Multi-Objective Particle Swarm Optimization on Computer Grids

Sanaz Mostaghim, Jürgen Branke, and Hartmut Schmeck
Institute AIFB, University of Karlsruhe
Karlsruhe, Germany
mostaghim@aifb.uni-karlsruhe.de, branke@aifb.uni-karlsruhe.de,
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.

Categories and Subject Descriptors: I.2.11 [Computing Methodologies]: Distributed Artificial Intelligence

General Terms: Algorithms.

Keywords: Parallel Optimization, Multi-objective Optimization, Particle Swarm Optimization, Grid Computing.

1. INTRODUCTION

Particle Swarm Optimization (PSO) is now established as an efficient optimization algorithm for static functions in a variety of contexts [24]. 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 [18]. 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. [21, 4]. 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 [21]. 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 [16]. 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 aimed at using a small ($<$50) number of computers in parallel, as they are available e.g. in a company grid network. For both algorithms, the basic idea is to repeatedly divide the population into a number of subswarms which can be processed in parallel. The subswarms

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GECCO'07, July 7–11, 2007, London, England, United Kingdom.
Copyright 2007 ACM 978-1-59593-697-4/07/0007 ...$5.00.

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 (CMOPSO) 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 (HMOPSO) 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 the first MOPSO (to our knowledge) 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. An empirical analysis and comparison with straightforward alternatives on homogeneous as well as heterogeneous computer clusters is 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 [12] on parallel single-objective PSO, and one paper on parallel MOPSO [23], none of them considering heterogeneous computing resources. Thus, in the following, we first discuss parallel multi-objective evolutionary algorithms (MOEAs), where parallelization has been studied extensively. Parallel evolutionary algorithms (EAs) can be grouped into three categories:

1. Farming model: 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 inter-processor 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 EAs is out of the scope of this paper. The interested reader is referred to e.g. [26, 11, 3]. 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 several 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 [15] uses ideas from the Guided-MOEA [7] to focus the different subpopulations on different trade-off ranges between the objectives. Branke et al. [9] proposed to explicitly divide up the objective space into "cones". Streichert et al. [27] refines this idea, integrating a clustering method. In the only parallel MOPSO paper [23] we are aware of, several single objective PSO algorithms are run in parallel, each optimizing a different objective. For the velocity update, however, as global best, one of the global bests from all single objective swarms are chosen at random. All of above approaches assume communication between processors, and tested only a rather small number of parallel processors (2-16). Heterogeneity of the processors has not been considered in either of these papers.

Subswarms in MOPSO have already been used in [22] in order to obtain a better covering of the front, and in [17] to maintain the spread of solutions.

Optimization in Grid Computing is an established field, see 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. In [10], a general framework for parallel and distributed metaheuristics has been proposed, another such framework is implemented in [2]. In [20], such a Grid framework is used to run several PAES-style [19] multi-objective EAs independently in parallel.

3. PARALLEL MULTI-OBJECTIVE PARTICLE SWARM OPTIMIZATION

In this section, we briefly explain Multi-Objective Particle Swarm Optimization (MOPSO) methods. Then the parallel MOPSO is being introduced.

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 in generation t. Each particle i has a position defined by $\vec{x}^i = (x_1^i, x_2^i, \cdots, x_n^i)$ and a velocity defined by $\vec{v}^i = (v_1^i, v_2^i, \cdots, v_n^i)$ in the search space S. Beside the population, another set A_t (called Archive) can be defined in order to store the obtained non-dominated solutions. Due to the presence of an archive, good solutions are preserved over 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 follows.

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

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 variables uniformly distributed in $[0, 1]$ and w is the inertia weight which is employed to control the impact of the previous history of velocities.

$\vec{p}_t^{\,i,g}$ and $\vec{p}_t^{\,i}$ are the positions of the global and local attractors, respectively. They guide the particles towards promising regions of the search space. In single-objective PSO, the global attractor is simply the best solution found by the swarm so far, while the local attractor is the best solution found by the particle itself. These solutions are uniquely determined because the solutions can be fully ordered according to objective function values.

But in MOPSO, in the presence of multiple objectives, an analogous choice of global and local attractor is not straightforward. The global attractor $\vec{p}_t^{\,i,g}$ might be any solution from the updated

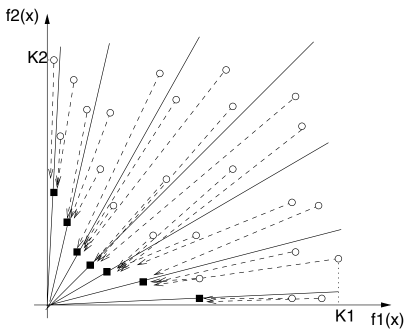


Figure 1: Selecting the global attractor using sigma method. o and ■ denote the population members and non-dominated solutions respectively.

set of non-dominated solutions stored in the archive, and several papers have proposed different methods to make a definite choice, see e.g. [13, 4]. For a recent survey on MOPSO, see [25]. We use the sigma-method for selecting $\vec{p}_t^{\,i,g}$ [21]. As global attractor for each particle, it chooses the archive member which is closest to the line connecting the particle's current position to the origin. This idea is illustrated in Figure 1 for 2 dimensions.

Different strategies to select the local attractor in the presence of multiple objectives have been examined in [8]. Here, we simply keep the non-dominated (best) position of the particle by comparing the new position $\vec{x}_{t+1}^{\,i}$ in the objective space with the current local attractor $\vec{p}_t^{\,i}$, and setting the local attractor to the new position if and only if the new position is non-dominated with respect to the current local attractor.

The steps of a MOPSO are iteratively repeated until a termination criterion is met. Note that our implementation of MOPSO additionally uses a "turbulence factor" [21] which introduces diversity into the swarm by randomizing the coordinates of a fraction of the particles within the area covered by the particles, i.e., within $[x_i^{min}, x_i^{max}]$ where x_i^{min} and x_i^{max} are the minimal and maximal coordinates for decision variable x_i over all particles in the population and the archive.

3.2 Parallelization

Here, we propose two parallelization techniques. We assume a small number of loosely connected computers as e.g. in a company network. Furthermore, we assume a central server maintaining the archive, distributing work to different processors and collecting the results. This assumption is in line with the usual grid technology. The parallel processors are assigned sub-swarms, and run independent MOPSOs based on these subswarms 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 focusing on 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.

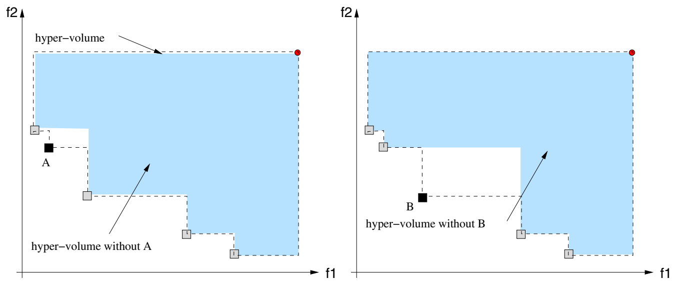


Figure 2: Selection of the global guide. Point B has a larger impact on the hyper-volume and therefore must be selected first. (□: solutions, ●: reference point)

3.2.1 Cluster-based subswarm MOPSO

The idea of the cluster-based subswarm MOPSO (CMOPSO) 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. The master processor's algorithm is sketched in Algorithm 1.

Algorithm 1 Cluster-based subswarm MOPSO (Master)

```
Initiate subswarms
Wait for results of all subswarms
repeat
  Update archive
  Clustering to determine new guides
  Send guides to subswarms
  Wait for results of all subswarms
until Termination condition met
Return archive
```

3.2.2 Hypervolume-based subswarm MOPSO

In the hypervolume-based subswarm MOPSO (HMOPSO), guides are selected one by one according to their marginal hypervolume. The hypervolume is the area dominated by all solutions stored in the archive [28]. 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 2 illustrates this. Point A has a smaller contribution to the whole hyper-volume value than Point B. Therefore, Point B would be selected first.

Note that this guide selection process is asynchronous, the master's algorithm is outlined in Algorithm 2. 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 CMOPSO.

For both variants, the region used to initialize a subswarm around

Algorithm 2 Hypervolume-based subswarm MOPSO (Master)

```
Initiate subswarms
repeat
  if A subswarm returns results then
    Update archive
    Determine next guide according to hypervolume
    Spawn new subswarm
  end if
until Termination condition met
Return archive
```

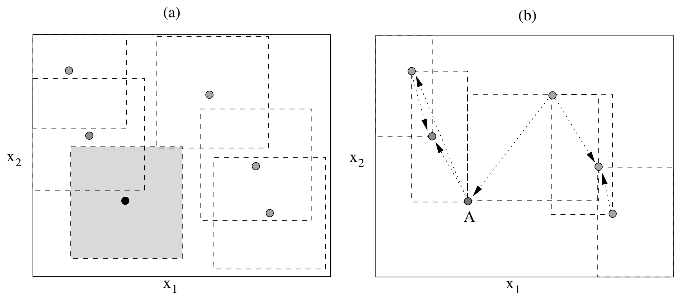


Figure 3: (a) A fixed sized area around each guide is selected to be the search space of the corresponding MOPSO for that guide. (b) The area in search space between surrounding neighbors in the objective space is selected to be the search space for the MOPSO. The neighbors are shown with arrows. For 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.

the guide can be chosen in different ways. In preliminary experiments, we tested the following three approaches:

- 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 $2m$ 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 3 shows the last two scenarios in an example with 2 parameters. • 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.

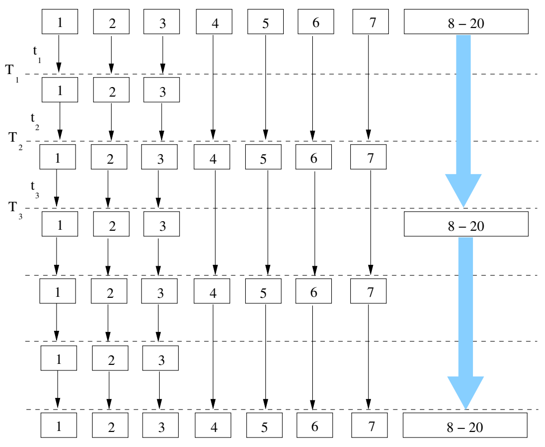


Figure 4: 20 heterogeneous processors are illustrated. t_1, t_2 and t_3 are equal.

4. EXPERIMENTS

In this section, we test CMOPSO and HMOPSO on two scenarios, one consisting of homogeneous processors, one consisting of heterogeneous processors. Furthermore, we compare its performance with several straightforward alternatives.

4.1 Parameter Settings

In both approaches, for a subswarm, a MOPSO method as proposed in [21] 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 selected test functions from literature [14] are 2-objective tests as in Table 1.

Table 1: Test functions.

Test	Function
FF	$f_1(\vec{x}) = 1 - exp(-\sum_i (x_i - \frac{1}{\sqrt{n}})^2)$
	$f_2(\vec{x}) = 1 - exp(-\sum_i (x_i + \frac{1}{\sqrt{n}})^2)$
	$x_i \in [-4,\ 4]$ and $n = 10$
OKABE	$x_1^{'} = \cos(\frac{\pi}{12})x_1 - \sin(\frac{\pi}{12})x_2$
	$x_2^{'} = \sin(\frac{\pi}{12})x_1 + \cos(\frac{\pi}{12})x_2$
	$f_1(\vec{x}) = x_1^{'}$
	$f_2(\vec{x}) = \sqrt{2\pi} - \sqrt{\lvert x_1^{'}\rvert} + 2\lvert x_2^{'} - 3\cos x_1^{'} - 3\rvert^{\frac{1}{3}}$
	$n = 2$
	$x_1 \in [6\sin(\frac{\pi}{12}),\ 6\sin(\frac{\pi}{12}) + 2\cos(\frac{\pi}{12})]$
	$x_2 \in [-2\sin(\frac{\pi}{12}),\ 6\cos(\frac{\pi}{12})]$

We examine both of the methodologies in homogeneous and heterogeneous environments. In order to simulate the heterogeneous environment, we consider 20 processors as shown in Figure 4. 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 4, 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 homogeneous environment uses 20 processors with the same speed, which corresponds to the speed of the slowest processor from the heterogeneous environment. Note that for CMOPSO, 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. HMOPSO, 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.

For a more in-depth evaluation, we additionally compare the results of our new algorithms to a number of straightforward alternatives:

Case1 Instead of running a parallel MOPSO, a simple alternative would just be to run a single MOPSO on the fastest processor available for the same time. While this scenario uses less computational power overall, it will tell us how much we can benefit from parallelization. Consequently, we use a single population with population size of 20 and run the optimization for $6 \times 6 \times 20 = 720$ generations (the fast processor computes 6 times 20 generations in each of the 6 cycles).

Case2 We could also let a single processor run for as many generations as all processors in our heterogeneous example together. In the above scenario we span 56 subpopulations per cycle. This leads to a population of 20 particles for $56 \times 6 \times 20 = 6720$ generations. This uses the same computational power as our heterogeneous HMOPSO, but takes much more time because it is not run in parallel.

Case3 In our parallel MOPSO, the 20 subpopulations are processed in parallel, i.e. the total population size at any time is actually 400. Thus, in Case 3 we run a single population for $\frac{56\times6\times20\times20}{400} = 336$ generations, which again corresponds to the same overall computational power as used for our heterogeneous HMOPSO.

Case4 Finally, we could run 20 independent MOPSOs on the 20 processors, merging the results of all the runs in the end, but otherwise without interaction. This means that 3 processors are run for $6 \times 6 \times 20 = 720$, 4 for $3 \times 6 \times 20 = 360$ and 13 for $2 \times 6 \times 20 = 240$ generations, all with a population size of 20 as in the heterogeneous HMOPSO.

The solutions are evaluated by measuring hypervolume [28] and C-metric values [28]. Hyper-volume is a measurement which takes into account the diversity as well as the convergence of solutions. High values of hypervolume indicate high diversity and convergence of solutions. For comparing two sets A and B in terms of their convergence, the C-metric of two sets A and B, $C(A, B)$, is defined as the percentage of solutions in B dominated by solutions in A. If all of the solutions in B are dominated by A, C(A,B) and C(B,A) are 1.0 and 0.0 respectively. All results reported below are averaged over 11 independent runs.

4.2 Evaluations

The results of CMOPSO as well as the heterogeneous and homogeneous HMOPSO are compared in Table 2 for the FF test function, and Table 3 for the OKABE test function, based on hyper-volume. These results are also visualized in Figure 5. In both test functions, the heterogeneous HMOPSO performs best, followed by the homogeneous HMOPSO and CMOPSO performing worst (as explained above, for CMOPSO, heterogeneous and homogeneous are identical). The superiority of heterogeneous HMOPSO over the homogeneous case shows clearly the benefit of allowing asynchronous operation in a heterogeneous computing environment. But also when comparing CMOPSO and HMOPSO on the homogeneous environment, HMOPSO performs better. Because both use exactly the same processing power, this difference has to be due to the strategy 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.

Table 2: Average values and std. error of Hyper-volume values computed for Heterogeneous HMOPSO (Hetero.HMOPSO), CMOPSO and Homogeneous HMOPSO (Homo.HMOPSO) methods after every cycle for FF test function.

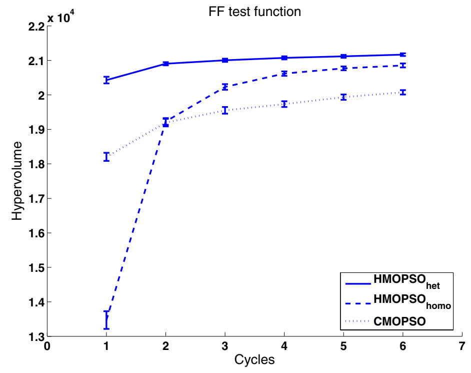
cycle	Hetero. HMOPSO	stderr	CMOPSO	stderr	Homo. HMOPSO	stderr
1	20429	96.57	18203	117.18	13473	256.92
2	20903	46.23	19196	110.82	19230	96.38
3	21004	46.98	19552	96.49	20230	81.02
4	21073	45.18	19733	83.86	20617	64.45
5	21118	42.46	19934	77.17	20769	54.52
6	21167	38.55	20075	64.51	20853	58.13

Table 3: Average values and std. error of Hyper-volume values computed for Heterogeneous HMOPSO (Hetero.HMOPSO), CMOPSO and Homogeneous HMOPSO (Homo.HMOPSO) methods after every cycle for OKABE test function.

cycle	Hetero. HMOPSO	stderr	CMOPSO	stderr	Homo. HMOPSO	stderr
1	28712	80.42	26405	79.5	27771	106.28
2	29006	72.29	27387	118.37	28272	119.28
3	29156	44.88	27794	112.95	28493	141.25
4	29220	47.28	28069	77.70	28611	130.94
5	29266	50.30	28252	50.30	28741	152.91
6	29315	52.40	28371	55.72	28789	157.84

As an alternative performance measure, Tables 4 and 5 show the C-metric values for the same scenarios. We observe that for the FF test function, the conclusions are very similar to the hypervolume metric: heterogeneous HMOPSO works best, followed by homogeneous HMOPSO and then CMOPSO. For the OKABE test function, the results are not so clear. Comparing the obtained fronts, we could see that HMOPSO is generating a larger spread of solutions, while CMOPSO finds slightly better solutions in the middle area of the Pareto front.

Figure 6 compares the average hypervolume (with error bars) obtained at the end of the run by the heterogeneous HMOPSO, the homogeneous HMOPSO, CMOPSO, and the four additional algorithms defined in Section 4.1. As would be expected, the single processor running the same time as the parallel algorithms clearly performs worst, demonstrating the advantage of using a parallel computing platform. The other three test cases perform comparably to CMOPSO, and all these perform worse than HMOPSO. The comparison with Case 4 (the independent MOPSOs run in parallel) shows that the indirect communication through the master node and the explicit division of subswarm to different regions of the search space by the selected guides is indeed advantageous. Somewhat surprising, the heterogeneous HMOPSO even performs better than Cases 2 and 3, which use the same processing power for a single swarm on a single processor. This indicates that the explicit separation in HMOPSO may be advantageous even when implemented on a single processor, due to its diversity maintenance (similar to the findings that the island model EA is often found superior to a single-population EA). Also, it is interesting that Case 2 (small population run for many iterations) and Case 3 (large population run for fewer generations) are almost equivalent, demonstrating the robustness of the used MOPSO. Finally, note that CMOPSO, al-



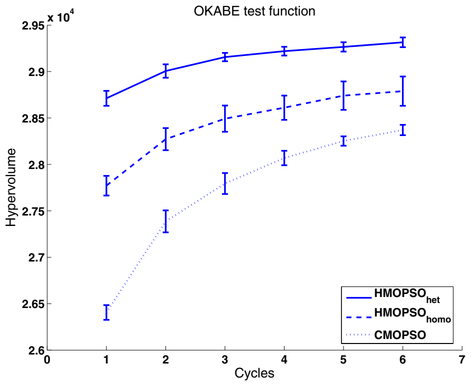


Figure 5: Hypervolume values computed for homogeneous and heterogeneous HMOPSO and CMOPSO methods over 6 cycles.

though similar in performance to Case 2-3, uses less computational power (the fast processor have to wait for the slow ones regularly).

5. CONCLUSION AND FUTURE WORK

In this paper, we proposed two new parallel methodologies of Multi-Objective Particle Swarm Optimization (MOPSO). One of the methods is particularly designed to also work in parallel environments with heterogeneous computing resources as is a typical setting for modern grid-computing environments. To our knowledge, it thus represents the first MOPSO specifically designed to work efficiently in such environments.

The basic idea behind these methods is to divide the population into subswarms which can be processed in parallel. Cluster-based MOPSO (CMOPSO) is designed to work on a fixed number of processors where hypervolume-based MOPSO (HMOPSO) is flexible to work on a heterogeneous set of processors. As the results show, HMOPSO outperformed CMOPSO 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. Moreover, HMOPSO clearly outperformed a set of straightforward alternative approaches like independent MOPSO runs or single population MOPSOs with an equivalent computational effort. In particular the latter finding is interesting, as it suggests that HMOPSO may also be helpful to improve MOPSO on a single processor.

Table 4: Average values and std. error of c-metric values computed for Heterogeneous HMOPSO (Hetero.HMOPSO), CMOPSO and Homogeneous HMOPSO (Homo.HMOPSO) methods after every cycle for FF test function.

cycle	C(Hetero. , Homo.) HMOPSO	stderr	C(Homo., Hetero.) HMOPSO	stderrr
1	1	0	0	0
2	0.926	0.012	0.014	0.004
3	0.831	0.016	0.045	0.005
4	0.751	0.019	0.078	0.006
5	0.735	0.016	0.099	0.005
6	0.720	0.017	0.107	0.006
cycle	C(Hetero.HMOPSO, CMOPSO)	stderr	C(CMOPSO, Hetero.HMOPSO)	stderrr
1	0.931	0.013	0.014	0.006
2	0.893	0.011	0.026	0.006
3	0.886	0.012	0.033	0.005
4	0.898	0.007	0.030	0.002
5	0.883	0.009	0.029	0.003
6	0.892	0.008	0.031	0.003

Table 5: Average values and std. error of c-metric values computed for Heterogeneous HMOPSO (Hetero.HMOPSO), CMOPSO and Homogeneous HMOPSO (Homo.HMOPSO) methods after every cycle for OKABE test function.

cycle	C(Hetero. , Homo.) HMOPSO	stderr	C(Homo., Hetero.) HMOPSO	stderrr
1	0.584	0.055	0.151	0.021
2	0.590	0.060	0.163	0.028
3	0.637	0.030	0.155	0.021
4	0.557	0.038	0.184	0.041
5	0.507	0.032	0.148	0.033
6	0.567	0.023	0.138	0.032
cycle	C(Hetero.HMOPSO, CMOPSO)	stderr	C(CMOPSO, Hetero.HMOPSO)	stderrr
1	0.477	0.037	0.278	0.018
2	0.371	0.033	0.330	0.032
3	0.327	0.039	0.319	0.041
4	0.259	0.033	0.321	0.040
5	0.235	0.031	0.332	0.035
6	0.198	0.033	0.339	0.032

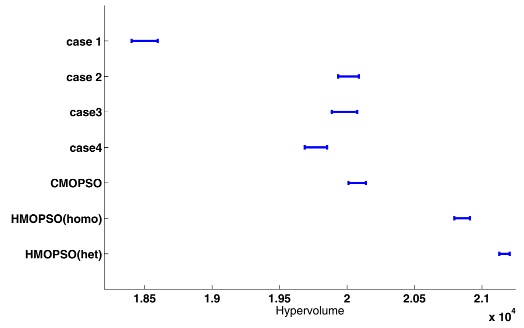
In future, we will test the approaches on a larger set of test problems and parallel environments. We will also test a number of possible improvements to the current approach, e.g. by allowing solutions with high marginal hypervolume to be re-selected after some time, even if some other non-dominated 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) [5] with a real world application.

6. 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, F. Ameida, M. Blesa, C. Cotta, M. Diaz, I. Dorta, J. Gabarro, C. Leon, G. Luque, J. Petit, C. Rodriguez, A. Rojas, and F. Xhafa. Efficient parallel LAN/WAN algorithms for optimization. the MALLBA project. *Parallel Computing*, 32:415–440, 2006.

[3] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–461, 2002.



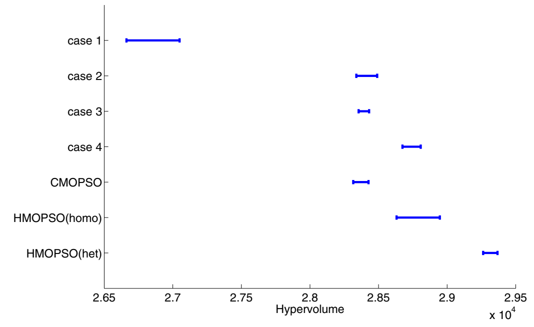


Figure 6: Hypervolume values for the FF (top) and OKABE (bottom) test functions. 'het' and 'homo' indicate the heterogeneous and homogeneous scenarios.

[4] 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., editor, *Evolutionary Multi-Criterion Optimization*, volume 3410, pages 459–73. Springer, 2005.

[5] M. Bonn, F. Toussaint, and H. Schmeck. Joschka: Job-scheduling in heterogenen systemen. In Erik Maehle, editor, *PARS Mitteilungen 2005*, pages 99–106. 20. PARS Workshop, 2005.

[6] J. Branke, A. Kamper, and H. Schmeck. Distribution of evolutionary algorithms in heterogeneous networks. In *Genetic and Evolutionary Computation Conference*, volume 3102 of *LNCS*, pages 923–934. Springer, 2004.

[7] J. Branke, T. Kaußler, and H. Schmeck. Guidance in evolutionary multi-objective optimization. *Advances in Engineering Software*, 32:499–507, 2001.

[8] J. Branke and S. Mostaghim. About selecting the personal best in multi-objective particle swarm optimization. In T. P. Runarsson et al., editor, *Parallel Problem Solving from Nature*, LNCS, pages 523–532. Springer, 2006.

[9] J. Branke, H. Schmeck, K. Deb, and M. Reddy. Parallelizing multi-objective evolutionary algorithms: cone separation. In *Congress on Evolutionary Computation*, pages 1952–1957, Portland, USA, 2004.

[10] S. Cahon, N. Melab, and E.-G. Talbi. ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004.

[11] E. Cantu-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer, 2000.

[12] 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.

[13] C. A. Coello Coello and M. S. Lechuga. Mopso: A proposal for multiple objective particle swarm optimization. In *Congress on Evolutionary Computation*, pages 1051–1056, 2002.

[14] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic, 2002.

[15] K. Deb, P. Zope, and A. Jain. Distributed computing of pareto-optimal solutions with evolutionary algorithms. In *Proceedings International Conference on Evolutionary Multi-Criterion Optimization (EMO03)*, pages 534–549, 2003.

[16] A. Hey, G. Fox, and F. Berman. *Grid Computing: Making The Global Infrastructure a Reality*. Wiley, 2003.

[17] 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.

[18] J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[19] J. Knowles and D. Corne. Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[20] F. Luna, A. J. Nebro, and E. Alba. Observations in using grid-enabled technologies for solving multi-objective optimization problems. *Parallel Computing*, 32:377–393, 2006.

[21] 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.

[22] S. Mostaghim and J. Teich. Covering pareto-optimal fronts by subswarms in multi-objective particle swarm optimization. In *Congress on Evolutionary Computation*, pages 1404–1411, Portland, USA, 2004.

[23] K. E. Parsopoulos, D. K. Tasoulis, and M. N. Vrahatis. Multiobjective optimization using parallel vector evaluated particle swarm optimization. In M. H. Hamza, editor, *IASTED International Conference on Artificial Intelligence and Applications*, pages 823–828, 2004.

[24] 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.

[25] M. Reyes-Sierra and C. Coello Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computational Intelligence Research*, 2(3):287–308, 2006.

[26] 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.

[27] F. Streichert, H. Ulmer, and A. 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.

[28] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. Shaker, 1999.