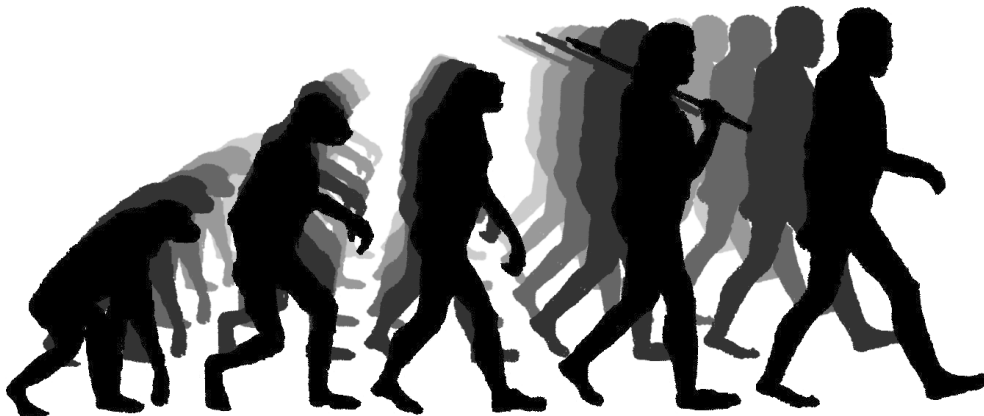


Semester Thesis

Parallelization of Evolutionary Algorithms



Author:

Samuel Welten

*Tutors:*Johannes Bader
Dimo Brockhoff

Preface

Fifteen weeks ago I began to engage in the principles of evolutionary computation. The acquirement of the required knowledge to do the investigations and to write this thesis was a demanding but valuable experience. The field of evolutionary algorithms is fascinating and the possibilities they offer are nearly boundless. Towards the end of the time reserved for this thesis, I had to detain myself not to do more interesting experiments, in order to have enough time for writing.

I would like to thank my two tutors Johannes Bader and Dimo Brockhoff for their competent help in the literature research, the assistance in writing this thesis and the clarifying discussions.

Zurich, June 5 2007

Samuel Welten

Contents

Abstract	ix
1 Motivation	1
2 Basic Definitions	3
2.1 Evolutionary Algorithms	3
2.2 Multi-Objective Optimization Problems	4
2.3 The Hypervolume Indicator	5
3 Parallel Evolutionary Algorithms	7
3.1 Master-Slave	7
3.2 Island Models	8
3.2.1 Migration Topology:	9
3.2.2 Migration Strategy:	9
3.2.3 Migration Interval:	10
3.2.4 Migration Size:	10
3.3 Fine Grained Parallel Evolutionary Algorithms	10
3.4 Hybrid Parallel Evolutionary Algorithms	11
3.5 Parallel Multi-Objective Evolutionary Algorithms	11
3.5.1 Migration in Parallel Multi-Objective Evolutionary Algorithms	11
3.5.2 Hypervolume Based Migration Strategy	12
4 Adapting the PISA Framework	15
4.1 Structural changes to the protocol	16
4.2 The Controller Module	16
4.2.1 Create the Needed Directories and PISA Files	17
4.2.2 Start the Islands Using Condor or SSH	17
4.2.3 Migrating the Individuals During the Run	17
4.2.4 Collecting the Results	19
5 Experiments	23
5.1 Settings	23
5.2 Migration Topology	25
5.3 Migration Strategy	25

5.4	Number of Islands	26
5.5	Migration Size	27
5.6	Development Over Time	28
5.7	Application to a Real World Problem	29
5.8	Superlinear Speedups	31
6	Conclusion	37
6.1	Future Work	38
A	List of Abbreviations	43
B	Parallelizing an Existing PISA Module	45
B.1	Necessary Changes to the Variator	45
B.2	Necessary Changes to the Selector	47
C	Using the Controller	49
C.1	Compiling and Running the Controller	49
C.2	The Controller Configuration File	49

List of Figures

2.1	A state graph displaying the sequence of an evolutionary algorithm.	4
2.2	The hypervolume (grey) of a two objective minimization problem with the individuals (blue dots) and the reference point (red dot).	5
3.1	Different common island topologies: a) ring, b) torus, c) random	9
3.2	a) clustering based assigning to island (cluster centers in red), b) cone separation based assigning to islands	12
3.3	The hypervolume of three individuals and the particular contributions	13
3.4	An example of a hypervolume based migration	13
4.1	Part 1 of the state graph of the extended PISA protocol	20
4.2	Part 2 of the state graph of the extended PISA protocol	21
5.1	The effect of different topologies on the hypervolume of the population	25
5.2	The effect of different migration selection and replacement strategies on the hypervolume of the population, including the performance of a serial EA	27
5.3	How the number of islands influences the migration, if the islands are arranged in a torus topology. The numbers on the arrows indicate how man individuals are migrated in each direction.	28
5.4	The effect of different number of islands on the hypervolume of the population	29
5.5	The effect of increasing the migration size without the proposed adaption on the hypervolume of the population	30
5.6	The effect of increasing the maximal migration size on the hypervolume of the population, using the hypervolume based adaption of the migration size	31
5.7	A comparison of the development of the hypervolume over 200 generations between the panmictic model and a 4 island model	32
5.8	The population of the island model nearly completely dominates the panmictic population after the same number of generations	33
5.9	A comparison of the execution time and the hypervolume of a serial EA and an island model, using the HypE selector.	34

- 5.10 A comparison of the execution time and the hypervolume of a serial EA and an island model, using the IBEA selector. 34
- 5.11 A comparison of the execution time and the hypervolume of a serial EA and an island model, using the HypE selector in sampling mode. 35

List of Tables

5.1	The different island models used for the experiments	28
5.2	The configurations for the two different runs of the wireless sensor network evolutionary algorithm	30
5.3	The configurations for the time measurement experiment	33
B.1	The structure of the <i>population</i> file	46
C.1	Parameter names and values of the controller configuration file . . .	50

Abstract

The following work investigates the effects and implications of parallelizing multi-objective evolutionary algorithms. In the scope of this work an island model was implemented and a new hypervolume based migration strategy for multi-objective island models is proposed. Island models using this migration strategy led to significantly better results than accordant serial evolutionary algorithms in the same number of generations and with the same total population size. Additionally, the migration size can be adapted automatically using this method. Under certain circumstances the island model running just on one single-core processor is faster than the normal evolutionary algorithm.

Die folgende Arbeit untersucht die Auswirkungen und Folgen der Parallelisierung Evolutionärer Algorithmen für die Mehrzieloptimierung. Im Rahmen dieser Arbeit wurde ein Inselmodell implementiert und eine neue, Hypervolumen-basierte Migrationsstrategie für die Mehrzieloptimierung mit Inselmodellen wird eingeführt. Inselmodelle mit dieser Migrationsstrategie führten zu signifikant besseren Resultaten als die entsprechenden seriellen Evolutionären Algorithmen in der gleichen Anzahl Generationen und derselben Gesamtpopulationsgrösse. Zusätzlich kann mit dieser Methode auch die Migrationsgrösse automatisch angepasst werden. Unter gewissen Umständen ist das Inselmodell sogar mit nur einem single-core Prozessor schneller als der normale Evolutionäre Algorithmus.

Chapter 1

Motivation

In the sixties several scientists around the world began independently of each other to do research on algorithms which are inspired by the biological evolution. These algorithms attracted a lot interest because they offer a black box approach to solve a broad variety of optimization problems without explicit knowledge of the problem. The success of these evolutionary algorithms (EA) grew with the computer industry, because more powerful computers allowed to run more complex EAs.

Nowadays, evolutionary algorithms are used for all types of different optimization problems. Be it to create plans of how to charge a container ship [1] or for the protein structure prediction in chemistry [2]. These highly complex and mostly multi-objective problems need a lot of computation power. The availability of multi-core processors and big computer clusters led to the development of parallel evolutionary algorithms which can run concurrently on several processors or cores. For single-objective optimization problems this was achieved fast but for the more complex and therewith more computation intensive multi-objective problems, an efficient parallelization is still an issue.

This thesis investigates the parallelization of multi-objective evolutionary algorithms on the example of island models. A general framework for EAs is extended to run island models in the scope of this work. Additionally a new migration strategy, which should make parallel multi-objective evolutionary algorithms more efficient, and a method to adapt the migration size automatically, are proposed.

The structure of the thesis is as follows: chapter 2 introduces the concepts of evolutionary algorithms, multi-objective optimization problems and the hypervolume indicator. It is followed by a chapter which describes the different possibilities to parallelize evolutionary algorithms. Subsequent to this, the implementation of an island model in an existing EA-framework is documented. This implementation is used to study the effects of the parallelization and especially of the proposed migration strategy. These experiments and their results are described in chapter 5.

Chapter 2

Basic Definitions

2.1 Evolutionary Algorithms

Biological evolution consists of three main principles: selection, recombination and mutation. The natural selection, also known as survival of the fittest, makes sure that individuals which are inefficient, weak or not well adapted to the environment don't survive. Recombination is the process of combining the genotypes of individuals and creating a new one. The third principle of evolution, the mutation, is the randomness, which is the reason the offspring is not an exact recombination of the parents but with a certain possibility parts of the genotype are changed randomly. These principles allowed the development of a big variety of creatures which adapt their appearance and survival strategies better to the environment in every generation.

The simplicity of the principles of biological evolution have led to the desire of imitating this method and using it for general optimization problems. In order to apply it to general mathematically formulated problems, a few changes have to be done (see Figure 2.1). A population consists not of creatures but of solutions to the problem one wants to solve with the evolutionary algorithm. The genotype of these individuals somehow represents the solutions of the problem as a vector of properties of the individual (decision vector). The set of all possible solution genotypes is commonly called the search-space. To perform the selection, a criterion has to be defined, which states how strong and therewith survivable an individual is. In the simplest case this is one objective function which assigns each individual a real-valued fitness, based on its decision vector. The selection is split up in two parts: the mating selection is accountable for the choice of the parents of the next generation and the environmental selection determines the set of individuals which survive, based on the fitness value.

Using an evolutionary algorithm, many optimization problems can be solved without explicit knowledge of the problems structure. However, the more is known about the problem, the more specific and efficient the individual representation and the recombination operation can be done and therewith, the evolutionary algorithm converges faster to the optimum.

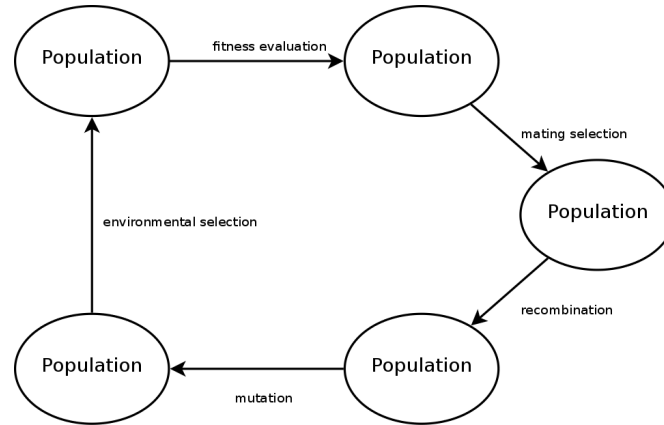


Figure 2.1: A state graph displaying the sequence of an evolutionary algorithm.

2.2 Multi-Objective Optimization Problems

The above description of the selection method is bounded to the case where only one objective function has to be optimized. Though in most real world problems, several objectives have to be optimized at the same time. These problems are called multi-objective optimization problems (MOOP). For example the design of a CPU is aimed at producing a processor which is as fast as possible, as small as possible and as cheap as possible. One can easily see it is impossible to optimize all objectives at the same time. The fastest processor design will not be the same as the cheapest processor design. Trade-offs between the objectives are common for multi-objective optimization problems. This leads to the problem that in a general case an individual is better in one objective than another individual but is worse in a different objective and the individuals cannot be compared in the same way as in the single-objective optimization problems. That is why in multi-objective problems not only one optimal solution is searched but a set of solutions, which are the best trade-offs between the objectives. An individual which is superior in all objectives than another individual dominates this individual. The set of individuals with the best trade-offs is called the Pareto front. An individual belongs to the Pareto front if it is not dominated by another individual.

The optimization of a multi-objective problem can also be considered as the search for the optimal Pareto front.

2.3 The Hypervolume Indicator

In the general case, two populations of multi-objective evolutionary algorithms cannot easily be compared, as some individuals of one population may dominate individuals of the other population and vice versa. But especially when the efficiency of an evolutionary algorithm should be improved, a measurement method for multi-objective populations becomes indispensable. A generic indicator to compare pop-

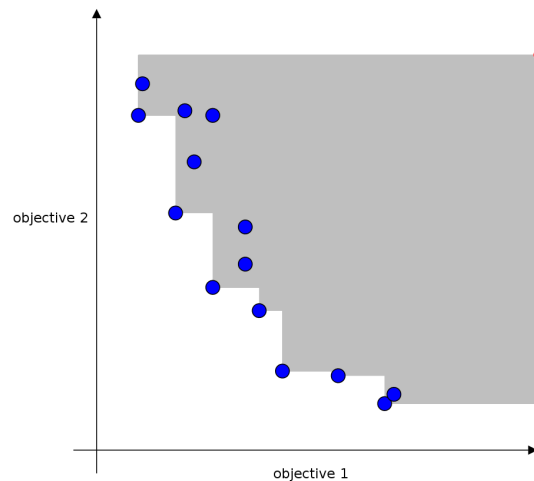


Figure 2.2: The hypervolume (grey) of a two objective minimization problem with the individuals (blue dots) and the reference point (red dot).

ulations, called hypervolume indicator, was proposed by Eckart Zitzler and Lothar Thiele and is used very often nowadays [3]. The hypervolume indicator measures the volume between the Pareto front of a population and an arbitrary reference point. An example of a hypervolume in a two-objective minimization problem is depicted in Figure 2.2. One of the appreciated properties of the hypervolume indicator is, that the search for the optimal Pareto front is equal to the maximization of the hypervolume [4].

Chapter 3

Parallel Evolutionary Algorithms

The increasing complexity of problems solved with evolutionary algorithms (EA) and the availability of big computer clusters and multiprocessor systems lead to a growing importance of parallel evolutionary algorithms (PEA). Two aspects of PEAs are of special interest: on the one hand they are faster than serial EAs in terms of execution time and on the other hand they can be more resistant against getting stuck in local optima, which can cause a faster convergence to the global optimum for multimodal problems [5].

Parallel evolutionary algorithms can be classified in four different types: single population master-slave, multiple populations, fine grained and hierarchical combinations [6, 7]. These approaches to parallelization are very different in terms of hardware requirements and efficiency. The most important difference to non-parallel EAs is that, except for the master-slave approach, the population is split up in smaller subpopulations, also called demes.

3.1 Master-Slave

As in most evolutionary algorithms the fitness evaluation of the individuals is the most computation intensive part, it is self-evident to distribute this operation to different processors. The master works on the same population as the serial EA would, but delegates the fitness evaluation of an individual to another processor. After this evaluation, the fitness value is written back to a shared memory, or is communicated through message passing, which allows the master to read the values. The environmental selection and the mating selection is done by the master process.

Master-slaves models can be implemented in two different ways: synchronous and asynchronous. In the synchronous model the master delegates the fitness evaluations to the slave processors and does this until the fitness of every individual of the population has been computed. Thereafter, it performs the selection and the variation. By contrast, the master of an asynchronous master-slave model delegates

the fitness computations and doesn't wait for the whole population to be evaluated. The newly evaluated individuals are directly added to the population and undergo selection and variation even if not the entire population is evaluated. The results of a synchronous master-slave parallelization don't differ from the results of a serial EA, because the performed algorithm is the same [6]. Unlike to the synchronous model, the asynchronous model produces different results, due to the algorithmic changes. In the optimal case the expected runtime decays nearly linearly with the number of deployed processors. In a real world application the speedup will be smaller because the selection process is not parallelized and there is communication overhead.

3.2 Island Models

One can observe in nature that the overall diversity is maintained through isolation, meaning that mating and environmental selection happen separately for each subpopulation. In nature this isolation is due to geographical constraints like valleys and islands, which delimit the mobility of the individuals. As this concept of several small subpopulations (also called demes), helped the biota to develop, it has been ported to evolutionary algorithms, where it is called island models (IM). Island models split up the population and the isolation of the demes is achieved by running a separate EA on each subpopulation. Parallelization comes in, because this different islands can easily be run on several computers at the same time. Though as complete isolation of the islands would be disadvantageous, from time to time individuals have to be exchanged between the subpopulations. This step is called migration. A well balanced migration lets the information of good individuals pass among the islands but helps also to preserve diversity by isolation of the different islands. In comparison to normal EAs, this differentiation of the subpopulations can be advantageous over the case with just one big population (panmictic EA), because the problems search space is explored more evenly and the overall diversity helps to fight population stagnation [8]. Additionally there have been reports that island models could reduce the overall computational effort in comparison to panmictic EAs [5]. The causes for these astonishing observations are not fully understood. However, as island models are not a simple parallelization of the normal evolutionary algorithm computations, but change how the algorithm works, one can expect qualitatively different results from an island model than from a panmictic EA. One explanation for these speedups is an additional selection pressure, which can be introduced by the migration, leading to a faster convergence (see chapter 5).

Migrating individuals in a island model is a highly non trivial action. Several parameters like numbers of individuals to migrate, selection and replacement strategy or island topology have a significant influence on the convergence speed and the selection pressure [5, 9]. The different issues concerning the migration of individuals between the islands are described in the next four subsections.

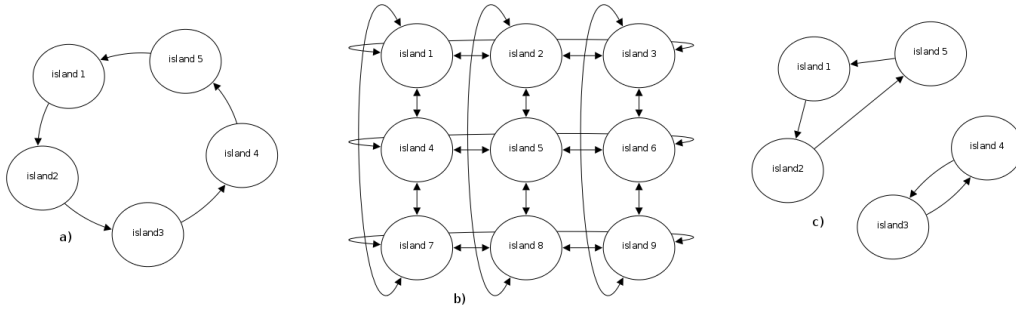


Figure 3.1: Different common island topologies: a) ring, b) torus, c) random

3.2.1 Migration Topology:

The topology in which islands are arranged is an important parameter, as the speed at which information travels through the network of islands depends on it [8]. The most common topologies described in scientific papers are: ring, random, torus and fully connected. A topology can be represented by a directed graph, where each node stands for an island and an arrow is a migration of individuals in this direction. Figure 3.1 depicts three graphs of common topologies. The diameter of a graph and therewith of a topology is the maximum distance (minimal path length) of any two nodes. Ring, random and torus all have different diameters, leading to variable information diffusion speed. The diameters of the topologies described below are depending on the number of islands n .

- In the ring topology each island receives individuals from the preceding island and gives individuals to the succeeding island. Its diameter is $O(n)$.
- A topology with a smaller diameter, $O(\sqrt{n})$, is the torus, meaning the islands are placed on a grid and each island exchanges individuals with the island in the north, south, west and east. This grid is folded into a torus, by connecting the upper and lower edge and the left and right edge of the grid.
- The next common topology, with diameter $O(\log(n))$, is the random topology. Each time a migration takes place, for each island a migration partner is chosen uniformly at random among the other islands.
- In a fully connected topology, as the name suggests, every island exchanges individuals with all the other islands. It is of diameter $O(1)$, which corresponds to the fastest possible information diffusion speed between the islands.

3.2.2 Migration Strategy:

A migration strategy consists of two parts. The first part is the selection of individuals, which shall be migrated to another island. The second part is to choose which individuals are replaced by the newly obtained individuals.

The most common strategies to select individuals for exportation are to choose the best individuals or to choose them randomly. Analog to the selection, there are two replacement strategies: to replace the worst individuals or just random individuals [5]. In the case of a one dimensional optimization problem the ranking of individuals can easily be done, for example according to their fitness. Using a defined ranking method, these four migration strategies are common:

- Select the best individuals, replace the worst individuals.
- Select random individuals, replace the worst individuals.
- Select the best individuals, replace random individuals.
- Select random individuals, replace random individuals.

3.2.3 Migration Interval:

In order to distribute information about good individuals among the islands, migration has to take place. This can either be done in a synchronous way every n^{th} generation or in an asynchronous way, meaning migration takes place at non-periodical times. In synchronous models the influence of the migration interval has been studied by several researchers [9, 5, 8]. It is commonly accepted that a more frequent migration leads to a higher selection pressure and therefore a faster convergence. But as always with a higher selection pressure comes the susceptibility to get stuck in local optima [8].

Another property which is influenced more or less by the migration interval is the overall execution time. Depending on the selection and replacement strategy, the subpopulation size and the migration size, it takes a significant time to perform the migration and therefore, a lower migration interval means a higher parallelization overhead.

3.2.4 Migration Size:

A further important factor is the number of individuals which are exchanged. Its influence has been studied, however, only for the single objective case [9, 5]. According to these studies the migration size has to be adapted to the size of a subpopulation of an island. When one migrates only a very small percentage, the influence of the exchange is negligible but if too much individuals are migrated, these new individuals take over the existing population, leading to a decrease of the global diversity.

3.3 Fine Grained Parallel Evolutionary Algorithms

The fine grained parallel Evolutionary Algorithm (FGPEA) is, like the island model, based on splitting up the population in several subpopulations. But unlike to island

models, where the population is split up into a few subpopulations, it is split up into many, very small demes. In the ideal case, a subpopulation consists of only one individual [10]. These small aggregations of individuals are commonly placed on a two or three dimensional grid. The environmental selection as well as the mating selection of a deme uses only the local neighboring subpopulations. In this way the selection pressure is reduced in comparison to the panmictic case and it takes a certain time, depending on the topology used, for the information to travel the network of demes. The selection pressure and the speed of information flow can be adjusted by changing the size of the neighborhood of a deme. This means, similar to the island model, the FGPEA does not just differ in execution time to the serial EA but is also different in an algorithmic sense.

3.4 Hybrid Parallel Evolutionary Algorithms

Hybrid parallel evolutionary algorithms are hierarchical combinations of the parallelization techniques mentioned above. For example Lin, Goodman and Punch used an island model, in which each island was a master-slave model, to compute the design of laminated composite beams [11].

Another, similar hybrid model was used for protein structure prediction. Several different islands evolve on a master machine, which distributes the fitness evaluations to slave computers [2].

3.5 Parallel Multi-Objective Evolutionary Algorithms

The broad possibilities of evolutionary algorithms and their high computational effort motivated a lot of research on parallelizing single objective EAs. However, the parallelization of multi-objective EAs received less attention. Because of the high availability of multi-processor systems, mostly master-slave models were used to solve multi-objective optimization problems.

For example a study of a parallel MOEA has been done by Coello and Sierra [12], who used a master-slave model with variable sized subpopulations on two test functions.

3.5.1 Migration in Parallel Multi-Objective Evolutionary Algorithms

A special problem concerning the parallelization of MOEAs with island models is the selection and replacement strategy of the migration. Opposed to the single objective case, where individuals can easily be ranked according to their fitness, there is no straight forward definition of better and inferior individuals for multi-objective problems.

The issue of how the individuals should be distributed among the island is an open research problem. Recent approaches let the islands focus on a part of

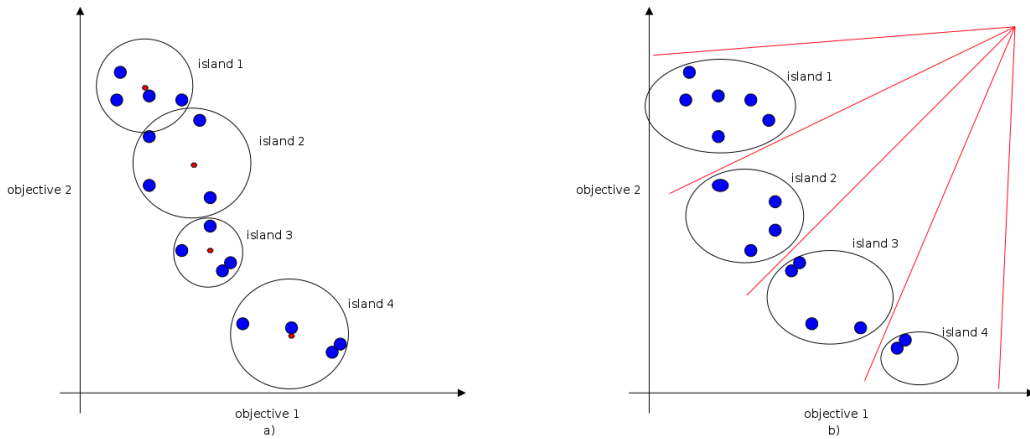


Figure 3.2: a) clustering based assigning to island (cluster centers in red), b) cone separation based assigning to islands

the objective space. This means individuals which are closer to each other in the objective-space are more likely to be on the same island. Streicher, Ulmer and Zell designed and implemented an island model and studied it on several test problems [13]. To assign the individuals to the islands, they used the k-Means clustering algorithm in the objective space and in the search space (see Figure 3.2 for a visualization). Their experiments showed that a clustering based island model performs not necessarily better than the standard island model. However, on one test problem they achieved significantly better results than the normal IM.

A geometric scheme to divide the objective-space was proposed by Jürgen Branke et al.. The so called cone separation uses a reference point and demarcation lines to decide which individuals are placed on which island, as depicted in Figure 3.2.

A drawback of these techniques is the open issue what to do if an island creates an individual, which is outside its part of the search space: Delete it or migrate it to another island?

In order to avoid this problem and to search in a different direction, a new approach to assign the individuals to the islands and to migrate them, is presented here.

3.5.2 Hypervolume Based Migration Strategy

A common goal for MOOPs is to search for an equally distributed population along the Pareto front. A good measure of how well a population is distributed in the objective-space is the hypervolume [3]. Therefore a hypervolume based migration strategy is proposed, which should allow every island to search the whole objective-space. The scheme of the hypervolume based migration is explained below.

The individuals of a population contribute more or less to the total hypervolume of the population. Figure 3.3 shows the hypervolume of three individuals in

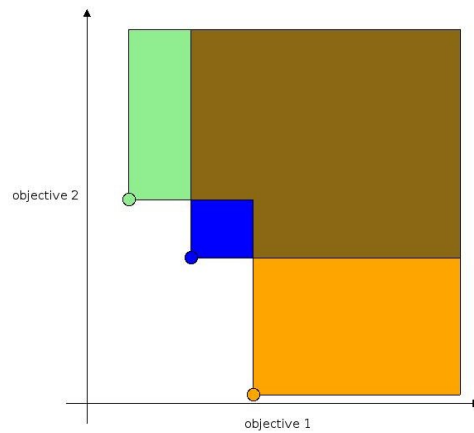


Figure 3.3: The hypervolume of three individuals and the particular contributions

a two dimensional minimization problem with an arbitrary reference point on the upper right. Areas which have the same color as the individual indicate the portion of the hypervolume an individual accounts for. To find the worst individuals of an island, which shall be replaced by the migrants, the entity contributing the least to the hypervolume have to be found. In Figure 3.3 this would be the blue individual. Analog to the *worst* individuals the *best* individuals can be defined. The *best* individuals, concerning the migration from one island to another, are the ones which increase the hypervolume of the receiving island the most, if the new individuals replace the *worst*. An example of a hypervolume based migration can be seen in

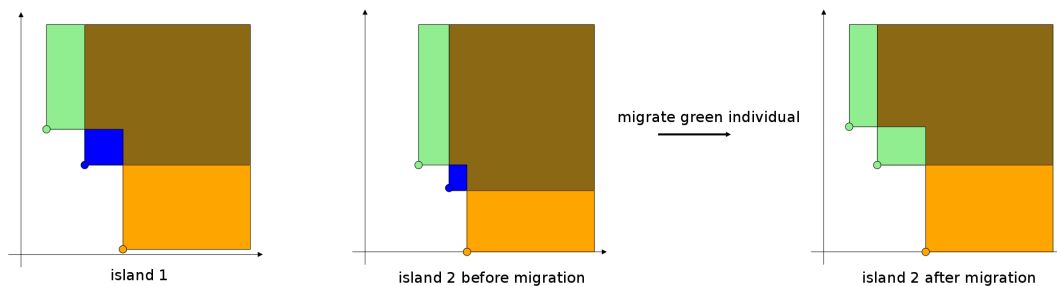


Figure 3.4: An example of a hypervolume based migration

Figure 3.4: First the *worst* individual of the receiving island (island 2) is computed, which is in this case the blue one. As a next step the possible migration candidates of island 1 are examined and one can see, that the green individual of island 1 would lead to the largest increase of the hypervolume if the blue one of island 2 has to be replaced.

Depending on the migration size, the hypervolume of an island is decreased by the migration operation. This leads to the idea, that one should migrate individuals only as long from one island to another, as the hypervolume of the receiving island

is increased by this operation. The effect of such a rule would be an automatic migration size adaptation. As the migration size is a difficult parameter to set in island models, this would be a useful simplification.

This method of automatic migration size adaptation was implemented and its effect tested in the scope of this thesis.

When using the contribution of an individual to the hypervolume as a ranking method, one has to think about that niching is emphasized in the objective-space, but not necessarily in the decision-variable-space [14]. If diversity in the decision-space is favored for a specific problem, another ranking method, emphasizing niching in the decision-space may be used.

Chapter 4

Adapting the PISA Framework

As multi-objective evolutionary algorithms often have parts which are problem independent, reusable modules for EAs would be a great help. This was the motivation for Stefan Bleuler et al. to specify an interface which separates the problem dependent from the problem independent part. In order to allow the reuse of modules on different platforms in combination with different programming languages, it was decided to be platform- an programming language independent. It is called "A Platform and Programming Language Independent Interface for Search Algorithms" (PISA) [15].

PISA separates the operations of an evolutionary algorithm in two parts: one is called the variator, which performs the variation and mutation operations, the other one is called the selector and performs the environmental- and mating selection. This allows to combine the problem dependent part (variator) with different well known selectors like SPEA2 or NSGA2 without having to reprogram them. To keep the communication platform independent, it is done through plain text files. An exact description of the protocol can be found in [15].

Out of the parallelization models presented in chapter 3, the island model seems to be the most promising, because it is flexible concerning the underlying computer infrastructure and the changes to the algorithm pledge not only time savings but also a faster convergence. In addition, the communication overhead is smaller than in the other models and, as communication is done over the file system in PISA, this variant will be faster than the others (fine grained and master-slave). These are the reasons why PISA was extended by the possibility to run island models and the extended protocol was implemented in the scope of this thesis.

4.1 Structural changes to the protocol

The logical and consistent method to control the two processes of a PISA run (variator and selector) is to insert a new program, which is responsible for the execution of the two modules and the migration of individuals from one island to another.

The introduction of this new program, called controller, requires adaptations of the PISA protocol. The changes to the protocol had to be made with these goals in mind:

1. The property of platform and programming language independence should be kept.
2. The changes to the existing modules should to be minimal and simple.
3. The protocol should allow a broad variety of island models.

In the existing PISA protocol the control flow is represented as a finite state graph, using the states 0 to 11. The synchronization of the processes is accomplished through these states. This method will also be used to implement the island model. The states and the transitions between them are shown in Figure 4.1 and Figure 4.2. The elements introduced for parallelization are marked red, the original PISA state graph is colored black.

Writing out the population to a file is only scheduled at the end of a run in the original PISA protocol. But the migration of individuals between two islands calls for an import/export functionality of the variator and the selector, which can be triggered from the outside. Therefore, the variator has to check at each generation in a file if the controller module requested a migration. In case of a migration, the variator writes out the whole population to a file and sets the new state 12, signaling the population was successfully exported to a file. From this moment on the variator pauses its execution and waits for the state file to change. Meanwhile the controller exchanges individuals between the islands, which are in state 12 according to a specified topology and migration strategy. When the controller completed the migration it sets the involved islands to state 13. This signals the variator to import the population from a file and write a new *initial population file* containing the indexes and objective values of the individuals. This file is read by the selector in state 14. After this, the migration is completed and the evolutionary algorithm can continue normally with state 2.

4.2 The Controller Module

In the following section the design of the controller module, which implements the protocol changes from the previous section, is described. The task of the controller is to offer the selector and variator an environment which is nearly the same as in

the panmictic case in order to keep the amount of necessary changes to the program codes of these modules at a low level. To achieve this, the procedure is subdivided into the following steps:

1. Create the needed directories and PISA files.
2. Start the islands using Condor [16] or SSH.
3. Migrate the individuals during the run.
4. Collect the results when all islands have finished.

These steps are described in detail in the following subsections.

4.2.1 Create the Needed Directories and PISA Files

The easiest way to provide a clean environment for each island is to let them run each in their own directory. Therefore, the controller creates a directory for each island and copies the executables of the selector, the executable of the variator and the PISA configuration file to it. The configuration files of the variator and the selector are not only copied, but also if they contain a line specifying a random seed, this value is changed randomly to allow the islands different starting points. Additionally a file named *migration_condition* is created in each directory. It contains the condition for the variator under which it exports the population. Normally this condition is a generation number, but could also be another criterion, like stagnation, that can be checked by the variator.

4.2.2 Start the Islands Using Condor or SSH

The computer infrastructure at the ETH Zurich offers two possibilities to start processes remotely on different machines. The first possibility is to use Condor [16], a software to distribute large, computation intensive processes on idle machines. The second possibility is to login via SSH on each machine, start the processes and log out. Both variants have their advantages and drawbacks. Condor is an uncomplicated program, which runs discretely only on computers which are idle. This discreteness can be a problem when each island is a separate condor job and one of them is interrupted, for example by a user login in the same machine. The problem in this case is that certain island topologies require all islands to run synchronously, meaning if one island is not ready it holds back all the others. Starting a process via SSH is much faster than executing a condor job, however the machines have to be specified by hand and adding to the cluster or removing compute server from the cluster is more complicated than in the case of Condor.

4.2.3 Migrating the Individuals During the Run

The available settings and parameters of an island model, described in chapter 3, influence the behavior of the island model a lot. In order to experiment with

these parameters on multi-objective problems, the user must be able to specify them. Therefore, the following parameters can be set in the parameter file of the controller:

Migration topology: To test the influences of different diameters and therewith the varying speed at which information travels through the network, the controller is able to arrange the islands in three common topologies: torus, ring and random. It can be expected that an IM which is using the random topology converges the fastest, as it adds the highest selection pressure due to its small diameter [8].

Migration Strategy: Using the hypervolume based migration method the controller offers four migration strategies:

- Best to worst
- Random to worst
- Best to random
- Random to worst

In addition to these well known strategies, a fifth migration policy is added to the controller: When this mode, called *Best of all to worst*, is chosen the search for the *best* individual is not only done on one island but on all islands. This equals a topology of a fully connected graph with variable migration size. The *Best of all to worst* migration policy seems to offer a faster convergence than the strategies described above, but has the drawback of bad scalability. The computational effort of the migration is in this case $O(n^2)$, whereas the other migration policies have complexity $O(n)$, with n being the number of islands.

Migration Interval: As the migration interval is an important factor, which influences the diversity of the overall population and the selection pressure on the islands, the migration interval can be specified in the controller configuration file (see appendix C).

Migration Size: Several papers emphasize the importance of a well chosen migration size. In single objective evolutionary algorithms it should be adapted to the subpopulation size, the island topology and the desired selection pressure. Especially if too big migration sizes are used a lost of diversity is reported. To avoid loss of diversity the controller uses the concept of the adaptive migration size described in chapter 3.

4.2.4 Collecting the Results

The last step the controller does, is to collect the results of the different islands when they have terminated. This is done by reading the PISA output files of the islands, getting the objective values, and writing them to a global output file.

Once the controller is configured, the island model can be started with one command, and the final population is written to one file, which means that one can run a parallel version of the evolutionary algorithm with only a little more effort than running the non-parallelized version, but with a significant saving of execution time.

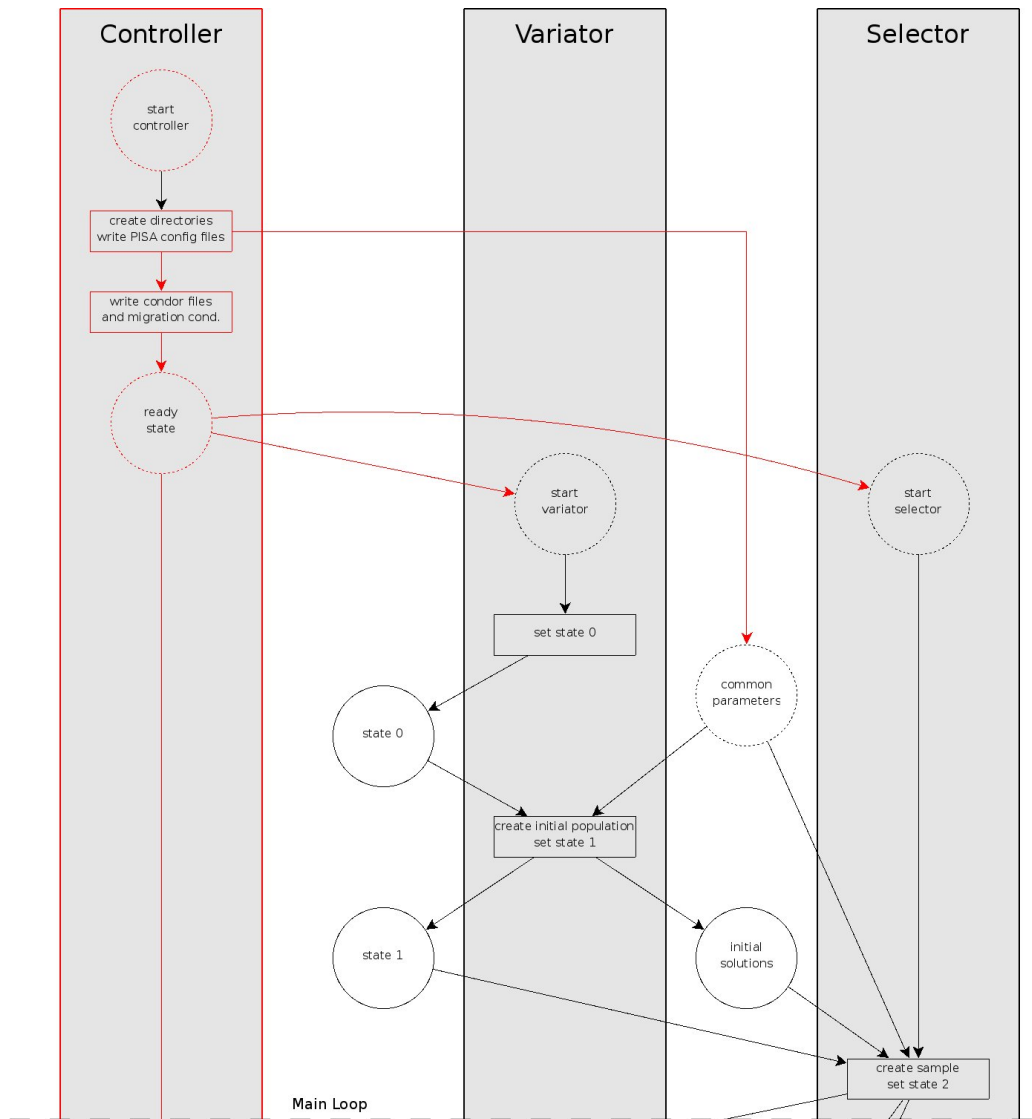


Figure 4.1: Part 1 of the state graph of the extended PISA protocol

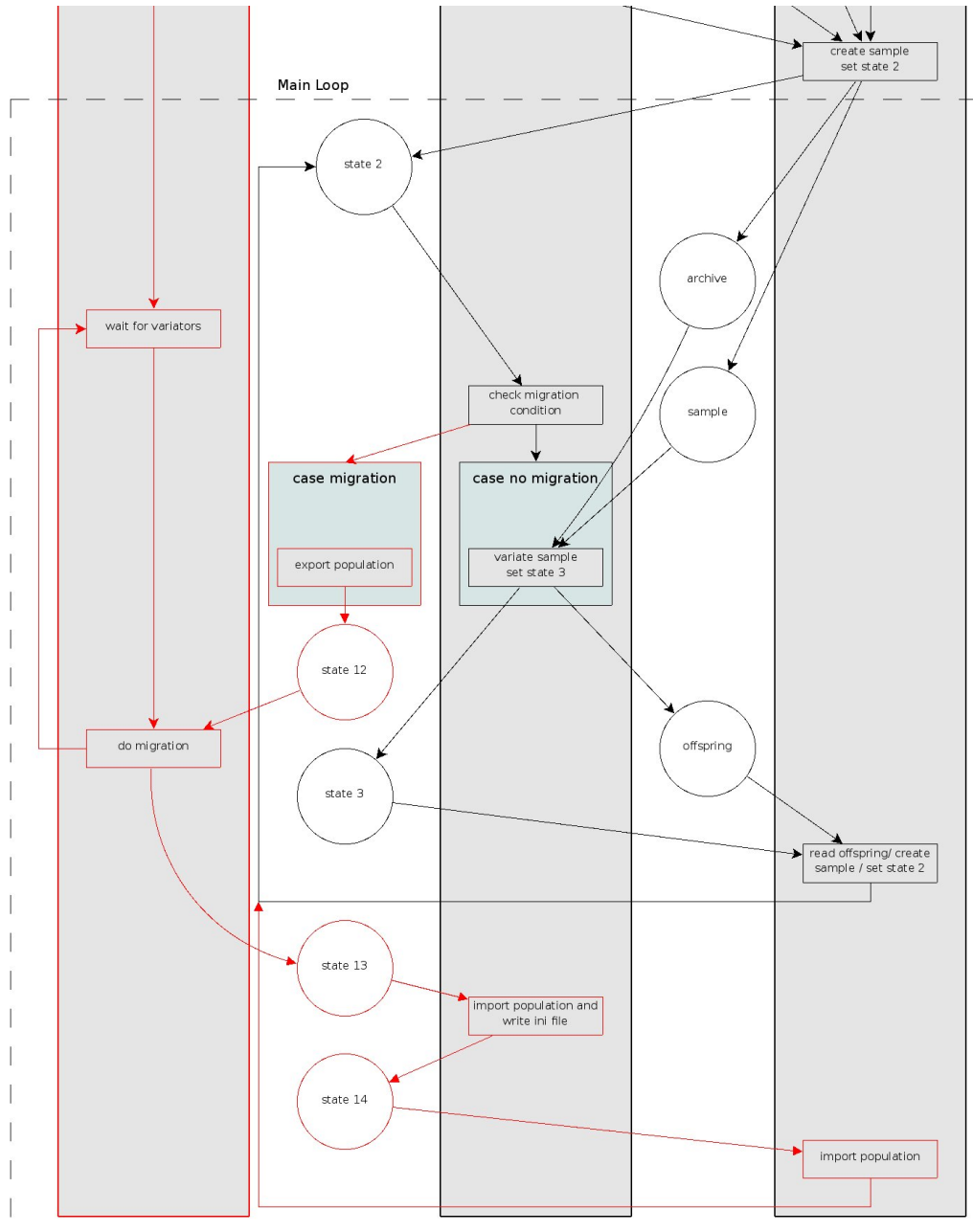


Figure 4.2: Part 2 of the state graph of the extended PISA protocol

Chapter 5

Experiments

The following experiments were made to investigate the influence of the parameters of the island models on the behavior and performance of multi-objective evolutionary algorithms. Furthermore, the proposed hypervolume based migration policy is analyzed. Concerning that, the concept of the adaptive migration size is tested and compared with the results of a constant migration size.

This chapter is organized as follows: first, the settings of the experiments and the test problem are described. The next section deals with the influence of the island topology, succeeded by a section which compares the different migration policies available in the controller module. Then the effects of the number of islands in the island model, a constant migration size and an adaptive migration size are experimentally investigated. At the end the development of the hypervolume over time is analyzed and the speedup of the parallelized version is tested.

5.1 Settings

To test the island model, a multi-objective benchmark problem should be used. The DTLZ test problem suit describes several MOPs with a well known Pareto front [17]. In the following experiments the DTLZ2 problem, described below, was used.

$$\begin{aligned} f_1(\vec{x}) &= (1 + g(\vec{x}_M))\cos(x_1\pi/2)\cos(x_2\pi/2) \cdots \cos(x_{M-2}\pi/2)\cos(x_{M-1}\pi/2), \\ f_2(\vec{x}) &= (1 + g(\vec{x}_M))\cos(x_1\pi/2)\cos(x_2\pi/2) \cdots \cos(x_{M-2}\pi/2)\sin(x_{M-1}\pi/2), \\ f_3(\vec{x}) &= (1 + g(\vec{x}_M))\cos(x_1\pi/2)\cos(x_2\pi/2) \cdots \sin(x_{M-2}\pi/2), \\ &\vdots \\ f_{M-1}(\vec{x}) &= (1 + g(\vec{x}_M))\cos(x_1\pi/2)\sin(x_2\pi/2), \\ f_M(\vec{x}) &= (1 + g(\vec{x}_M))\sin(x_1\pi/2), \\ 0 &\leq x_i \leq 1, \quad \text{for } i = 1, 2, \dots, n, \\ \vec{x}_M &= (x_M, x_{M+1}, \dots, x_n) \\ \text{with } g(\vec{x}_M) &= \sum_{x_i \in \vec{x}_M} (x_i - 0.5)^2 \end{aligned}$$

The functions f_i have to be minimized and at the Pareto front they satisfy the constraint:

$$\sum_{i=1}^M (f_i)^2 = 1$$

DTLZ2 allows to set the number of decision variables in the vector \vec{x} and the number of dimensions freely. For the experiments, the number of dimensions M was set to 2 and the number of decision variables n to 300. These settings were determined using the original PISA framework with the goal that 500 generations take around 300 seconds.

Three state of the art selectors were adapted to be able to run island models: HypE [18], IBEA [19] and SPEA2 [20]. For the experiments, HypE was chosen because it leads to better distribution in the objective-space than IBEA and works better in higher dimensional problems than SPEA2. If not mentioned differently, the following settings for the island model were used:

- 4 islands
- 90 individuals per island
- torus topology
- best to worst migration strategy (hypervolume based)
- adaptive migration size of maximally 12 (3 in each direction)
- migration interval of 50 generations
- 500 generations per run

To evaluate the results and to be able to compare them, a performance measure has to be chosen. Instead of just visually comparing the Pareto fronts found by evolutionary algorithms, a few performance measurement methods appropriate to compare different populations of multi-objective problems were developed by Knowles, Thiele and Zitzler [3, 21]. The populations of the experiments are compared using the hypervolume indicator, as it indicates the dominance of one population over another with a strictly larger indicator value and the maximization of the hypervolume implicates the search for a population which contains all Pareto-optimal objective vectors. The hypervolume indicator is the only known multi-objective measure with these properties. Furthermore, in some cases statistical tests on the significance were done using the Kruskal-Wallis analysis and the Conover Inman post hoc tests with a significance level of 5% [22].

All tests were executed on AMD 64 bit quad-core PCs from the computer cluster of the Computer Engineering and Networks Laboratory at ETH Zurich. Each test was run 30 times with identical parameter settings to allay effects of randomness.

5.2 Migration Topology

Island models with islands arranged in the three implemented topologies should result in qualitatively different results due to different information diffusion speed. This effect can be seen in Figure 5.1. It seems, that the performance benefits from

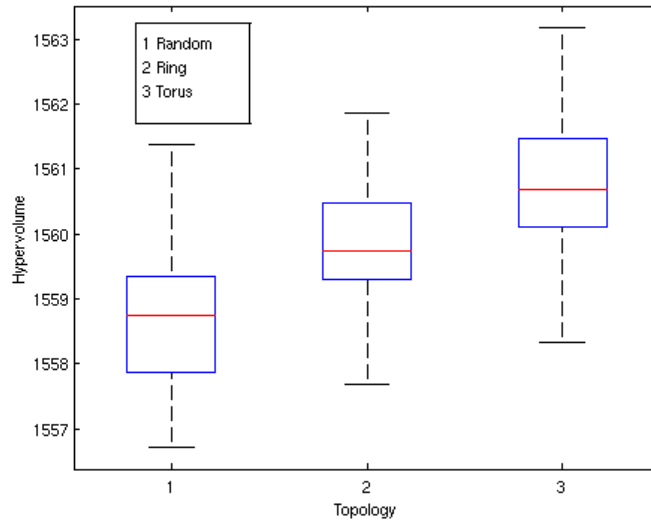


Figure 5.1: The effect of different topologies on the hypervolume of the population

regular information diffusion, because the ring topology and the torus topology lead to a better performance than the random topology. However, in the case of 4 islands the torus and ring topology are densely connected but for a higher number of islands, the diameters of these graphs increase faster than the diameter of the random graph. Therefore, the search for the best topology would have to be redone for a large number of islands.

Interestingly, the topology does not influence the statistical dispersion of the hypervolume values of several runs strongly but the medians differ significantly.

5.3 Migration Strategy

The study of the migration strategy is of special interest because the approach presented in this work differs conceptually from the hitherto selection and replacement policies of parallel MOEAs. Most of them are focused on splitting up the search range by assigning each island a different part of the objective space. This was done for example by using clustering algorithms for the migration policy [13] or by using cone separation [23]. The hypervolume based approach proposed in this thesis works not by dividing the search space but by letting each island search on the whole objective space.

Erick Cantú Paz observed that the selection and replacement strategies influence strongly the selection pressure [5]. The migration policies can be ranked by the selection pressure they add. According to [5], the order of the migration policies, sorted by decreasing selection pressure, looks like this:

1. Select the best individuals, in terms of hypervolume contribution, for migration and replace the worst individuals with the smallest hypervolume contributions.
2. Select the best individuals, in terms of hypervolume contribution, for migration and replace random individuals.
3. Select random individuals for migration and replace the worst individuals with the smallest hypervolume contributions.
4. Select random individual for migration and replace random individuals.

For a higher selection pressure one can expect a faster convergence to the Pareto-front and therewith a higher hypervolume. Figure 5.2 shows the boxplots, for the comparison of the migration strategies, of the hypervolume of the population after 500 generations. Additionally to these four well known strategies, the *Best of all to worst* (see chapter 4) was tested too and the hypervolume of the panmictic evolutionary algorithm is included as a reference. The effect of the increased selection pressure can be seen very well. The first three strategies are statistically significantly better than the serial EA. The two remaining migration policies perform significantly worse. This means that the additional selection pressure is not high enough to compensate the drawback of the smaller population.

5.4 Number of Islands

The impact of an increasing number of islands is important to study, as this could show potential limits of the parallelization. Although it may be possible to run island models with a large number of islands this seems not reasonable. Splitting up the population in very small demes is common in fine grained parallel evolutionary algorithms and should be tested in the context of such models.

To have the same number of fitness evaluations and therewith to permit a fair comparison, the populations should have the same number of individuals. Therefore, the population was chosen to be 360 individuals which can be split up in several equally sized subpopulations. The migration sizes were chosen to be approximately 10% of the size of a subpopulation as suggested in [9]. Table 5.1 lists the island numbers, subpopulation sizes and migration sizes. The rightmost column gives an impression of how the information diffusion varies by specifying the total number of migrated individuals. One can see in Figure 5.4 that the hypervolume of the final population generally increases with the number of islands. However, in the case of 6 islands and in the case of 8 islands the performance is worse. This may be due

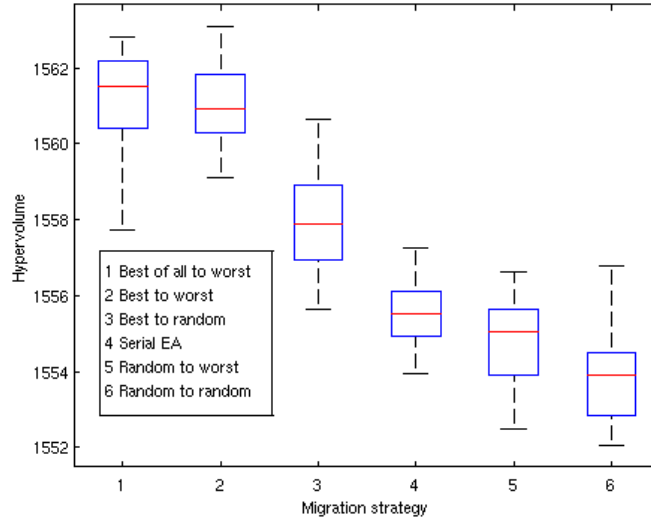


Figure 5.2: The effect of different migration selection and replacement strategies on the hypervolume of the population, including the performance of a serial EA

to the torus topology, of which the underlying grid is sometimes a rectangle with unequal length sides and sometimes a square, depending on the number of islands. This can lead to different information diffusion among the islands. Another issue is that the migration size can only be varied in steps of 4 and the minimal migration size in a torus topology is 4 (one in each direction). The model with 5 islands has a subpopulation size of 72 leading to a migration size of 8 individuals per island (2 in each direction). 5 islands migrating 8 individuals per migration interval is a total of 40 exchanged individuals. The island model with 6 islands has a subpopulation size of 60 individuals and a migration size of 4 individuals per island. The total number of migrated individuals of all islands is 24. This decrease is also observed with 8 islands for the same reason. This issue of the torus topology is displayed in Figure 5.3 for five, six and nine islands.

One can conclude that a higher information transfer leads to better results. An increase of the number of islands influences the performance positively, if the total number of migrated individuals is not reduced.

5.5 Migration Size

Figure 5.5 shows the interrelation of the resulting hypervolume and the exponentially increasing migration size. It is noteworthy that the hypervolume values increase for migration sizes of up to 16 individuals but then get worse.

Number of islands	Subpopulation size	Migration size	Total nr. of migrated individuals
1	360	0	0
2	180	16	32
3	120	12	36
4	90	8	32
5	72	8	40
6	60	4	24
8	45	4	32
9	40	4	36
10	36	4	40
12	30	4	48

Table 5.1: The different island models used for the experiments

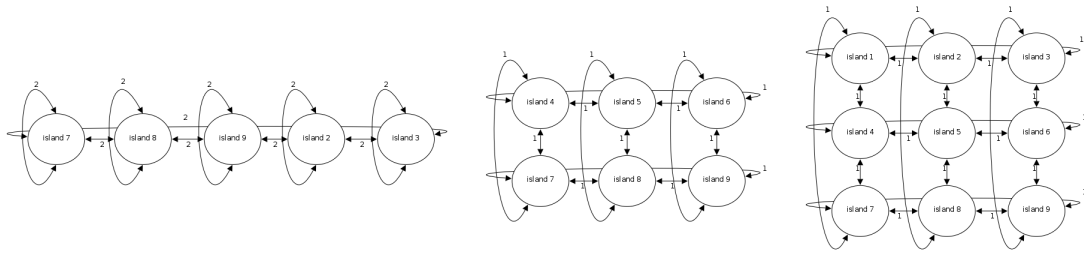


Figure 5.3: How the number of islands influences the migration, if the islands are arranged in a torus topology. The numbers on the arrows indicate how many individuals are migrated in each direction.

This is due to the individuals, which are migrated independent of whether they increase the hypervolume of the subpopulation of the target island or not. Migrating too many individuals leads to a regression of the evolution and decreases the overall diversity.

The experiments using the proposed automatic migration size control can be seen in Figure 5.6. The hypervolume values for big migration sizes indicate that this works better. Higher migration sizes offer the possibility to migrate many individuals, if one island is much worse than the others, but the adaptation makes sure that the high migration size is not disadvantageous in terms of decreasing the target islands hypervolume.

5.6 Development Over Time

Another interesting observation is how the superior performance of the island model over the serial EA can be explained. To test this, the hypervolumes of the population

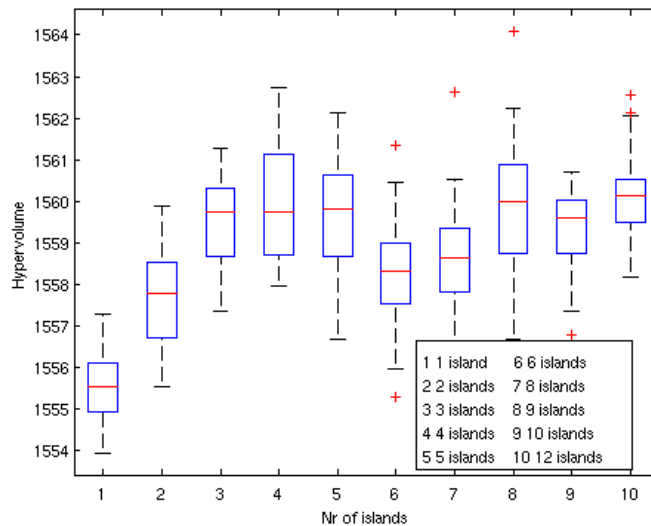


Figure 5.4: The effect of different number of islands on the hypervolume of the population

and the subpopulations were computed at each generation. Figure 5.7 displays the development of the Pareto fronts. The graph displays the mean of the hypervolume indicator over 10 runs. In this experiment migration took place every 10 generation, which is clearly visible in the plot as a hypervolume boost. Noteworthy is also that the island model seems to constantly start with a worse population than the serial EA. The reason for this is that the hypervolume is calculated the first time after one generation. As the hypervolume improvement of the panmictic model is much bigger in the first generation than the increase of the island model, it looks like the starting points would be different. But measurements confirmed the initial populations have the same mean hypervolume.

5.7 Application to a Real World Problem

As we have seen, the results of the hypervolume based migration strategy and of the migration size adaptation work well on a synthetic test problem. But are these principles also applicable to a real world multi-objective optimization problem?

At the Computer Engineering and Network Lab at the ETH Zurich Matthias Woehrle et al. recently developed an evolutionary algorithm which tries to find a set of optimally placed wireless sensor nodes [24]. Optimally placed in this case means the minimization of the number of nodes to cover a specified area and at the same time the minimization of the mean transmission error rate. An individual in the wireless sensor network EA consists of the number and the positions of the wireless sensor nodes. From these positions and the given scenario (walls etc.) the

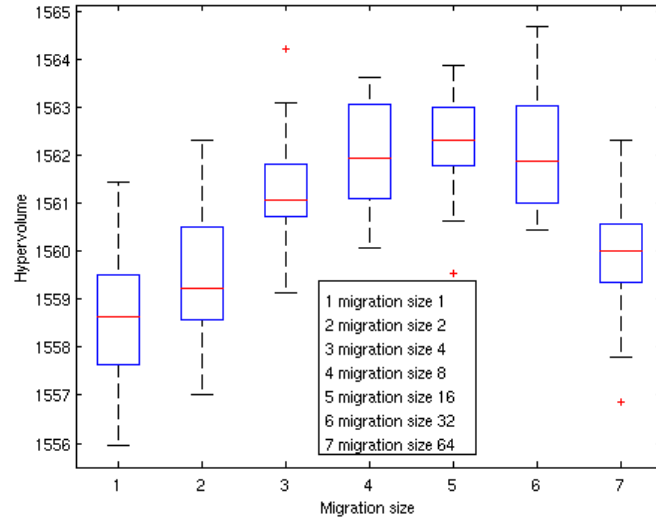


Figure 5.5: The effect of increasing the migration size without the proposed adaption on the hypervolume of the population

transmission error rate is calculated. These computations are very time consuming and a fitness evaluation of an individual can easily take 20 seconds or more. An execution of the EA therefore lasts from several hours to some days, depending on the population size and the number of generations. The fact that the fitness evaluation is very costly and one expect a considerable time saving, makes it ideal to investigate the advantages of the implemented island model on a real world problem.

The wireless sensor network EA was executed with the configurations depicted in Table 5.2. One can also see from Table 5.2 the time the execution of the EA took. The island model is 3.35 times faster than the standard EA. Normally one

Parameter	Panmictic EA	Island model
Nr. of islands	1	4
Individuals per island	40	10
Generations	30	30
Migration interval	-	2 generations
Migration size (adaptive)	-	9 individuals
Island topology	-	torus
Total time	10245 seconds	3053 seconds
Total hypervolume	6.866	7.121

Table 5.2: The configurations for the two different runs of the wireless sensor network evolutionary algorithm

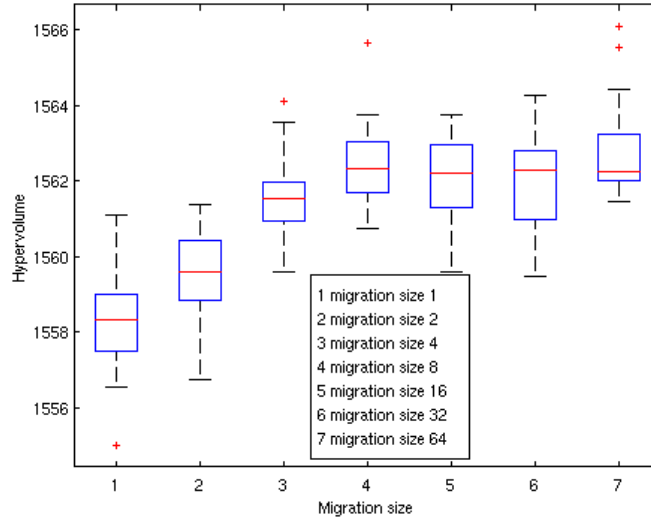


Figure 5.6: The effect of increasing the maximal migration size on the hypervolume of the population, using the hypervolume based adaption of the migration size

would assume that an island model with 4 islands is nearly 4 times faster than the non-parallelized version. The observed slowdown has two reasons: On the one hand this is the migration which demands some computations from the controller and on the other hand the islands are migrated synchronously which caused waiting times of over 100 seconds for some islands. The migration in this setting averages around 56 seconds of which only 6 seconds are really needed to execute the migration – the rest is waiting time. For EAs such as the wireless sensor network EA, in which the computation time of the fitness is depending on the individuals and therefore can be different, an asynchronous island model would be faster.

Qualitatively the island model lead to better results, like one can see in Figure 5.8. The Pareto front of the IM nearly completely dominates the population of the serial EA. Concerning this, the speedup is even more than 3.35 as the panmictic model would need more time to reach a similar Pareto front.

5.8 Superlinear Speedups

By looking more precisely at the execution times from the wireless sensor network experiment one can find an interesting fact: the execution of the island model took 3053 seconds, of which 28% seconds were either waiting for the other islands to be ready for a migration or to actually perform the migration, but the time used for the actual migration is maximally 10 seconds per migration interval.

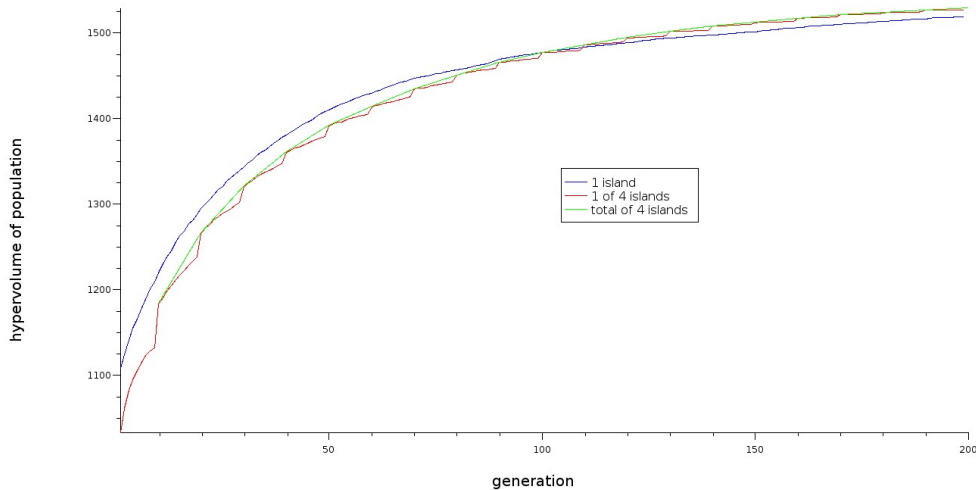


Figure 5.7: A comparison of the development of the hypervolume over 200 generations between the panmictic model and a 4 island model

If one could eliminate this waiting time, by either using an asynchronous model or choose a problem in which the time, the fitness evaluation takes, is independent of the individual, the island model could be more than n times faster on n islands.

Several observations of speedups of this kind have been reported [5, 25, 26]. The speedup s_n of a parallel algorithm is defined by:

$$s_n = \frac{T_1}{T_n}$$

where T_1 is the execution time of the non-parallelized algorithm, T_n the execution time of the parallelized version. If $s_n > n$, one commonly refers to this as a super-linear speedup. In the case of parallel evolutionary algorithms this definition is not very clear as T_n could be the time used to reach the same number of generations or to achieve the same result as the non-parallelized version. Van Veldhuizen et al. propose that for a fair comparison both algorithms should produce identical results, which means in the case of evolutionary algorithms an identical expected solution quality [7].

To efficiently compare the running times, the parallelized model ran on just one single-core machine, using threads, to exclude the influence of different architectures. A linear speedup would in this case be if both executions take the same time (because the islands run all on the same processor). A superlinear speedup would mean a shorter execution time for the island model. The details of the two compared algorithms can be seen in Table 5.3. It is visible from Figure 5.9 that the parallelized algorithm is not only faster but also produces better results. Both

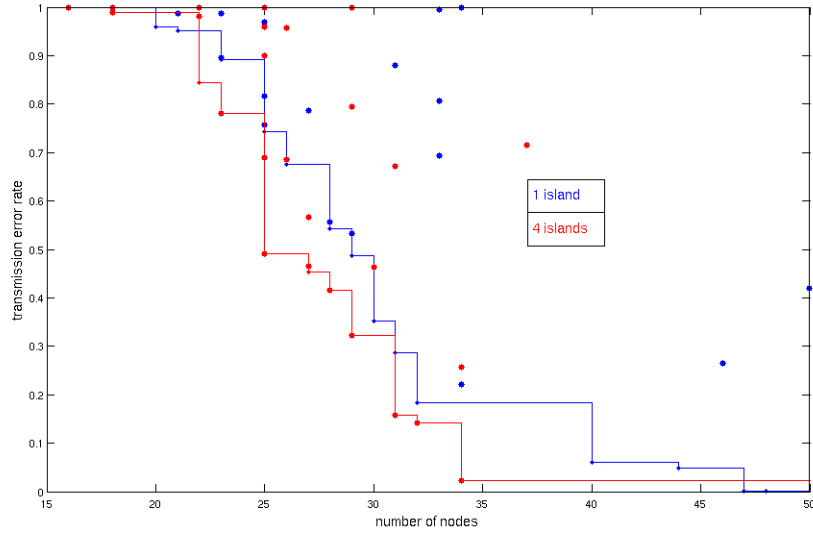


Figure 5.8: The population of the island model nearly completely dominates the panmictic population after the same number of generations

phenomena have different causes: the shorter overall execution time is caused by the bad scalability of HypE for large population sizes, which means the computational effort of HypE increases nearly quadratically with the population size in a two objective problem. As a consequence dividing the population in 9 subpopulations diminishes the time needed by the selector more than 9 times. The better quality of the results is caused by the previously discussed, added selection pressure from the migration strategy.

Two further tests have been performed to analyze how these improved results and the time savings depend on the selector. For these two tests the configuration of the island models was identical to the previous one, only the selectors were different. The results of the runs with IBEA as selector can be seen in Figure 5.10 and the

Parameter	Panmictic EA	Island model
Problem	DTLZ2	DTLZ2
Nr. of islands	1	9
Individuals per island	360	40
Generations	500	500
Migration interval	-	50 generations
Migration size (adaptive)	-	30 individuals
Island topology	-	torus

Table 5.3: The configurations for the time measurement experiment

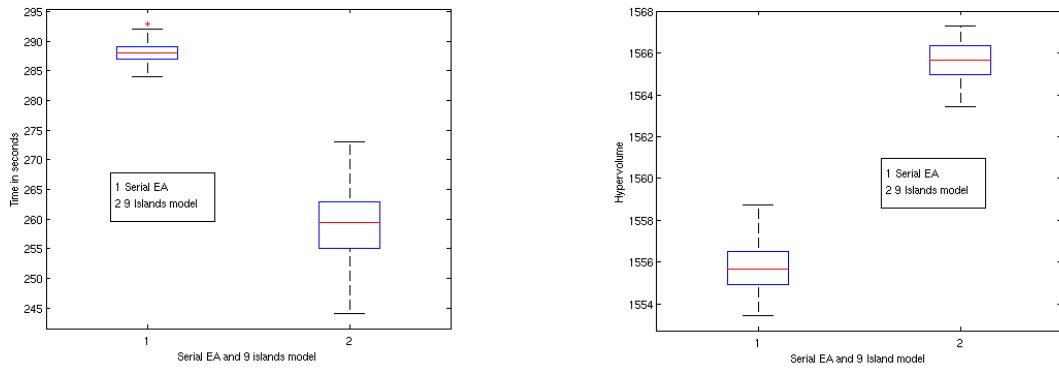


Figure 5.9: A comparison of the execution time and the hypervolume of a serial EA and an island model, using the HypE selector.

runs with HypE in sampling mode (see [18]) are shown in Figure 5.11.

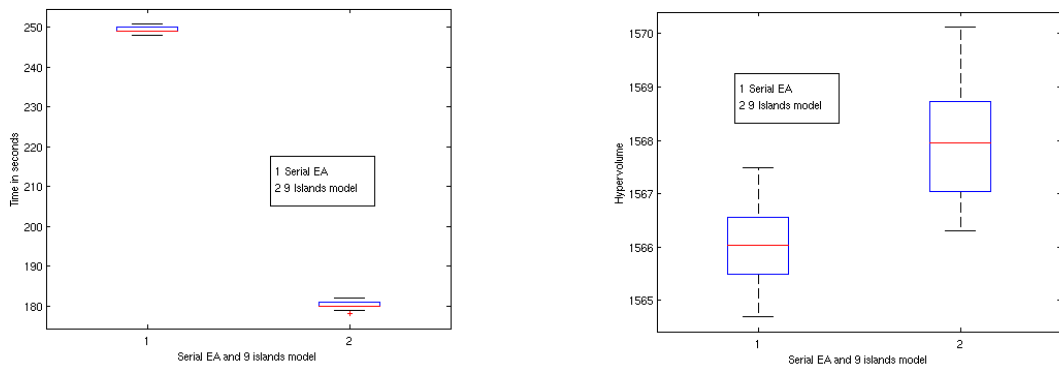


Figure 5.10: A comparison of the execution time and the hypervolume of a serial EA and an island model, using the IBEA selector.

IBEA's execution time increases quadratically with the population size, which causes the island model to be faster in this case, too. The difference of the final hypervolume value is smaller than when using HypE but it is still significantly higher.

For the last test, HypE was set to compute the selection in sampling mode. As a consequence of this, the computational effort of it increases only nearly linearly with the population size. This leads to a faster execution of the serial EA compared to the island model. The slowdown of the IM is mainly caused by the migration overhead. Although it seems there is no superlinear speedup in this case, an interesting calculation can be done: the mean hypervolume of the final island model population is 1563.52. Further tests showed that the panmictic EA would have to run for 896 generations to achieve a mean hypervolume of 1563.52. 896 generations

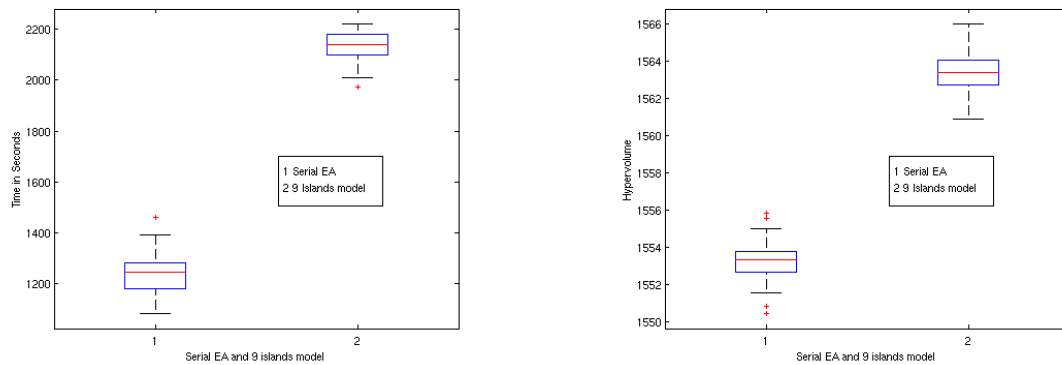


Figure 5.11: A comparison of the execution time and the hypervolume of a serial EA and an island model, using the HypE selector in sampling mode.

take with the serial EA approximately 2223 seconds, which would make the serial EA again to be slower than the IM (mean 2134 seconds) to achieve the same results.

As a conclusion one can say, that it may take less time for an island model to reach the same results than for a serial EA, even if all processes are executed on one single core PC, mainly due to the selection pressure of the migration strategy and the scalability of the selectors. Therewith, superlinear speedups with island models are possible.

Chapter 6

Conclusion

Parallelizing evolutionary algorithms helps to make use of the parallel architectures and computer clusters, which become more and more widespread. This increase of available computation power is strongly needed, as the evolutionary algorithms employed nowadays can have very long execution times.

Out of the several common parallelization strategies, this thesis concentrated on the investigation of island models applied to multi-objective EAs. The platform- and programming language independent framework for evolutionary algorithms PISA, was extended by the possibility to run island models.

Instead of using existing migration strategies for the island model, a new approach was proposed. It is based on maximizing the hypervolume of each island and is therewith complementary to the commonly applied niching techniques, because all islands cover the whole objective-space. With this migration strategy, a new way to automatically regulate the migrations size was introduced.

Experiments with the island models showed how evolutionary algorithms can profit from parallelization and which parameters influence the performance. The proposed migration strategy led to superior results of the island models compared to serial EAs in the same number of generations and with an equal total population size. The adaptive migration size takes care that the information flow between the islands is appropriate and the difficult decision about the migration size does not have to be made manually.

To demonstrate that the island model does not only work on test problems but can also be applied to real world problems, the performance of a parallelized version of the wireless sensor network EA was compared to the serial version.

The superior results of the island models motivated the last tests, which analyzed the behavior of island models in non-parallel architectures with different selectors. Due to the scalability of the selectors, higher hypervolume values can be achieved in less time if island models are used instead of the serial EA.

6.1 Future Work

The experiments with the wireless sensor networks EA disclosed the need for a asynchronous island model, in order to maximize the time savings for evolutionary algorithms with variable expensive fitness evaluations. An asynchronous model would be significantly faster on such problems and the property of not having to wait for all islands to be ready, would make it more resistant against errors caused by the failure of a single island.

As the hypervolume based migration strategy works well, it could be generalized to be used with an arbitrary performance indicator. This means a receiving island could replace the individuals which contribute the least to this performance measurement and they would be replaced by the individuals from other islands which increase the performance measurement the most.

Considering the results of this thesis, there is a big potential in the parallelization of multi-objective evolutionary algorithms. However, as long as the implications of the parallelization are not understood theoretically, there is still a lot of interesting research to do in this field.

Bibliography

- [1] J. Bose, T. Reiners, D. Steenken, and S. Vos. Vehicle dispatching at seaport container terminals using evolutionary algorithms. In *HICSS*, 2000.
- [2] A. A. Tantar, N. Melab, E. G. Talbi, B. Parent, and D. Horvath. A parallel hybrid genetic algorithm for protein structure prediction on the computational grid. *Future Gener. Comput. Syst.*, 23(3):398–409, 2007.
- [3] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. In *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 292–304, London, UK, 1998. Springer-Verlag.
- [4] M. Fleischer. The measure of Pareto optima. Applications to multi-objective metaheuristics. In C. M. Fonseca et al., editors, *Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)*, volume 2632 of *LNCS*, pages 519–533, Faro, Portugal, 2003. Springer.
- [5] E. Cantú-Paz. Migration policies, selection pressure, and parallel evolutionary algorithms. *Journal of Heuristics*, 7(4):311–334, 2001.
- [6] G. Winter, D. Greiner, and B. Galvan. Parallel evolutionary computation. CEANI (Evolutionary Computation and Applications) Division of the Institute of Intelligent Systems and Numerical Applications in Engineering (IUSIANI), University of Las Palmas de Gran Canaria, Spain, 2005.
- [7] D. A. Van Veldhuizen, J. B. Zydallis, and G. B. Lamont. Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 7(2):144–173, 2003.
- [8] M. Tomassini. *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
- [9] Z. Skolicki and K. De Jong. The influence of migration sizes and intervals on island models. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1295–1302, New York, NY, USA, 2005. ACM.

-
- [10] S. Park, J. Kim, and C. Lee. Topology and migration policy of fine-grained parallel evolutionary algorithms for numerical optimization. Taejon-shi, 305-701, Republic of Korea, 2000. Dept. of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST).
- [11] W. F. Punch, R. C. Averill, E. D. Goodman, S. Lin, and Y. Ding. Using genetic algorithms to design laminated composite structures. *IEEE Expert: Intelligent Systems and Their Applications*, 10(1):42–49, 1995.
- [12] C. A. Coello Coello and M. R. Sierra. A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In *MICAI*, pages 688–697, 2004.
- [13] F. Streichert, H. Ulmer, and A. Zell. Parallelization of multi-objective evolutionary algorithms using clustering algorithms. In *EMO*, pages 92–107, 2005.
- [14] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons, Chichester, 2001.
- [15] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA—A Platform and Programming Language Independent Interface for Search Algorithms. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)*, volume 2632 of *LNCS*, pages 494–508, Berlin, 2003. Springer.
- [16] University of Wisconsin-Madison. <http://www.cs.wisc.edu/condor>, May 2008. Condor Project Homepage.
- [17] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable Test Problems for Evolutionary Multi-Objective Optimization. In A. Abraham, R. Jain, and R. Goldberg, editors, *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*, chapter 6, pages 105–145. Springer, 2005.
- [18] J. Bader and E. Zitzler. Hype: Fast hypervolume-based multiobjective search using monte carlo sampling. 2008. to appear.
- [19] E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In Xin Yao et al., editors, *Parallel Problem Solving from Nature (PPSN VIII)*, pages 832–842, Berlin, Germany, 2004. Springer-Verlag.
- [20] E. Zitzler, M. Laumanns, and L. Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.
- [21] J. Knowles, L. Thiele, and E. Zitzler. A tutorial on the performance assessment of stochastic multiobjective optimizers. 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland, February 2006. revised version.

-
- [22] W. J. Conover. *Practical Nonparametric Statistics*. J. Wiley, 3 edition, 1999.
- [23] J. Branke, H. Schmeck, K. Deb, and M. Reddy. Parallelizing multi-objective evolutionary algorithms: cone separation. In *Congress on Evolutionary Computation*, volume 2, pages 1952–1957. IEEE, JUN 2004.
- [24] M. Woehrle, D. Brockhoff, T. Hohm, and S. Bleuler. Investigating Coverage and Connectivity Trade-offs in Wireless Sensor Networks: The Benefits of MOEAs. In *Conference on Multiple Criteria Decision Making (MCDM 2008)*. Springer, 2008. to appear.
- [25] J. He and X. Yao. Analysis of scalable parallel evolutionary algorithms. IEEE, JUN 2006.
- [26] E. Alba. Parallel evolutionary algorithms can achieve super-linear performance. *Inf. Process. Lett.*, 82(1):7–13, 2002.
- [27] ETH Zurich Systems Optimization. <http://www.tik.ee.ethz.ch/sop/pisa/>, May 2008. PISA - A Platform and Programming Language Independent Interface for Search Algorithms - Webpage.
- [28] R. Gagnon. <http://codeguru.earthweb.com/java/articles/366.shtml>, May 2008. Java code for the little endian - big endian conversion.

Appendix A

List of Abbreviations

EA	Evolutionary Algorithm
FGPEA	Fine Grained Parallel Evolutionary Algorithm
IM	Island Model
MOEA	Multi-Objective Evolutionary Algorithm
MOOP	Multi-Objective Optimization Problem
PEA	Parallel Evolutionary Algorithm
SSH	Secure Shell

Appendix B

Parallelizing an Existing PISA Module

In order to use an existing PISA module with the controller software a few changes have to be done. Modules which already contain these changes can be downloaded from the PISA project Website [27] and can be used as a reference.

B.1 Necessary Changes to the Variator

The following things have to be added to the variator:

- Allow states 12 - 14
- Check migration file
- Add an *export* function
- Add an *import* function
- Add a *state 13* function

Allow states 12 - 14 Most PISA modules check if the state read from the state file is valid. Therefore they throw an exception if the state is bigger than 11. In most implementations the upper bound of the state number has to be adjusted to 14 in the function `read_state()` and in the function `write_state()`.

Check migration file Before entering state2 the file *migration_condition*, which is in the same directory as the executable, has to be opened for reading and interpreted as follows:

- If the first line of the file is of the form *generation <integer>* and the actual generation count equals *<integer>* then
 - write $0 \setminus n$ to the *migration_condition* file

Length	Content
4 Bytes	Length of Ind. 1 = N_1
N_1 Bytes	Individual 1
4 Bytes	Length of Ind. 2 = N_2
N_1 Bytes	Individual 2
...	...
...	...
4 Bytes	Length of Ind. α = N_α
N_α Bytes	Individual α

Table B.1: The structure of the *population* file

- remove the individuals which are not in the *arc* file (In most modules there's a function with this functionality)
 - call the *export* (see below) function
 - set the state to 12 and write it to the state file
- If the first line equals the word *migrate* then
 - write $0 \setminus n$ to the *migration_condition* file
 - remove the individuals which are not in the *arc* file (In most modules there's a function with this functionality)
 - call the *export* (see below) function
 - set the state to 12 and write it to the state file

Add an export function The export function writes the whole population to two files. Both files lie in the same directory as the executable of the variator. One file is a binary file and is named *population*. It contains the internal representation of all individuals of the population. The other file, a text file, is named *objectives* and contains the objective values of all the individuals in the *population* file in the same sequence as in the *objectives* file.

The structure of the *population* file can be seen in table B.1. It contains the length of the representation of an individual in bytes and after this the actual individual. The length is a 4 byte integer in **little endian** notation (on x86 machines). This is of special importance if java programs are involved, as the java virtual machine normally stores integers in the big endian notation. A simple java algorithm for the conversion between the two formats can be found at [28].

The *objectives* file consists of one line per individual. The objective values on one line are in exponential format and are separated by whitespaces. A line is terminated by a $\setminus n$.

Add an export function First the import function deletes the existing population. Then it reads the new population from the *population* file and the corresponding objectives from the *objectives* file. These individuals are added to the new population. After importing all individuals from the files the *PISA.ini* file, according to the original PISA protocol, is written containing this new population.

Add a state 13 function This function does three things:

1. It reads the *arc* file according to the PISA specification.
2. It reads the *sel* file but the read individual indexes are not used.
3. It calls the *import* function.

The last thing to add is a branching to the *state 13* function, at the loop where state file is read and derived to the appropriate state function, has to be added. If the *state 13* function returns successfully state 14 has to be written to the state file.

B.2 Necessary Changes to the Selector

The following things have to be added to the selector:

- Allow states 12 - 14
- Add a *state 14* function

Allow states 12 - 14 Most PISA modules check if the state read from the state file is valid. Therefore they throw an exception if the state is bigger than 11. In most implementations the upper bound of the state number has to be adjusted to 14 in the function *read_state()* and in the function *write_state()*.

Add a state 14 function The action of the function *state 14* is very similar to state 1. The only difference is that the population struct doesn't have to be created. The *PISA.ini* file is read and the environmental and mating selection written to *arc* and to the *sel* file respectively.

Additionally a branching to the *state 14* function, at the loop where state file is read and derived to the appropriate state function, has to be added.

Appendix C

Using the Controller

C.1 Compiling and Running the Controller

The controller source file has only standard C dependencies. It can be compiled under Linux or Unix with the command:

```
make
```

After a successful compilation it can be run under Linux or Unix with the command:

```
./controller < param_file_name >
```

C.2 The Controller Configuration File

All parameter of an island model described in this thesis can be set in the controller configuration file. Each line of it contains a parameter and the value of the parameter, separated by a whitespace. A line not beginning with a parameter name, is ignored. Parameters, which are not essential for the execution, are set to standard values by the controller, if they are not set in the configuration file. The names and possible values (separated by commas) for the parameters are in Table C.1. Compulsive parameters are bold.

<*string*>, <*integer*> or <*float*> mean the value can be an arbitrary string, a positive integer or a positive floating point number. The controller allows only to specify one executable file and one configuration file for the selector and for the variator. If additional files or folders are necessary to execute the selector or the variator and have to be copied in the directories of the islands, a shell script should be written, which copies the according files or folders and after this executes the selector/variator. The parameter which specifies the executable of the selector/variator should be set to the name of this shell script.

Parameter	Possible values	Comment
pisa_filenamebase	<string>	The file name base of the PISA configuration file
variator_config_file	<string>	Name of the configuration file of the variator
selector_config_file	<string>	Name of the configuration file of the selector
variator_executable	<string>	Name of the executable file of the variator
selector_executable	<string>	Name of the executable file of the selector
number_of_islands	<integer>	Number of islands of the island model
output_file	<string>	Name of the file where the final population is written by the variator
dirnamebase	<string>	Name base of the island directories
variator_polling_interval	<float>	The interval in seconds at which the variator polls the state file
selector_polling_interval	<float>	The interval in seconds at which the selector polls the state file
migration_size	<integer>	Number of individuals migrated in each direction
migration_topology	random, torus, ring	Topology of the island model
migration_type	best_to_worst, best_to_random, random_to_worst, random_to_random, bestofall_to_worst	Set the migration strategy (hypervolume based)
distribution_type	ssh_2_threads, condor_2_threads, local	Defines if the threads are started by SSH, Condor, or on the local machine
ssh_machines_file	<string>	Name of the file containing all names of the machines (separated by newlines), which can be used for the SSH execution (without password prompt)
migration_interval	<integer>	-
grid_length	<integer>	Grid length of the torus in case of torus topology
requirements	<string>	Additional requirements for the Condor execution (see condor documentation)

Table C.1: Parameter names and values of the controller configuration file