springer-Verlag, Berlin, Germany.