The Scalability of JavaSpaces for the Implementation of different Parallel Evolutionary Algorithms

Christian Setzkorn, Ray C. Paton

Department of Computer Science, University of Liverpool, UK

Abstract. Evolutionary algorithms (EAs) have been successfully deployed in many areas such as medicine, biotechnology, robotics, telecommunications, and economics to mention only a few. However, EAs do not scale up when the problems to be tackled become more complicated. This has given rise to parallel evolutionary algorithms (PEAs). The implementation of parallel applications, however, is not trivial, often requiring specific hardware and software settings and frequently resulting in barely reusable programs. This study investigates the JavaSpaces technology, which is known to simplify the implementation of parallel applications. It is independent of hardware and software settings and allows the implementation of highly reusable applications. We utilised the JavaSpaces technology to implement three PEA paradigms: the synchronous master slave PEA, the asynchronous master slave PEA and the coarse grained PEA. We examine the scalability of the three PEA implementations for a particular problem: the construction of fuzzy classification rules from data.

1 Introduction

Evolutionary algorithms (EAs) are powerful problem-independent heuristic search algorithms [35]. They have been successfully used in areas such as medicine, biotechnology, robotics, telecommunications, and economics to mention only a few [2]. However, EAs require a large amount of computational resources, especially when a problem tackled becomes more and more complicated and/or when the fitness evaluation of the EA turns out to be too costly. For these reasons many researchers have put forward paradigms to execute EAs in parallel. This has given rise to Parallel Evolutionary Algorithms (PEAs). The implementation of PEAs, however, is complicated. It requires specific hardware and software settings and results in very problem-specific and barely reusable applications.

This study investigates a relatively new technology known as JavaSpaces[1] [24] that may help to overcome the afore-mentioned constraints. This technology is known to simplify the implementation of parallel/distributed applications and is independent of hardware and software settings. We have implemented three well-known PEA paradigms: synchronous master-slave PEA (SMS-PEA), asynchronous master-slave PEA (AMS-PEA) and coarse-grained PEAs (CG-PEA). These paradigms have been chosen because they can be seen as supersets of currently existing PEAs. We investigated their scalability on a particular problem - the construction of fuzzy classification rule systems from data. Naturally the frameworks used can also be applied to other problems.

[1] JavaSpaces is a trademark of SUN Microsystems, Inc.

It is important to note that we only investigated the scalability of the three implementations rather than their capability of finding good solutions for a given problem. Our intention is to make researchers aware of the JavaSpaces technology and to show that it can be used to implement different PEA paradigms to tackle difficult problems.

To keep this report self-contained we provide a brief introduction to EAs in section 2. Section 3 describes the problem of fuzzy classification rule system inference from data. Section 4 briefly introduces the principles of the three mentioned PEA paradigms and provides the reader with further references to other PEA approaches and PEA reviews. Section 5 describes the JavaSpaces technology together with an account of our use of this technology to implement the three PEA paradigms. Section 6 outlines the chosen methodology and presents the results. The conclusions follow in section 7. Finally the possible avenues for future work are suggested in section 8.

2 EAs a brief Introduction

Evolutionary Algorithms (EAs) is a generic term for a computer programs that aim to solve problems by running processes that are inspired by nature [29]. Many varieties of EAs exist [34]. They were independently put forward by numerous researchers [14]. Examples of EAs are Genetic Algorithms (GA), Evolutionary Strategies (ES) and Evolutionary Programming (EP) [19][22] [32][13][33][3][28]. The EA processes act on a population of candidate solutions (individuals) for a given problem. Three distinct processes are usually recognised: genetic operators, fitness evaluation and selection.

The genetic operators processes recombine and slightly change a particular number of individuals within a given population. The fitness evaluation process determines how well each individual solves the given problem. This is usually expressed by a numeric value - the individual's fitness. The selection process generates a new population of individuals (offspring) from the parent population. The newly generated population typically exhibits the same or a larger average fitness than the original population. There are numerous methods to implement such a process [4]. It should be clear that the main objective of this process is to emulate Darwin's principle of the survival of the fittest [9].

Figure 1 gives a high level description of an EA. It will be used to illustrate how the previously described processes act together to 'evolve' a population of highly fit individuals - solutions for a given problem.

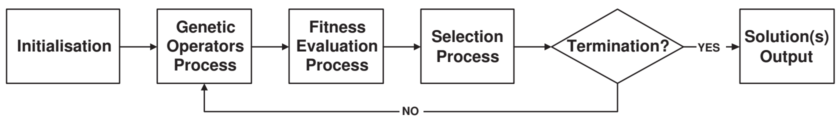

Fig. 1. Evolutionary algorithm structure.

An EA starts by passing a randomly generated population of individuals to the genetic operators process. This process applies genetic operators to some of the population's individuals. The resulting population is then passed to the fitness evaluation process, which evaluates the population's individuals and passes them to the selection process. The selection process then produces a new population of individuals. The termination test decides whether the EA terminates or whether another population (generation) of individuals is produced. This is done by passing the current population to the genetic operators process. A possible termination criterion is a maximum number of generations. The type of EA structure is often referred to as a generational EA because there is a flow of populations (generations) between the EA processes; in other words individuals 'evolve' in generational steps.

3 The Problem of Constructing Fuzzy Classification Rule Systems from Data

Classification systems (CSs) are models capable of determining (predicting) the class memberships of (unseen) cases from a particular domain. A case is represented by a number of features. Features can be quantitative (e.g. continuous) and/or qualitative (categorical) [25].

There are two main ways of constructing CSs: automatically, from pre-classified cases, and manually, utilising the knowledge of domain experts [25]. The automatic construction of CSs is usually much faster than the manual construction of CSs. Several methods for the automatic construction of CSs exist, originating from many scientific disciplines including statistics, machine learning and artificial intelligence. Many of the proposed methods, however, require a great deal of expertise for their correct deployment and/or for the understanding of the produced models. This can be disadvantageous; especially when the constructed model is used to formulate general hypotheses with regard to the observed domain [20].

One way to overcome this problem is the use of EAs for the construction of CSs from data. Here the CSs correspond to a particular knowledge representation form (an individual) that can easily be understood. The individual's performance is mainly determined by its capability of classifying presented cases. The evaluation of an individual, and a population of individuals in particular, can be very computationally expensive especially when there is a large number of cases.

In our research, we deploy EAs to infer fuzzy classification rule systems from data. This type of knowledge representation is very comprehensible and can deal with uncertain information that is very common in real classification systems [8]. An extensive introduction to the problem of fuzzy classifier design in general can be found in [25] whereas [7] provides an introduction to the construction of fuzzy classification rule systems utilising EAs.

4 PEAs - Motivation and Brief Review

When attempting to solve harder problems utilising EAs one frequently faces two problems: high memory demand and long computation time. The reason for high memory demand is because difficult problems often require the deployment of more individuals to find acceptable solutions. Longer computation time is the result of larger populations and costly fitness evaluations. It is, therefore, beneficial to distribute the increased memory and computational demands of EAs over several machines resulting in parallel evolutionary algorithms - PEAs.

There are two distinct groups of PEA paradigms. The first group changes the dynamics of the underlying evolutionary process whereas the second group does not. The afore-mentioned SMS-PEA paradigm belongs to the first group whereas the AMS-PEA and the CG- PEA belong to the second group. Many more PEA paradigms exist. Several excellent reviews of PEA paradigms are available [31] [1] [6]. The three PEA paradigms chosen in this study are now briefly described.

4.1 Synchronous Master-Slave Parallel Evolutionary Algorithm

Grefenstette proposed the first SMS-PEA in 1981 [21]. In this scheme an EA is executed on what is called a 'master' computer. The master is connected to several 'slave' computers to which it sends individuals when they require evaluation. The slaves evaluate the received individuals and send them back to the master. Basically, the fitness evaluation process is distributed over the available slave computers. This approach can significantly speed up EAs that deploy a computational expensive fitness evaluation process. In other words, when it is much more expensive (takes much longer) to evaluate an individual than to send and receive it (communication costs). The SMS-PEA is sometimes also referred to as global parallelisation [6], fitness parallelisation [27] or the standard parallel approach [1]. The SMS-PEA, however, has one disadvantage. It is not efficient in exploiting the power of the slaves. This is due to the idle times of the master and the slaves. The master has to stop the EA until all the slaves have returned the evaluated individuals; the slaves are idle until the master initiates the fitness evaluation again.

4.2 Asynchronous Master-Slave Parallel Evolutionary Algorithm

The AMS-PEA was also put forward by Grefenstette in 1981 [21]. In order to tackle the 'idle time problem' of the SMS-EA, a non-generational EA is executed on the master. Here, single individuals are transferred between the EA's processes rather than generations of individuals wrapped together in populations. This results in lower idle-times for both slaves and master. Slaves are constantly supplied with work in terms of individuals and the master can proceed to execute its evolutionary process as soon as a particular number of evaluated individuals has been received from the slaves. That number of individuals depends on the selection process used. For example the binary tournament selection [4] requires only two individuals to produce one offspring. The fact that the selection process does not work on the global population significantly changes the dynamics of the non-generational EA in comparison to the generational EA. As for the SMS-PEA, the fitness evaluation process is also distributed over several slaves. The AMS-PEA, however, still runs the remaining EA processes locally. Thus there may still be high communication costs since the individuals always have to be returned to the master that runs the actual EA.

4.3 Coarse Grained Parallel Evolutionary Algorithm

For the CG-PEA, each slave is equipped with an EA. Hence each slaves evolves a subpopulation (demes) of the global population. It is only occasionally that an exchange (migration) of individuals between the slaves takes place, which decreases the communication costs. The CG-PEA is also often referred to as island model or distributed EA [31]. Standard CG-PEAs have many parameters such as the number of demes, their size, the topology in which the slaves are connected, migration magnitude and frequency. This makes the CG-PEA relatively hard to configure [5]. Despite this, one can expect much lower computation time in comparison to the SMS-PEA and AMS-PEA due to lower idle times and communication costs.

5 Implementation of PEAs using JavaSpaces

This section provides a brief overview of the JavaSpaces technology and explains how it can be used to implement the previously described PEA paradigms.

5.1 JavaSpaces - An Introduction

SUN Microsystems Inc. has put forward the JavaSpaces specification in 1999. It was inspired by the concept of Linda [18]. Linda's core idea is that an object storage, which can be accessed by several processes using a small number of operations, can greatly simplify the implementation of parallel and distributed applications. The object storage is referred to as 'tuple space' [18], hence the name JavaSpaces. SUN Microsystems Inc. offers an implementation of the JavaSpaces specification[2], which utilises technologies such as Jini/RMI [26] [11] and the programming language JAVA [23], making the concept even more powerful. For an in-depth review of JavaSpaces see for example [10] [12].

As already mentioned, the processes which run within a network of computers, can access a space in a concurrent manner allowing them to communicate and coordinate their actions by writing and reading objects to and from that space. Objects are entities that are defined in terms of data and behaviours. They can represent numbers, strings or arrays, but also executable programs (e.g. an EA individual). The space itself handles the concurrent access and hence frees the programmer from problems such as race conditions. Objects can be exchanged using operations such as *write*, *read*, and *take*. A Space also offers what is called associative lookup, which enables processes to only take or read very specific objects from that space. There are many potential sources of failure within a network. For example, objects can get lost on the way between a writing/reading process and a space. Transactions [10] make operations such as write secure. A process can register its interest for a particular type of object with the space. If such an object is written into the space it is notified by the space via a RemoteEvent [30]. Another attractive property of a space is that the processes accessing it and the space itself can be executed on many different computer platforms due to JAVA's platform

[2] It can be downloaded free of charge from http://www.sun.com/consumer-embedded/cover/jstk-990615.html

independence. In summary, a space provides a shared, persistent, and securely accessible opportunity to exchange objects for processes in a computer network, allowing the simple implementation of parallel and distributed applications.

5.2 Synchronous Master-Slave Parallel Evolutionary Algorithm

Figure 2 depicts the JavaSpaces SMS-PEA implementation. It was originally inspired by the compute-server approach proposed by Freeman et al. [15] [10].

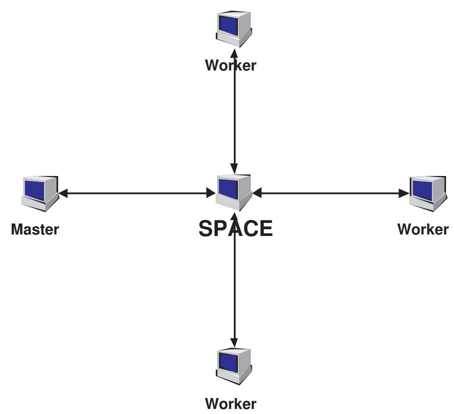

Fig. 2. SMS-PEA implementation using JavaSpaces.

The implementation works in a manner similar to that described in section 4.1. The master executes the actual EA. Copies of individuals that require evaluation are written into the space. When the master creates a copy of an individual it attaches to it a unique identifier as well as a web server address (URL) of the data that is required to evaluate the individual (see section 3). Workers (slaves) take individuals from the space (one at the time) and evaluate them. The evaluation requires data, which is downloaded from the web server indicated by the unique URL attached to each individual. Of course this is only done if the worker has not already downloaded the necessary data. A worker does not return the evaluated individual into the space but rather a 'result object' consisting of the resulting fitness value plus the unique identifier of the individual. The master takes result objects from the space and uses the unique identifier to update the fitness value of the original individual (only a copy has been written into the space). It is cheaper, in terms of communication costs, to send result objects back to the master rather than whole individuals. The master carries on with the evolutionary process as soon as it has taken all the result objects from the space.

There are many advantages of this approach. It is characterised by automatic load balancing because the workers can access the space concurrently. Workers representing more powerful computers, or those that receive simpler individuals can evaluate more of them. Workers can be added and removed while the evolutionary process is running.

Worker programs can be executed on many different hardware and software platforms because they are implemented in JAVA. Workers represent sleeping processes as long as no individuals are written to the space. This means that they do not steal any computational resources and could run/sleep on all the computers of an institution at all times. A policy could be established that individuals are only written into the space outside normal working times. In this manner all the computer power of an institution could be harvested without virtually any costs since the computers are often not switched off. The programming language JAVA also offers what is known as dynamic class downloading [11], which enables the workers to evaluate individuals originating from different EAs that are tackling different problems. As long as the object (a particular individual) that has been written in the space implements a specific interface (e.g. SpaceIndividual) a worker is capable of downloading the necessary class files from a web server to be able to execute and thus evaluate this individual. This means that a JavaSpaces-based PEA implementation can be reused (without shut-down) or could even be used by several users, running different EAs at the same time.

5.3 Asynchronous Master-Slave Parallel Evolutionary Algorithm

Figure 3 illustrates the JavaSpaces AMS-PEA implementation.

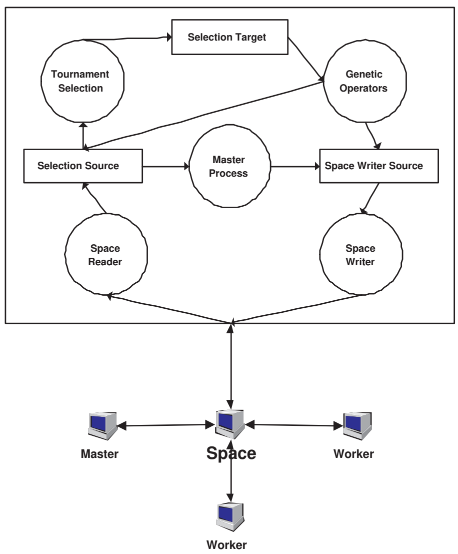

Fig. 3. AMS-PEA implementation using JavaSpaces.

The top part of figure 3 depicts the master running the asynchronous EA. It consists of synchronously accessible object (individual) storages (rectangles) and processes (circles). The processes are running in a concurrent manner on the master. Arrows indicate the flow of individuals. If a process encounters an empty storage while attempting to take individuals from it, it automatically goes to sleep. If a process writes individuals into an empty storage the sleeping process is then woken up (notified).

At the beginning, the AMS-PEA master process initialises a population of individuals and writes it into the storage denoted *space writer source*. The space writer process is notified and starts to take individuals from the space writer source and writes copies of them into the space. It also attaches the unique identifier and the web server address to the copies (see section 5.2). Workers take individuals from the space, evaluate them and write result objects into the space (as described in section 5.2). The *space reader process* takes result objects from the space and uses the attached unified identifier to remove the corresponding individuals from the space writer process. The individuals' fitness values are updated and written into the *selection source storage*. The *tournament selection process* performs the selection incorporating the individuals that are currently in the *selection source* and writes the selected individual in the *selection target*. The *genetic operators process* removes individuals from the *selection target* and applies genetic operators to them. Depending on whether a particular individual requires re-evaluation (genetic operators are only successful with a particular probability) it is either written into the *space reader source* for re-evaluation or into the *selection source storage*.

The end result is a constant (non-generational) flow of individuals between all the processes representing the EA. Only the *tournament selection process* can stop this flow acting as a dam. Only a limited number of individuals are written into the *selection target*. If the *tournament selection process* reaches this target, the *master process* is notified which then removes all individuals from the *selection source*, and presents them as the final population.

5.4 Coarse Grained Parallel Evolutionary Algorithm

Figure 4 depicts the JavaSpaces CG-PEA implementation.

As described in section 4.3 the workers (slaves) run their own EA. The encircled area in figure 4 depicts the internal structure of a worker and its EA. Again, the EA processes are depicted as circles and synchronously accessible object (individual) storages as rectangular. Arrows indicate the flow of individuals.

The workers of a CG-PEA are activated first. Each worker writes its IP-Address in the form of an object into the space and starts its local processes. Now the master can be started which takes all the IP-Addresses from the space. They are wrapped into one object and written back into the space. Each worker reads a copy of this object from the space. Essentially this makes the workers aware of each other. The master now initialises a population of individuals and writes them into the space. Each of these individuals has an IP-Address attached to it plus the web server address of the data (see section 5.2). An IP-Address is chosen with a uniform probability from the previously accumulated IP-Addresses. The *space reader process* reads those individuals from the space that contain the IP-Address of the machine on which it is running. The individuals are then written into the *genetic operators source*. The *genetic operator process* removes individuals from the *genetic operators source* and applies genetic operators to them. The process also evaluates individuals if genetic operators have changed them. All individuals are written into the *selection source*.

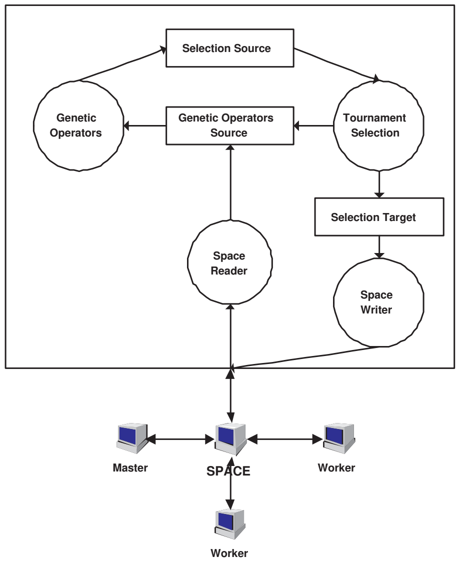

Fig. 4. CG-PEA implementation using JavaSpaces.

The *tournament selection process* works in the same manner as described in section 5.3. Again, it also acts as a dam. In the CG-PEA approach, each individual carries a loop-counter with it that is inherited to its children. Each time an individual passes through a *tournament selection process* its loop-counter is increased. If the loop-counter reaches a maximum value, an indicator is attached to the individual. This indicator makes it impossible for a *space reader process* to take such an individual from the space again. From the space the master can only take individuals that have this indicator. This enables the master to accumulate the final population. As long as the loop-counter of an individual has not yet reached its maximum value, the *tournament selection process* has to decide whether a particular individual is sent to another worker (migration) or whether it is written into the genetic operators source. An individual only migrates with a specific probability (migration probability). To migrate an individual, another worker's IP-Address is attached to it before it is written into the space. This IP-Address is sampled with a uniform probability from the other workers' IP-Addresses.

An interesting property of the JavaSpaces CG-PEA approach is that its behaviour only depends on two parameters: the migration probability and the number of workers. Our approach is also topology independent because all workers are virtually interconnected via the space. This makes the configuration of our CG-PEA much easier in contrast to other approaches (see section 4.3). The current implementation has, however, one disadvantage. Workers cannot be added arbitrarily as described in section 5.2 because the other workers do not have an 'awareness' of workers that are added subsequently.

6 Methodology and Results

6.1 Methodology

To investigate the scalability of the implemented PEA approaches we made two assumptions. Firstly, that the fitness evaluation process is the most computationally expensive EA process. Secondly, that the computational complexity of the fitness evaluation process is mainly determined by the number of cases that are used to evaluate an individual.

The investigation comprised twenty experiments. Six experiments for each of the three PEA approach and two experiments for a standard EA (see section 2). For each PEA experiment we utilised five, ten or twenty workers. There is no particular reason for the choice of these numbers apart from the fact that the number always doubles. Hence it could theoretically be expected that, for example, a PEA that deploys five workers has twice the execution time of the PEA that deploys ten workers.

Each experiment consisted of 100 EA runs. For each run we sampled N cases from 50000 existing cases with a uniform probability. These cases were randomly initialised and could belong to one of three classes. Each case consisted of five categorical and five continuous features. The particular feature values were randomly initialised. The number of cases N was determined randomly through uniform sampling from either the interval [200...6000] or [6000...200000]. We measured the execution time of each run. Preliminary experiments showed that all PEAs outperformed the standard EA approach in the range between [200...6000 cases] but also that the deployment of more than five workers is not necessarily beneficial within this range. We therefore sampled from this range rather than from the range [200...200000] in order to establish the threshold of out performance with a greater accuracy. Preliminary experiments have also shown that the deployment of more than five workers is beneficial for the range [6000...200000].

An EA run terminated after the selection process produced a maximum number of 1000 individuals. This termination criterion enabled us to compare the execution times of generational and non-generational approaches. This termination criterion also works for the CG-PEA approach, although it has several selection processes. The loop counter enables us to measure how many individuals all selection processes have produced globally (see section 5.4).

All evolutionary approaches utilised the same processes (fitness evaluation, selection and genetic operators) and parameters (crossover rate 0.7, mutation rate 0.01, number of individuals 100 etc.). The individuals had a fixed size to keep the communication costs steady. We used a migration probability of 0.1 for the CG-PEA. This means that on average every tenth individual that the selection process of a worker's EA 'produces' is migrated to another worker.

The hardware and software configuration that was available for our experiments was as follows: All worker (slave) implementations were executed on PCs with the operating system Linux, a single AMD Athlon 1.3 GHz processor and 512 MB PC133 SDRAM memory. The master implementations were executed on a PC with the operation system Windows 2000, and a single AMD Athlon 850 MHz processor and 256 MB PC100 SDRAM memory. JavaSpaces was executed on a machine with the same specification as that of the workers. We used the JavaSpaces implementation that comes with

Jini 1.4. JAVA programs utilised JDK 1.4. The utilised web server was a HP RP 2400 machine, with one PA 8500 RISC 440 MHz processor, 630 MB memory and Apache 1.3.6 under the HP-UX 11.00 operating system. All computers were connected via a 100 Mbit switched network. It is important to note that other users could also use all the machines, except the master, while we performed the experiments.

As stated above, an experiment consisted of a hundred pairs of independent values (number of cases) and dependent values (execution time in seconds). In order to compare experiments visually we used the statistical package SAS [3]. For each experiment we fitted a regression line through the data and depicted the ninety-nine percent confidence interval for the mean predicted values for each experiment.

6.2 Results

Figure 5 compares the five-worker SMS-PEA with the standard EA for the interval [6000 ... 200000]. It indicates that the five-worker SMS-PEA clearly outperforms the standard EA when a large number of cases is used for the fitness evaluation.

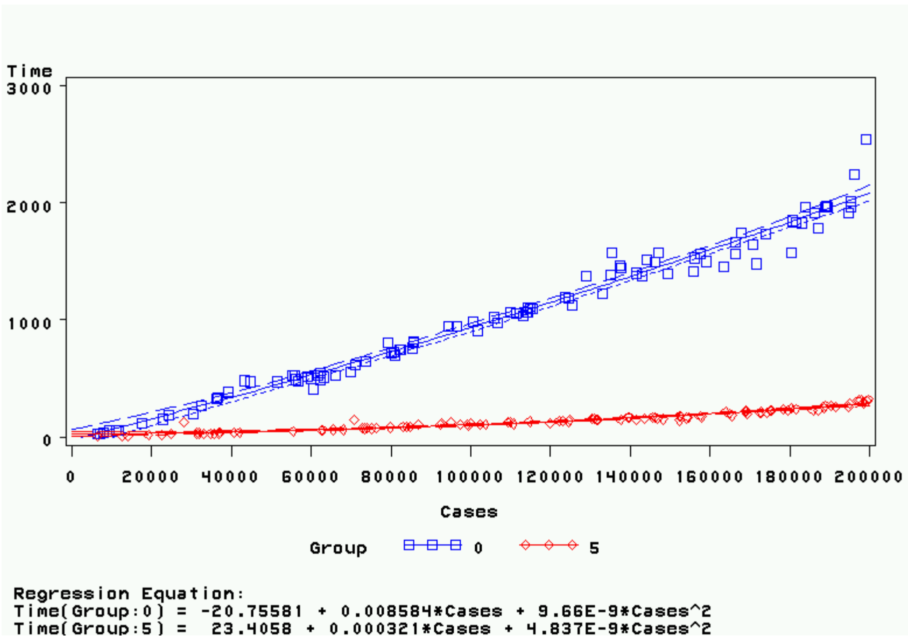

Fig. 5. Group 0 - standard EA, Group 5 five-worker SMS-PEA

[3] SAS is a trademark of SAS Institute Inc.

SMS-PEA Figure 6 compares the SMS-PEA approach (utilising five, ten and twenty workers) with the standard EA approach for the interval [200 ... 6000]. The five-worker SMS-PEA outperforms the standard EA when more than 2500 cases are used for the fitness evaluation. The ten-worker SMS-PEA outperforms the standard EA for more than 3500 cases and the twenty-worker SMS-PEA for more than 4750 cases. Costs, such as communication costs, prevent an increase in performance below this threshold. It can also be seen that more than five workers do not result in any performance increase for the interval [200...6000]. It appears as if the existence of more workers is responsible for the additional costs.

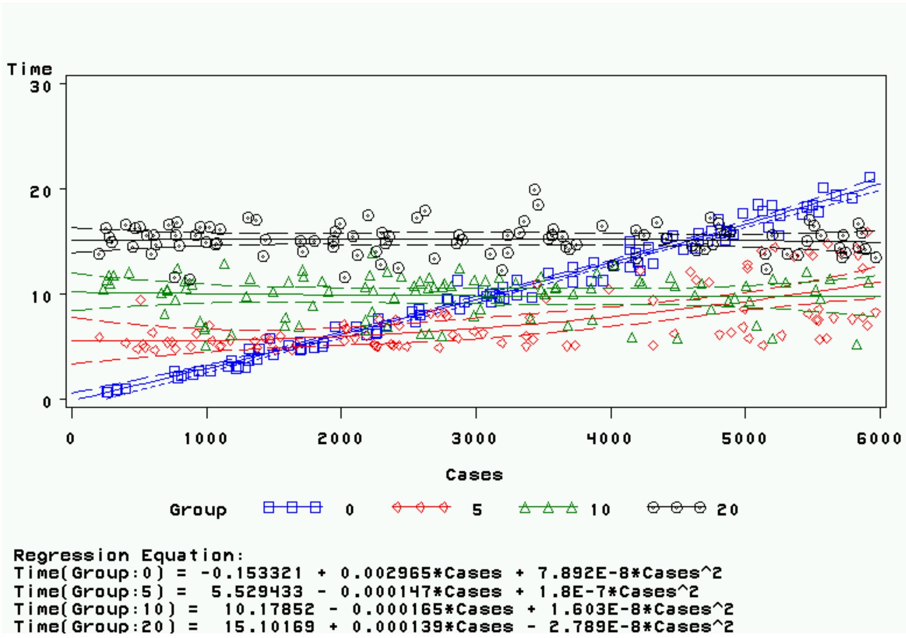

Fig. 6. Group 0: standard EA, Group 5: five-worker SMS-PEA, Group 10: ten-worker SMS-PEA, Group 20: twenty-worker SMS-PEA

Figure 7 compares three SMS-PEA experiments, utilising five, ten and twenty workers for the interval [6000 ... 200000]. It indicates that the ten-worker SMS-PEA outperforms the five-worker SMS-PEA for more than 35000 cases. The ten-worker SMS-PEA is outperformed by the twenty-worker SMS-PEA for more than 105000 cases.

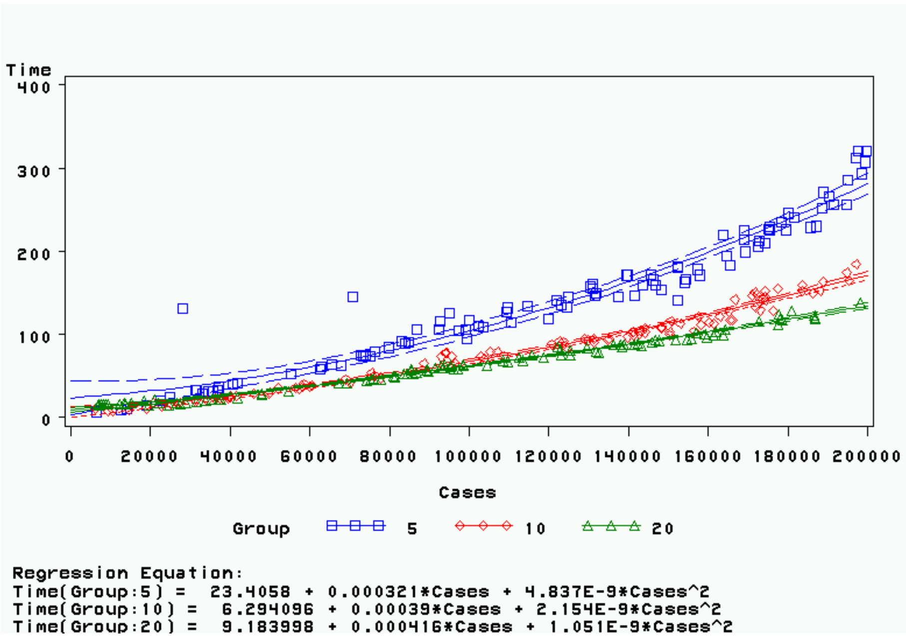

Fig. 7. Group 5: five-worker SMS-PEA, Group 10: ten-worker SMS-PEA, Group 20: twenty-worker SMS-PEA

AMS-PEA Figure 8 compares the ASM-PEA approach (utilising five, ten and twenty workers) with the standard EA approach for the interval [200 ... 6000]. The five-worker AMS-PEA outperforms the standard EA for more than 2250 cases. The ten-worker AMS-PEA outperforms the standard EA for more than 2750 cases and the twenty-worker AMS-PEA for more than 3750 cases. As for the SMS-PEA, it can be observed that more than five workers do not result in any performance increase for the interval [200...6000].

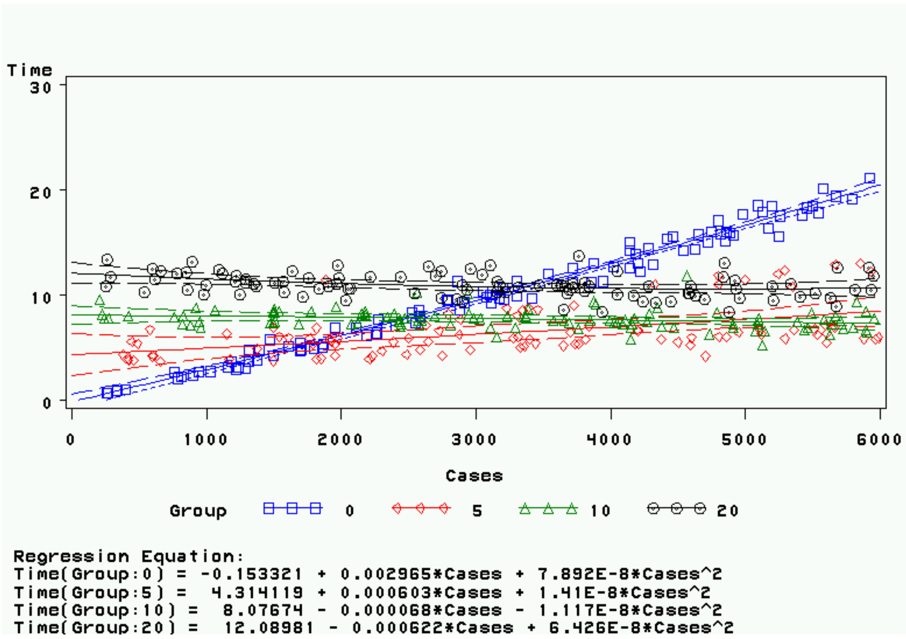

Fig. 8. Group 0: standard EA, Group 5: five-worker AMS-PEA, Group 10: ten-worker AMS-PEA, Group 20: twenty-worker AMS-PEA

Figure 9 compares three AMS-PEA experiments, utilising five, ten and twenty workers for the interval [6000 ... 200000]. The ten-worker AMS-PEA outperforms the five-worker AMS-PEA for more than 25000 cases. The twenty-worker AMS-PEA outperforms the ten-worker AMS-PEA for more than 100000 cases. Figure 9 also indicates that the ASM-PEA approach exploits the larger number of workers better than the SMS-PEA because there is a larger area between the five-worker ASM-PEA and the ten-worker ASM-PEA. The slopes of the fitted curves also make a flatter impression.

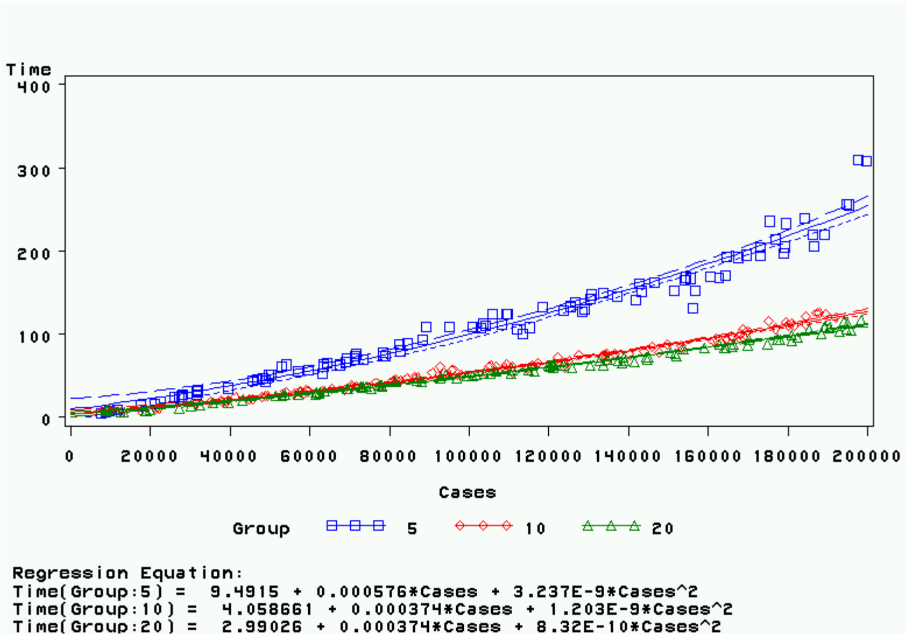

Fig. 9. Group 5: five-worker AMS-PEA, Group 10: ten-worker AMS-PEA, Group 20: twenty-worker AMS-PEA

CG-PEA Figure 10 compares the CG-PEA approach (utilising five, ten and twenty workers) with the standard EA approach for the interval [200 ... 6000]. It indicates that the five-worker CG-PEA outperforms the standard EA for more than 750 cases. The ten-worker CG-PEA outperforms the standard EA for more than 1000 cases and the twenty-worker CG-PEA for more than 1500 cases. It can clearly be observed that the CG-PEA outperforms the standard EA much earlier than the SMS-PEA and the ASM-PEA approach. This is due to the fact that the CG-PEA communicates less often with the space than the other PEAs. However, the existence of more than five workers still does not result in a performance increase for the interval [200...6000].

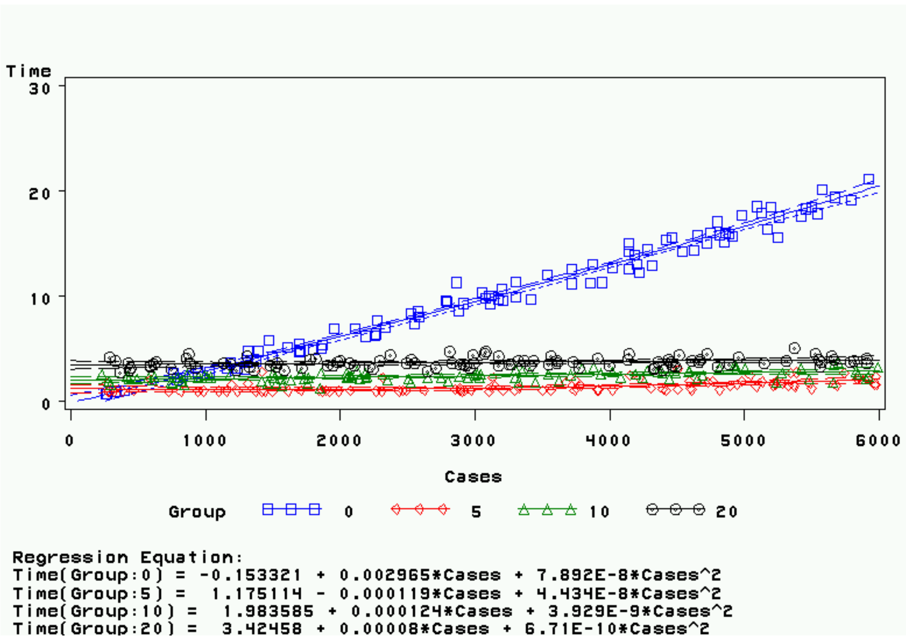

Fig. 10. Group 0: standard EA, Group 5: five-worker CG-PEA, Group 10: ten-worker CG-PEA, Group 20: twenty-worker CG-PEA

Figure 11 compares three CG-PEA experiments, utilising five, ten and twenty workers for the interval [6000 ... 200000]. The ten-worker CG-PEA outperforms the five-worker CG-PEA for more than 65000 cases and the twenty-worker CG-PEA outperforms the ten-worker CG-PEA after approximately 70000 cases.

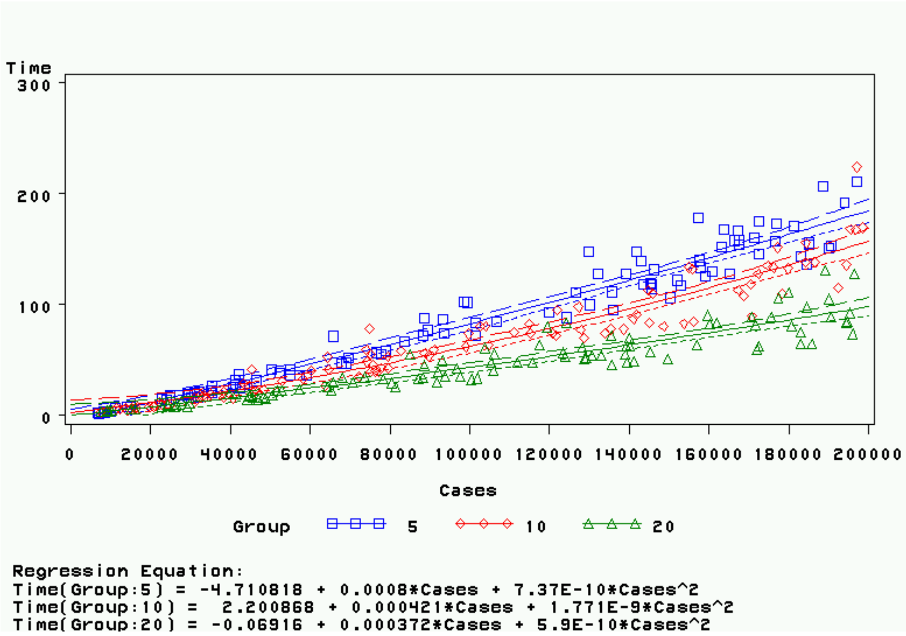

Fig. 11. Group 5: five-worker CG-PEA, Group 10: ten-worker CG-PEA, Group 20: twenty-worker CG-PEA

Comparisons between the PEAs We will now compare all three PEA implementations for the interval [6000...200000]. Figure 12 compares the three five-worker PEAs. The CG-PEA outperforms the other PEAs when the number of cases was greater than 80000. The ASM-PEA outperforms the SMS-PEA after approximately 160000 cases.

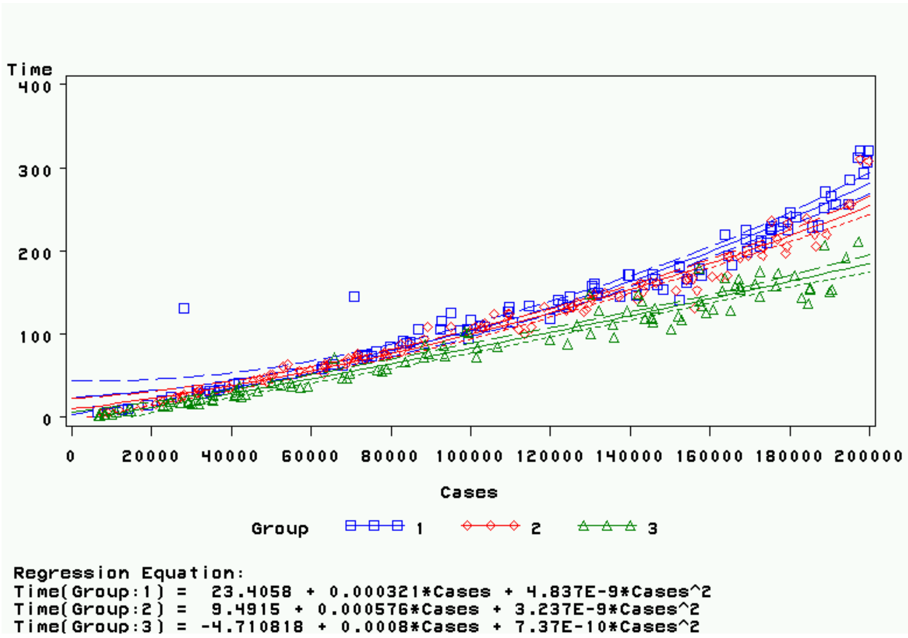

Fig. 12. Group 1 - five-worker SMS-PEA, Group 2 - five-worker AMS-PEA, Group 3 - five-worker CG-PEA

Figure 13 compares the three ten-worker PEAs. It is interesting to notice that there seems to be no significant difference between the CG-PEA and the AMS-PEA. The SMS-PEA is outperformed by the AMS-PEA and the CG-PEA for more than 60000 cases.

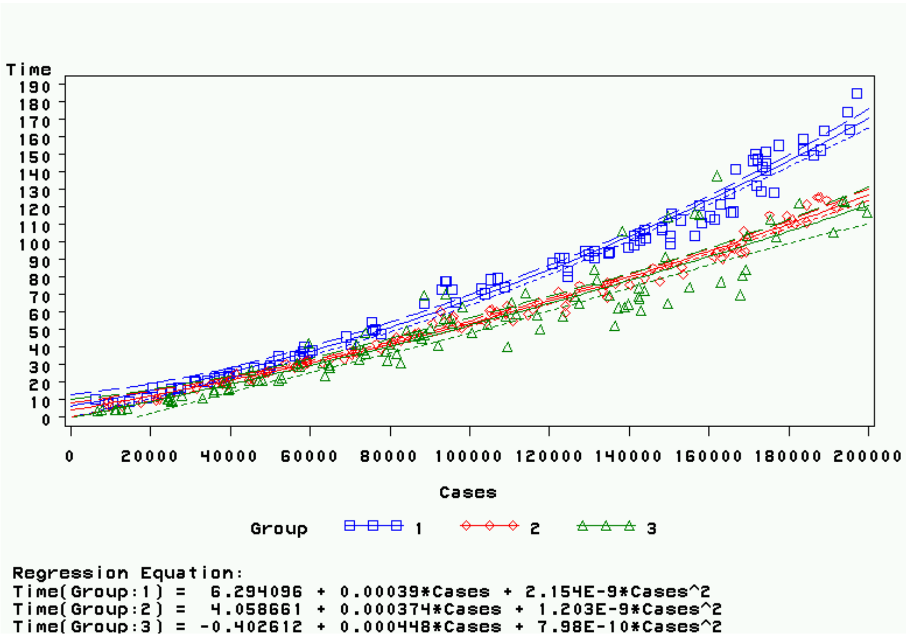

Fig. 13. Group 1 - ten-worker SMS-PEA, Group 2 - ten-worker AMS-PEA, Group 3 - ten-worker CG-PEA

Figure 14 compares the three twenty-worker PEAs.

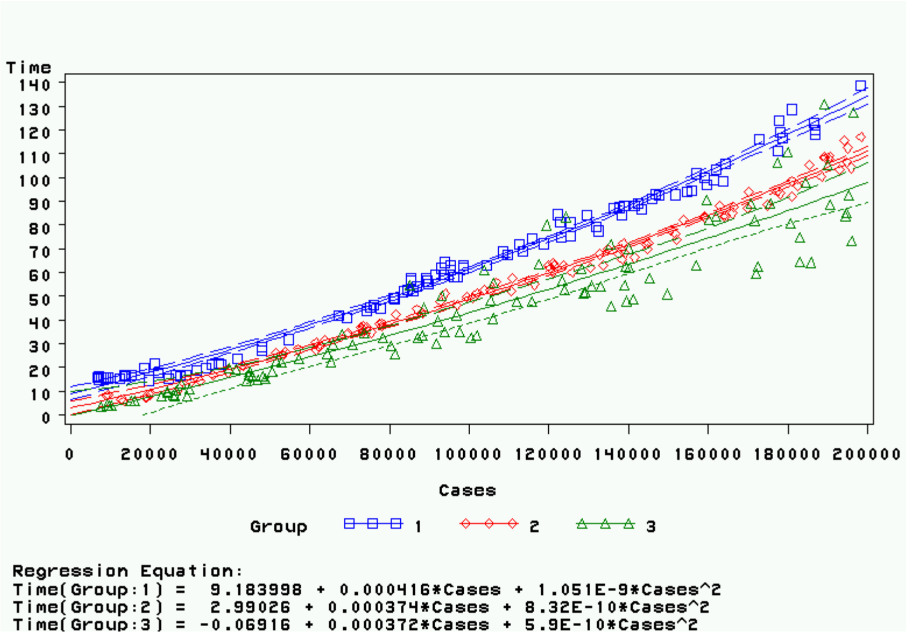

Fig. 14. Group 1 - twenty-worker SMS-PEA, Group 2 - twenty -worker AMS-PEA, Group 3 - twenty -worker CG-PEA

The AMS-PEA and the CG-PEA outperform the SMS-PEA for more than 15000 cases. The CG-PEA outperforms the AMS-PEA for more than 120000 cases.

7 Conclusions

JavaSpaces allows the implementation of different PEA paradigms that outperform the standard local EA for complicated problems. This is especially true for the CG-PEA, which can be declared as the 'winner' of the three PEAs investigated. However no PEA outperforms the standard EA for very small problems. This is mainly due to communication costs. It has also been observed that the existence of more workers does not necessarily result in a performance increase for relatively small numbers of cases. This problem has also been discussed in [16].

We have shown that worker and master implementations can be executed on different computer platforms in a multi-user environment. Hence it is possible to harvest the power of heterogeneous multi-user computer networks utilising this technology. We also believe that this technology allows the implementation of highly reusable PEA paradigms. The workers of the AMS-PEA and SMS-PEA can evaluate different types of individuals (see section 4.1). Several researchers may run different PEAs and could use the space (for the fitness evaluation) without any exchange or shutdown of the workers. It is also possible to reuse the same PEA implementation for different problems by just exchanging the individuals and the corresponding fitness evaluation. The workers and processes such as genetic operators and selection are able to deal with all types of individuals that implement a particular interface (see section 4.1).

8 Further Work

Although the CG-PEA is capable of evolving the same amount of individuals in a shorter period of time, in comparison to the other PEAs, it remains an open question whether or not these individuals perform as well as the individuals that were evolved by the other PEAs. Future work will investigate this issue. We have tested our approaches in a heterogeneous multi-user network. In the future we will test the approaches on a cluster of computers that is solely dedicated to our particular problem. Further performance increases are expected.

We believe that the JavaSpaces technology allows the implementation of many more PEA paradigms. The CG-PEA, for example, can be made finer grained (resulting in a fine grained PEA or FG-PEA) by deploying of a larger number of workers. The CG-PEA also does not have any topology that determines the connectivity between the workers. Individuals are migrated to other demes (workers) with a uniform probability. This simplifies the configuration of the CG-PEA but may prevent a more efficient exploitation of the deployed machines and a more efficient exploration of the search space defined by the problem. The utilisation of several spaces, as suggested in [16] [17], may exploit larger numbers of workers more efficiently. We also need to make our implementations more robust using Transactions [16].

References

1. Panagiotis Adamidis. Parallel evolutionary algorithms: A review, 1998. HERCMA'98, 4th Hellenic-European Conference on Computer Mathematics and its Applications.
2. Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. Applications of evolutionary algorithms. Technical Report SYS-2/92, 1993. available at: citeseer.nj.nec.com/hoffmeister92genetic.html.
3. Thomas Bck. *Evolutionary algorithms in theory and practice : evolution strategies, evolutionary programming, genetic algorithms*. New York : Oxford University Press, 1996.
4. Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in genetic algorithms. Technical Report 11, 1995. available at: http://citeseer.nj.nec.com/blickle95comparison.html.
5. E. Cant'u-Paz. Topologies, migration rates, and multi-population parallel genetic algorithms, 1999. available at: citeseer.nj.nec.com/article/cantu-paz99topologies.html.
6. Erick Cantu-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141–171, 1998. available at: http://citeseer.nj.nec.com/155991.html.
7. Oscar Cordn, Francisco Herrera, Frank Hoffmann, and Luis Magdalena. *GENETIC FUZZY SYSTEMS*. World Scientific, 2001.

8. O. Cordon, M.J. del Jesus, and F. Herrera. Evolutionary approaches to the learning of fuzzy rule-based classification systems, 1999. in: Evolution of Enginnering and Information Systems and Their Applications. L.C. Jain (Ed.), CRC Press, 1999, pp. 107-160.
9. Charles Robert Darwin. *On the origin of species by means of natural selection*. Murray, London, 1860.
10. Ken Arnold Eric Freeman, Susanne Hupfer. *JavaSpaces(TM) Principles, Patterns and Practice (The Jini(TM) Technology Series)*. Addison Wesley Publishing Company, 1999.
11. Kathleen McNiff Esmond Pitt. *Java.RMI*. Addison Wesley, 2001.
12. Robert Flenner. *Jini and JavaSpaces Application Development*. Sams, 2001.
13. David B. Fogel. *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*. Wiley-IEEE Press, 1999.
14. D.B. Fogel. *Evolutionary Computation: The Fossil Record*. IEEE Press, 1998.
15. Eric Freeman and Susanne Hupfer. Make room for javaspaces, part 1, 1999. available at: http://www.javaworld.com/jw-11-1999/jw-11-jiniology.html.
16. Eric Freeman and Susanne Hupfer. Make room for javaspaces, part 4, 2000. available at: http://www.javaworld.com/jw-04-2000/jw-0421-jiniology.html.
17. Eric Freeman and Susanne Hupfer. Make room for javaspaces, part 5, 2000. available at: http://www.javaworld.com/jw-06-2000/jw-0623-jiniology.html.
18. D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.
19. David Edward Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.
20. A.D. Gordon. *Classification*. Chapman and Hall, 1981.
21. J.J. Grefenstette. Parallel adaptive algorithms for function optimization. Technical Report CS-81-19, 1981. Vanderbilt University, Computer Science Department, Nashville, TNavailable at: http://citeseer.nj.nec.com/blickle95comparison.html.
22. John H. Holland. *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press, 1992.
23. John Hunt. *Java and Object Orientation*. Springer, 1998.
24. Sun Microsystems Inc. Javaspaces service specification. available at: http://java.sun.com/products/javaspaces/.
25. Ludmila I. Kuncheva. *Fuzzy classifier design*. Heidelberg : Physica-Verlag, 2000.
26. Sing Li. *Professional Jini*. Wrox, 2000.
27. Xavier Llor and Josep M. Garrell. Knowledge-independent data mining with fine-grained parallel evolutionary algorithms. In *Proceeding of the Genetic and Evolutionary Computation Conference (GECCO2001)*, 2001. available at: http://www.salleurl.edu/ xevil/Publications/papers.html.
28. Zbigniew MICHALEWICZ. *Genetic algorithms + data structures = evolution programs*. Springer, Berlin, 1996.
29. Zbigniew Michalewicz and David B. Fogel. *How to Solve it : Modern Heuristics*. Springer, 2000.
30. Sun Microsystems. Java distributed event specification, 1998. available at: www.javasoft.com/products/javaspaces/specs/ev.pdf.
31. Mariusz Nowostawski and Riccardo Poli. Parallel genetic algorithm taxonomy. In *Proceedings of the Third nternational conference on knowledge-based intelligent information engineering systems (KES'99)*, pages 88–92. IEEE Press, 1999. available at: http://cswww.essex.ac.uk/staff/poli/papers/publications.html.
32. I. RECHENBERG. *Evolutionsstrategie'94*. Frommann-Holzboog, 1994.
33. Hans-Paul Schwefel. *Evolution and Optimum Seeking*. John Wiley, 1995.

34. W. M. Spears, K. A. De Jong, T. Bäck, D. B. Fogel, and H. de Garis. An overview of evolutionary computation. In Pavel B. Brazdil, editor, *Proceedings of the European Conference on Machine Learning (ECML-93)*, volume 667, pages 442–459, Vienna, Austria, 1993. Springer Verlag. available at: citeseer.nj.nec.com/spears93overview.html.
35. Ingo Wegener. On the design and analysis of evolutionary algorithms. In *Proceedings of Workshop on Algorithm Engineering as a New Paradigm*, pages 36–47, 2000. available at: citeseer.nj.nec.com/wegener00design.html.

9 Acknowledgements

We would like to thank Dr. Peter McBurney and Dr. Costas Vlachos and the technical staff of the Computer Science Department of the University of Liverpool for their valuable comments and help. Our thanks also go to the technical support of SAS Institute Inc.; especially to James Gillian and the many people from the JavaSpaces mailing list who have answered several questions.