Distribution of Evolutionary Algorithms in Heterogeneous Networks

Jürgen Branke, Andreas Kamper, and Hartmut Schmeck

Institute AIFB, University of Karlsruhe, 76128 Karlsruhe,Germany
{branke,kamper,schmeck}@aifb.uni-karlsruhe.de

Abstract. While evolutionary algorithms (EAs) have many advantages, they have to evaluate a relatively large number of candidate solutions before producing good results, which directly translates into a substantial demand for computing power. This disadvantage is somewhat compensated by the ease of parallelizing EAs. 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". However, unlike dedicated parallel computers with a number of identical processors, the computers forming a grid are usually quite heterogeneous. In this paper, we look at the effect of this heterogeneity, and show that standard parallel variants of evolutionary algorithms are significantly less efficient when run on a heterogeneous rather than on a homogeneous set of computers. Based on that observation, we propose and compare a number of new migration schemes specifically for heterogeneous computer clusters. The best found migration schemes for heterogeneous computer clusters are shown to be at least competitive with the usual migration scheme on homogeneous clusters. Furthermore, one of the proposed migration schemes also significantly improves performance on homogeneous clusters.

Keywords. Evolutionary Algorithm, Heterogeneous Networks, Parallelization, Island Model, Grid Computing

1 Introduction

Evolutionary algorithms (EA) are randomized search techniques inspired by natural evolution. They have proven to work successfully on a wide range of optimization problems. While they have many advantages, they are computationally expensive since they have to evaluate a relatively large number of candidate solutions before producing good results. This drawback is partially compensated by the apparent ease of parallelizing EAs.

Consequently, there is a wide range of publications on how to best parallelize EAs. However, basically all these publications assume identical (homogeneous) processors, as it is usually the case if one has access to a dedicated parallel machine. More recently, however, it became possible to also harness the combined computing power of any heterogeneous set of networked computers, also known as "computer grids" (see e.g. [10]). These computer grids make the power of parallelization available to a much larger

K. Deb et al. (Eds.): GECCO 2004, LNCS 3102, pp. 923–934, 2004.

group of people, as for example most companies have a network of (heterogeneous) PCs, and also the Internet is basically a huge computer network. The power of computer grids has been demonstrated e.g. by the project seti@home [1], which connected thousands of computers to search for extraterrestrial life, or by the many applications in drug design [7]. Companies like Parabon [2] or United Devices [3] commercialize the idea of networked computing. Clearly, computer grids have the potential to resolve the problem of high computer resources required by EAs, and would help pave their way to an even more widespread application.

However, as we show in this paper, approaches to parallelize EAs for homogeneous parallel computers can not readily be applied in heterogeneous environments, or at least their efficiency drops significantly. Our goal therefore is to develop parallelization schemes particularly suitable for heterogeneous computer clusters. More precisely, in this paper we consider an island model and examine different aspects of migration, like the connectivity pattern or the time for migration.

The paper is structured as follows: In the following section, we will provide a brief overview on related work. Then, we will explain the experimental setup and show how heterogeneity affects standard parallelization approaches. Section 5 looks at the influence of the connectivity pattern, Section 6 compares sender- and receiver-initiated migration, and Section 7 suggests migration based on a population's level of convergence. The paper concludes with a summary and some ideas for future work.

2 Related Work

As has already been mentioned in the introduction, parallelization of EAs is relatively straightforward. The genetic operators crossover and mutation as well as the evaluation can be performed independently on different individuals, and thus on different processors. The main problem is the selection operator, where global information is required to determine the relative performance of an individual with respect to all others in the current population. There is a vast amount of literature on how to parallelize EAs. The 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 from 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.
3. Diffusion model: Here, the individuals are spatially arranged, and mate with other individuals from the 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 parallelization approaches is out of the scope of this paper. The interested reader is referred to e.g. [13,8,5].

In loosely coupled networks such as computer grids, due to slow communication, the island model seems to be the most promising approach. Therefore, in the remainder of this paper, we will focus exclusively on the island model.

Almost all papers on parallelizing EAs assume equally powerful (homogeneous) processors, only very few deal with heterogeneity. While in a homogeneous network, populations usually exchange individuals in a synchronized way, this doesn't make sense in a heterogeneous network, as the faster processors would always have to wait for the slowest one. Chong [9] was among the first to look at this aspect and naturally concluded that communication should be non-blocking (i.e. asynchronous) and buffered, a result that has been confirmed in [11]. In [4], the performance of the island model is compared on a number of small homogeneous and heterogeneous network clusters, again with asynchronous migration. The DREAM project [6] distributes EAs over the Internet and thus naturally has to deal with heterogeneous computers. Again, communication is asynchronous, but otherwise the aspect of heterogeneity is not addressed explicitly.

3 Experimental Setup

We base our experiments on the standard island model, with a more or less standard EA running on each island. Each island has a population size of 25, and a $(25 + 12)$ reproduction scheme is used, i.e. in every generation, 12 offspring are generated and compete for survival with the individuals from the old population. Crossover type is two-point, mutation is Gaussian with step size $\sigma = 1.0$. Islands are connected in a ring. In a migration step, each processor sends a copy of its best individual to its two neighboring populations, where it replaces the worst individual. Unless stated otherwise, migration is executed always at the end of a generation.

In order to have full control over all aspects of the environment, we simulate the underlying computer network. The network's computing power is assumed to be such that a total of 1920 generations (equivalent to $1920 \cdot 12 = 23040$ evaluations) can be computed per virtual time unit. In this paper, we ignore communication time and bandwidth restrictions and focus primarily on the aspect of different processing speeds.

The performance measure is the best solution quality obtained after 300 time units, which corresponds to almost 7 Mio. evaluations.

In the case of *homogeneous* processors, we assume to have 64 processors of equal speed, i.e. each processor computes 30 generations per virtual time unit. Because all processors have the same speed, all populations exchange migrants synchronously.

For the *heterogeneous* case, we assume 63 processors with a scale-free distribution of processing power, i.e. there is 1 processor capable of computing 320 generations per virtual time unit, 2 processors capable of computing 160 generations, 4 processors capable of computing 80 generations, down to 32 processors capable of computing 10 generations. The total computing power is the same as in the homogeneous case. In heterogeneous networks, islands communicate asynchronously using a buffer: Each island has a buffer, and a population which wants to send migrants to their neighbors simply puts them in the respective buffer. After each generation, a population checks

its buffer for possible immigrants and, if there are any, integrates. This scheme largely corresponds to the asynchronous migration scheme proposed in [11], except that our buffers are unlimited. Note that in the case of neighboring islands with very different speeds, the buffer may contain more individuals than the island's population size.

For our comparisons, we use the following 30-dimensional test function which has once been suggested by Michalewicz for a conference competition:

$$f(x) = \sum_{i=0}^{29} sin(y_i) * \left(sin\left(\frac{i+1}{\pi} y_i^2 \right) \right)^{20}$$

$$y_i = \begin{cases} x_i * cos(\frac{\pi}{6}) - x_{i+1} * sin(\frac{\pi}{6}) & : \quad i = 0, 2, 4, \ldots, 28 \\ x_i * cos(\frac{\pi}{6}) + x_{i-1} * sin(\frac{\pi}{6}) & : \quad i = 1, 3, 5, \ldots, 29 \end{cases}$$

$$-2 \leq x_i \leq 2 \qquad i = 0 \ldots 29$$

When comparing different migration schemes, whether one approach works better than another may strongly depend on the parameter settings. We assume that the most critical parameter in our setting is the migration interval, i.e. the number of generations between migrations. Therefore, for each migration scheme, we tested migration intervals of 10, 50, 100, 150, 200, 250, 300, 350, 400, and 450 generations, and usually report on the results of the best parameter setting for the respective migration scheme. All results reported are averaged over 120 runs.

4 EAs in Homogeneous and Heterogeneous Networks

Figure 1 compares the performance of the standard island model on the homogeneous and the heterogeneous network. Since we expected that the sorting of the processors in the ring could influence the result, for each of the 120 test runs, the sorting was generated at random. As can be seen, the heterogeneous network converges much faster, but the final solution quality is significantly worse. The faster convergence is due to the fact that the faster processors can perform a larger number of generations per time unit, thereby faster advancing search. However, this is at the expense of diversity: The good individuals from the fast processors will soon migrate to slower populations and dominate them, while the individuals from the slower processors are usually not good enough to compete with the individuals from other populations, and their genetic material is lost.

The steps in the curve of the homogeneous network are a result of the synchronous communication at migration. All populations export and import at the same time, and whenever new genetic material is introduced by migration, search seems to benefit and it is likely to quickly find better solutions. So even in the average of 120 runs we can see these bursts of improvement after every migration step. In the heterogeneous network, the communication is asynchronous, and the performance boosts due to migration average out. The steps that are visible are at points in time when all islands happen to migrate at the same time.

The final results of all test runs are also reported at the end of this paper in Table 1. Table 2 contains a pairwise comparison of the different migration schemes and results of a t-Test to determine significance. As can be seen, the difference between runs on homogeneous and heterogeneous networks is highly significant.

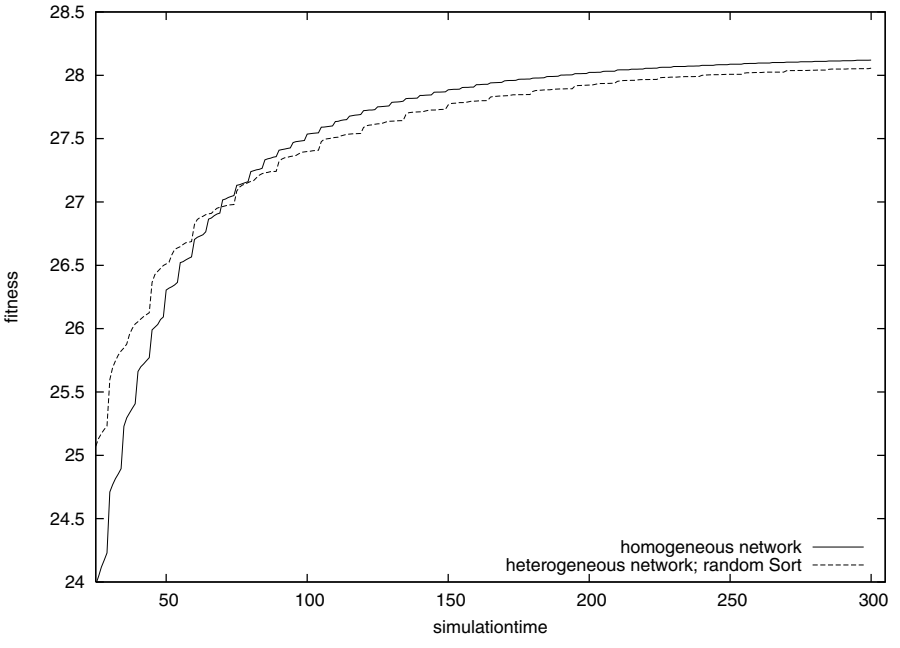

Fig. 1. Convergence plot, comparison of EA in homogeneous and heterogeneous networks, export-oriented migration.

5 Sorting

The first attempt to improve performance in a heterogeneous environment was to look at the sorting of the processors on the ring topology. In the tests reported before, we used a random sorting. Now we introduce two extreme ways of sorting the islands.

The first way is called "minimal difference sum sort" (MinSumSort) in which we minimize the sum of all speed differences between adjacent islands. The other way is to maximize those differences (MaxSumSort). These sortings are not unique. There are different possibilities for a given sum to arrange the islands. Figure 2 shows, exemplary for 15 populations, the sortings that we used for the following tests.

Figure 3 compares the convergence curves for the three chosen sortings MinSumSort, MaxSumSort, and random sorting, together with the homogeneous case for reference. Again, there seems to be a trade-off between fast convergence to an inferior solution, and slower convergence to a better solution. The MinSumSort converged quickest, but to the worst solution, while MaxSumSort is the slowest in the beginning, but yielding the best solution in the end. The random sort, as expected, is somewhere in between. Actually, the good performance of MaxSumSort came as a surprise to us, as we assumed that in the case of neighboring processors having quite different speeds, diversity would be lost much faster. However, a closer examination showed that the slow processors are, by themselves, not competitive. For MinSumSort, it takes a very long time for a good individual to travel from the fastest processor to the slowest islands. This results in very

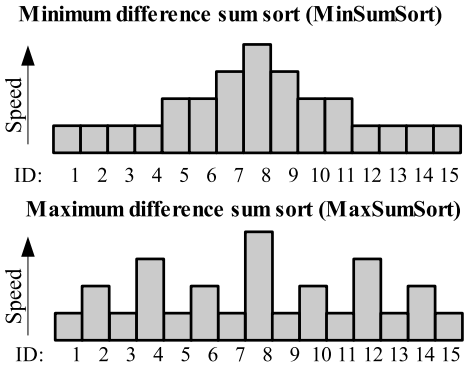

Fig. 2. Minimum and maximum difference sum sorting at the example of 15 populations

different fitness levels between the fast and slow islands, the slow islands' populations are simply not competitive. The power of these islands is lost because virtually every individual that the population exports will be rejected immediately by the receiving island due to its inferior fitness. On the other hand, in the case of MaxSumSort, slow islands surround every fast island. Although they quickly loose their own individuals and can not really follow their own search direction, they serve as a buffer between faster populations, and since they always work on relatively good individuals (migrated from its faster neighbors), even with their low computing power, they sometimes generate good individuals. They may even be regarded as melting pots, where good individuals from the two faster neighbors can meet on "neutral ground".

Another interesting result when comparing the test runs for the different sortings is that the optimal migration interval is different: it increases from 100 generations for MinSumSort over 150 generations for random sorting to 200 generations with MaxSumSort. If communication time is significant, that is another advantage for MaxSumSort, requiring only half the number of communications as MinSumSort. The different optimal migration intervals also confirm the above observations that for MinSumSort, the distance between fast and slow processors is too large, which is partially compensated by a short migration interval. On the other hand, MaxSumSort benefits from larger migration intervals, which help to maintain diversity and give the slower populations a chance to improve on the imported individuals.

Looking at Table 2 confirms that the differences between the three considered sortings are significant. The best sorting, MaxSumSort, is almost as good as the homogeneous network; the difference is not significant.

Overall, if the sorting can be controlled, MaxSumSort is clearly the sorting to be preferred. However, in many practical grid environments, the use of the computers is not exclusive, and the load and thus the preferable sorting can change dynamically.

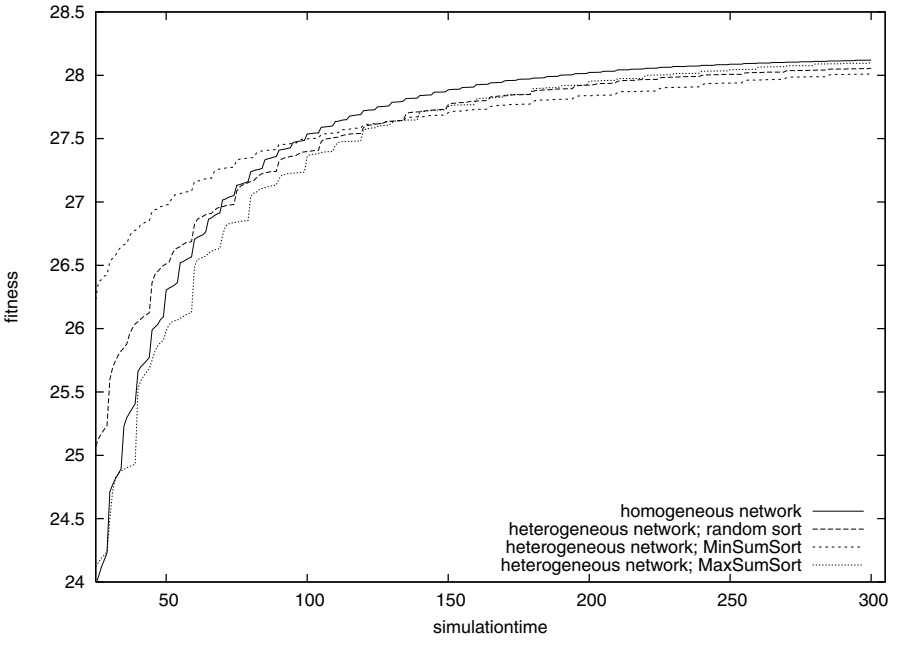

Fig. 3. Comparison of different sorting strategies (with homogeneous network for comparison)

6 Import, Export, Time

In a homogeneous network, it doesn't matter whether importing or exporting islands initiate the migration. For heterogeneous networks however, faster populations would like to migrate much more often than slower populations. In the previous experiments, as in all other publications we are aware of, we assumed that migration is initiated by the exporting island. That is, after an island has reached its migration interval, it exports a copy of its best individual to the buffers of its neighbors. If the speed difference between two islands is high, the buffer must be very large or old individuals must be removed to make space for new ones. In such a situation it is possible that all individuals in a population are replaced by individuals from the buffer at migration. Good search directions could be destroyed because of a continuous flow of new individuals form a faster island. An alternative way of communication would be to have migration initiated by the importing population. In that case, a population that has reached its migration interval would ask its neighbors to send over their current best individual. Thus, the number of individuals imported depends on an island's own speed and not on the speed of its neighbors. Good individuals or genes from fast processors will still spread, but presumably slower than by an export-oriented strategy. Figure 4 compares the import and export oriented migration strategies. There seems to be relatively little difference in performance, with the import-oriented yielding a slightly higher final solution quality, and export-oriented converging slightly faster. According to Table 2, the difference of the final fitness is not significant. Note that the import-oriented strategy has a slightly

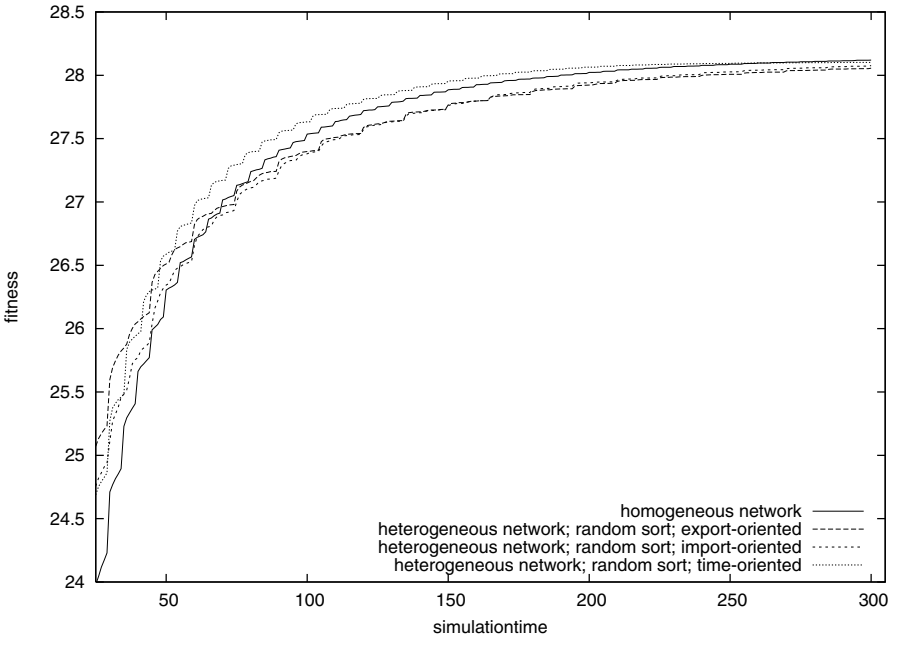

Fig. 4. Comparison of import- and export- and time-based migration strategies (with homogeneous network for comparison)

higher communication overhead because a population has to request immigrants. On the other hand, memory usage is reduced because there is no need for a buffer anymore.

Besides the just discussed import-initiated and export-initiated migration, migration could be triggered independently by clock time. In that case, communication would again be synchronous, as in the homogeneous network, where all population communicate at the same time. We tested this strategy again for different migration intervals. Best results for this testfunction were achieved for migration every 6 time steps, which corresponds to an average of 183 generations per population, i.e. the migration interval is in the same range as was optimal for export- or import-oriented migration schemes. The result is also included in Figure 4. As can be seen, the runs with time-based migration are slightly better than either export-based or import-based migration. Its main advantage however, is a much faster convergence. Overall, synchronous time-based migration seems to be the best choice, performing just as good as synchronous migration on a homogeneous network, but with a somewhat faster convergence.

7 Convergence-Based Migration

The experiments have shown that migration is an important feature, and the infusion of new genetic material into a population at the right time boosts performance. If good individuals are imported too early, the new individuals take over the population and

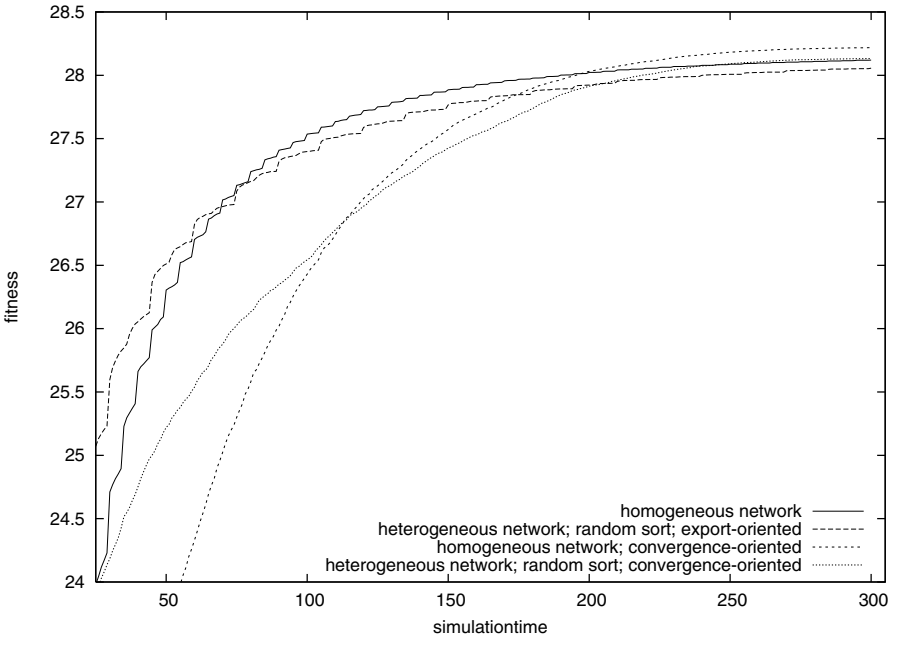

Fig. 5. Comparison of export-oriented and convergence based migration on homogeneous and heterogeneous environments.

are likely to re-direct the current (possibly good) search focus. On the other hand, if migration is done too late, search stagnates due to loss of diversity.

Therefore, it seems reasonable to have each island decide independently on the time it would like to import new individuals from its neighbors, based on its degree of convergence. Munetomo et al. [12] have previously suggested to initiate migration (import and export) whenever the standard deviation of a population's fitness distribution falls below a certain threshold. In this section, we propose to only initiate import, and use the number of generations without improvement[1] as an indicator for convergence.

That way, even slow populations get a chance to continue undisturbed and to explore their current search focus until they get stuck. Then, they can ask for new genetic material from their neighbors.

Note that this method may be applied in heterogeneous as well as homogeneous networks and leads to asynchronous communication in any case.

Figure 5 compares the results of the different approaches and the reference curves. As can be seen, the convergence-based migration improves over the standard generation-based strategies in the heterogeneous as well as the homogeneous network. Differences in final solution quality are significant (cf. Table 2). While overall convergence of these strategies is slower than by all other variations, they yield the best final results.

[1] We tested the same numbers of generations without improvement as the migration intervals for the generation-based strategies. Again, we only report on the best results

Overall, with convergence-based migration on a randomly sorted network we have found a strategy which is competitive with the standard synchronous migration on homogeneous networks. If convergence-based migration is combined with MaxSumSort sorting, performance can be further improved, and results get significantly better than the standard migration in homogeneous environments (cf. Tables 1 and 2).

Furthermore, convergence-based migration can also be applied in homogeneous environments, and then outperforms all other tested strategies by a significant margin.

8 Conclusion

The availability of networked computer clusters, so-called "grids", is a great possibility to harness the power of evolutionary algorithms without having to invest in dedicated parallel computers. However, while parallel computers usually have a number of identical (homogeneous) processors, computer grids usually consist of computers with very different speeds.

So far, almost all literature on parallel evolutionary algorithms assumed homogeneous clusters. In this paper, we have examined the island model on heterogeneous computer grids and shown that the standard migration schemes suffer significantly in performance when compared to their application in homogeneous environments.

As a consequence, we proposed and compared a number of alternative migration schemes for the island model, specifically targeted at heterogeneous computer grids. We demonstrated that the solution quality and convergence speed are influenced by the sorting of computers in a ring structure, and that specific sortings perform better than others. We have compared strategies which base the time of migration on the importing population, the exporting population, or an external clock. Furthermore, we proposed to make migration adaptive by allowing each population to request new immigrants from its neighbors whenever it has converged. This strategy, which is also applicable in homogeneous networks, turned out to perform best.

Overall, the paper provided new insights into the role of migration and the behavior of the island model on heterogeneous computer clusters. We have shown that by sorting the computers appropriately in the ring structure, results competitive to homogeneous networks can be obtained. Furthermore, the suggested convergence-based migration, which is also applicable in homogeneous networks, leads to a further significant improvement in either case.

There are several avenues for future work. First of all, the presented results should be examined for their sensitivity to changes in the problem instance, the heterogeneity of the computers, the number of computers in the network, the island model topology, etc. Furthermore, strategies should be explored to cope with dynamic computer speeds, as in a computer grid, the computers can usually not be used exclusively, and the effective speed of a computer may vary with its workload. Finally, communication costs like latencies should be considered.

References

1. http://setiathome.ssl.berkeley.edu/.
2. http://www.parabon.com.
3. http://www.ud.com/home.htm.
4. E. Alba, A. Nebro, and J. Troya. Heterogeneous computing and parallel genetic algorithms. *Journal of Parallel and Distributed Computing*, pages 1362–1385, 2002.
5. E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–461, 2002.
6. M. Arenas, P. Collet, A. Eiben, M. Jelasity, J. Merelo, B. Paechter, M. Preuß and M. Schoenauer. A framework for distributed evolutionary algorithms. In *Parallel Problem Solving from Nature*, pages 665–675. Springer, 2002.
7. R. Buyya, K. Branson, J. Gidy, and D. Abramson. The virtual laboratory: a toolset to enable distributed molecular modelling for drug design on the world-wide grid. *Concurrency and Computation: Practice and Experience*, 15:1–25, 2003.
8. E. Cantu-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer, 2000.
9. F. S. Chong. Java based distributed genetic programming on the internet. Technical report, School of Computer Science, University of Birmingham, B15 2TT, UK, 1999.
10. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999.
11. P. Liu, F. Lau, and J. Lewisand C. Wang. Asynchronous parallel evolutionary algorithm for function optimization. In *Parallel Problem Solving from Nature*, pages 405–409. Springer, 2002.
12. M. Munetomo, Y. Takai, and Y. Sato. An efficient migration scheme for subpopulation-based asynchronously parallel genetic algorithms. In S. Forrest, editor, *International Conference on Genetic Algorithms*, page 649. Morgan Kaufmann, 1993.
13. 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.

Appendix: Numerical Results

Table 1. Fitness of different migration strategies after 100, 200, and 300 time steps. Numbers are averages over 120 runs.

Strategy	Fitness after 100 time units	Fitness after 200 time units	Fitness after 300 time units
Heterogeneous network			
random sort export-oriented	27.39	27.92	28.05
minSumSort export-oriented	27.49	27.83	28.02
maxSumSort export-oriented	27.35	27.95	28.10
random sort import-oriented	27.37	27.94	28.07
random sort time-based	27.85	28.04	28.10
random sort convergence-based	26.54	27.91	28.13
maxSumSort convergence-based	26.11	27.96	28.15
Homogeneous network			
Standard EA	27.53	28.02	28.12
convergence-based	26.43	28.03	28.21

Table 2. Pairwise comparison of the different migration strategies and cluster characteristics based on the fitness after 300 time steps (values in brackets). Upper triangular matrix depicts T-value of a two-sided t-Test, lower triangular matrix contains a "+" if the results differ significantly with significance level 97.5%, and "-" otherwise.

			Heterogeneous							Homogeneous	
			random				MinSum	MaxSum			
			export (28.05)	import (28.07)	time (28.10)	conv. (28.13)	export (28.02)	export (28.10)	conv.(28.15)	standard (28.12)	conv. (28.21)
heterogeneous	random	export (28.05)		1.25	3.10	4.92	2.50	3.31	7.98	4.68	11.98
		import (28.07)	-		1.71	3.77	3.91	1.85	6.59	3.24	11.52
		time (28.10)	+	-		2.10	6.17	0.0	4.74	1.44	9.27
		conv. (28.13)	+	+	-		8.19	2.3	2.24	0.90	7.64
	minSum	export(28.02)	+	+	+	+		6.61	12.17	8.19	16.74
	maxSum	export(28.10)	+	-	-	+	+		5.16	1.56	10.28
		conv.(28.15)	+	+	+	+	+	+		3.52	6.10
hom.	standard(28.12)		+	+	-	-	+	-	+		8.9
	convergence (28.21)		+	+	+	+	+	+	+	+	