The Generalized Island Model

Chapter *in* Studies in Computational Intelligence · January 2012

DOI: 10.1007/978-3-642-28789-3_7

Dario Izzo
European Space Agency
331 PUBLICATIONS **6,534** CITATIONS

The Generalized Island Model

Dario Izzo, Marek Ruciński, and Francesco Biscani

Abstract. The island model paradigm allows to efficiently distribute genetic algorithms over multiple processors while introducing a new genetic operator, the migration operator, able to improve the overall algortihmic performance. In this chapter we introduce the generalized island model that can be applied to a broad class of optimization algorithms. First, we study the effect of such a generalized distribution model on several well-known global optimization metaheuristics. We consider some variants of Differential Evolution, Genetic Algorithms, Harmony Search, Artificial Bee Colony, Particle Swarm Optimization and Simulated Annealing. Based on an set of 12 benchmark problems we show that in the majority of cases introduction of the migration operator leads to obtaining better results than using an equivalent multi-start scheme. We then apply the generalized island model to construct heterogeneous "archipelagos", which employ different optimization algorithms on different islands, and show cases where this leads to further improvements of performance with respect to the homogeneous case.

1 Parallelizing Optimization Tasks

Parallel computing is an indispensable tool of modern science. Dividing the given task into independent units that can be performed in parallel can lead to significant reduction of the time needed to obtain desired results. While in the past decades parallel computing machines were a scarce resource, available exclusively to the

Dario Izzo · Francesco Biscani
Advanced Concepts Team, European Space and Technology Center (ESTEC), The Netherlands
e-mail: dario.izzo@esa.int,bluescarni@gmail.com

Marek Ruciński
Centre for Robotics and Neural Systems, Univeristy of Plymouth, United Kingdom
e-mail: marek.rucinski@plymouth.ac.uk

customers of computing centers, nowadays virtually every personal computer sold has parallel computing capabilities in the form of multi-core/multi-threaded CPU or even massively parallel-capable GPU. It is not surprising then that much research is being done about how to utilize best these now omnipresent capabilities.

In the context of optimization, parallel architectures were considered from very early on, actually the first concepts appeared when parallel computing machines were still in a primitive stage [1]. Much of both practical and theoretical research has been done on parallelization of many types of algorithms [14, 2, 16]. Equally much work was then needed to classify and categorize the multitude of existing approaches in order to provide the global view on the subject. From our standpoint, the ways in which parallel computing has been utilized in order to speed up the processes of optimization could be roughly divided into three categories described in the following paragraphs.

The first class of algorithms benefits from the possibility of using parallel computing to speed up the computation of the objective function value for a given solution. This is of course most practical in cases where such calculation requires a relatively large amount of computational effort, such as those where simulations of some sort are involved. Because calculation of the objective function is the core operation of every optimization process, any gain in the execution time of this operation translates directly to shortening of the optimization process itself, often resulting in a linear speed-up. Because parallelism remains "encapsulated" in the objective function, the optimization algorithm does not need to be modified in any way.

Second class of approaches to exploit parallel architectures in optimization takes advantage of the fact that often one step of the optimization algorithm requires evaluating many solutions. This is usually the case for algorithms like the Generic Algorithm (GA) or Differential Evolution (DE), which operate on a set of solutions, often called a *population*. Because objective function evaluations for different solutions from the population are completely independent, they can be easily performed in parallel, shortening the time needed to evaluate the whole set of solutions, and again resulting in a linear speed-up. Also in the case of this class only simple modifications to the original sequential algorithm are required, as the only thing that changes is that the solutions in the population are evaluated in parallel instead of iteratively in a loop. This type of parallelization/distribution strategy is also called master-slave model [5] in the context of GAs, where it was born, as one CPU (the Master) carries out the computations necessary to apply the genetic operators, while the slaves CPUs are delegated to evaluate the chromosome fitness. The common characteristic of the two approaches described above is that the original optimization algorithm is not modified in any significant way. Although some changes naturally have to be done to allow for a parallel implementation, the logic and flow of the optimization process does not change – the results obtained with the sequential and parallel variant of the algorithm are exactly the same, only that in the latter case they are available faster.

The final, third category of approaches is the one in which introduction of the parallel execution is done not only to speed up the processing, but also to exploit completely new dynamics present because of the existence of parallel populations and possible interactions between them. A classical example of such an approach is so-called *coarse-grained* or *island model* parallelization scheme designed for GAs in which many populations co-exist and exchange individuals at certain intervals. Our definition includes however also other schemes, like the *fine-grained* model, in which a local structure of the neighborhood between solutions is introduced and exploited. The important difference between this category of algorithms and previously introduced ones is the necessity of adaptation of the original optimization paradigm.

In this chapter we focus on the island model parallelization scheme. For parallel GAs (PGAs), it has been shown to be superior to the *global* approach (i.e. with only one population) both in practice and by theoretical analysis [5]. In one of our previously published works we have shown that the island model paradigm can be relatively easily used with algorithms other than GAs, both those explicitly utilizing populations and not [17]. While in that work we were focussed mainly on the impact of the network topology defining the island connectivity (migration paths) here we take a step back and we formally introduce the generalized migration operator which allows for a coarse-grained parallel implementation of virtually any optimization algorithm (provided that it fulfills few simple criteria) as well as study heterogeneous clusters of *islands* implementing various algorithms which cooperate together in one optimization process to "evolve" good solutions. Based on computational experiments involving 10 algorithms and 12 benchmark problems we show that introduction of the migration operator allows in case of most of the considered optimization algorithms to obtain better solutions than in a sequential multi-start case with the same amount of computational effort. Subsequently, we make use of the knowledge about preferences of particular algorithms toward migration we gained in those experiments in order to construct a heterogeneous archipelago of various algorithms and compare its performance with the homogeneous case.

Motivations behind performing the study presented herein are the following:

- because of the technological advances, implementation of parallel optimization algorithms is these days much less constrained by the hardware architecture than in the past. This enables much more flexible experimentation with various parallel designs, as the problem of communication overheads became negligible in most cases;
- computation of the objective function is, in a number of representative cases, fast enough to allow for parallel processing involving a large number of solutions grouped in many generously sized populations on one machine;
- parallel processing involving many different optimization algorithms and exchange of information between them is expected to improve the exploration-exploitation balance thanks to the additional variety;

The remaining part of the chapter is organized as follows. First, we present the generalized island model paradigm and an in-depth discussion of the parameters involved. Next, we provide a short overview of PaGMO, our open-source implementation of the model. Then, we present two computational experiments: #1) "the migration dilemma" aimed at establishing for which algorithms introduction of the migration operator yields better results than an equivalent multi-start execution, and #2) "can they cooperate?" comparing performance of homogeneous and heterogeneous archipelagos. The chapter ends with the presentation of conclusions from the computational experiments and a short discussion of future prospects.

2 The Generalized Island Model

2.1 Definition

The island model has been proposed as an extension of a traditional GA with the intention of improving the diversity of solutions and thus delaying stagnation. First works to mention the idea of using a number of subpopulations can be dated as early as 1967. With the dawn of parallel and distributed computing machines in the 1980s, the research on PGAs gained significant momentum (see [1] for a more detailed historical note). The core idea behind the island model is the introduction of a structure to the population used in the original GA. Instead of permitting recombination between any two individuals in the solution pool, this possibility becomes restricted only to solutions belonging to the same sub-population, or *deme*, which number traditionally varies between a few to a few dozen. The sub-populations evolve mostly independently, however at certain relatively sparsely distributed moments of time they are allowed to exchange solutions in a process called migration. From the point of view of the GA theory, the transition from the original model to the island model has thus been frequently viewed as a modification of the selection operator [18]. However, the perspective that allows for a few more general conclusions is to consider the island model simply as running a certain number of global GA algorithms in parallel with the additional introduction of a new operator called the migration operator. Informally, this operator is engaged at certain points of time between two consecutive generations of the original GA algorithm, and its job is both to select individuals from the current island to be sent to other islands, as well as potentially introduce foreign individuals to the local population. Note how, from the point of view of the migration operator, the details of the optimization process that take place in between two events of migration are irrelevant. GAs on different islands could easily use different parameters for example the selection rule, crossover operator or mutation probability. The optimization process would still work, and actually could perform even better than the variant with fixed parameters across islands thanks to the additional variability. One can go even a next step further, and state that for the migration operator it does not matter at all what kind of algorithm is used on the islands, and whether different islands use the same algorithms or not, as long as all the islands use the same problem solution coding and the migration can be "plugged in" on every island. This informal discussion shows that the island model is a general paradigm that can be applied not only to genetic or evolutionary algorithms, but to a much broader family of optimization processes, and even can be used to form heterogeneous archipelagos of islands which use various algorithms. Let us now introduce the concept more formally.

We define an *archipelago* $\mathbb{A}$ as a couple:

$$\mathbb{A} = \langle \mathbb{I}, \mathscr{T} \rangle \tag{1}$$

where $\mathbb{I} = \{I_1, I_2, \ldots, I_n\}$ is the set of *islands*, and $\mathscr{T}$ is the *migration topology*, a directed graph with $\mathbb{I}$ as the set of vertices. Every island I_i is a quadruple:

$$I_i = \langle \mathscr{A}_i, P_i, \mathscr{S}_i, \mathscr{R}_i \rangle \qquad i = 1, 2, \ldots, n \tag{2}$$

where $\mathscr{A}_i$ is the *optimization algorithm* used by the *i*-th island, P_i the population there contained, and $\mathscr{S}_i$ and $\mathscr{R}_i$ are respectively the *migration-selection policy* and *migration-replacement policy* for the island. A population P_i is a couple $\langle \mathscr{P}, \mathbb{P} \rangle$ containing a set of individuals $\mathbb{P}$ whose fitness value is always referred to the problem $\mathscr{P}$ we are considering. Note that, unlike other works, we speak of populations always in connection with their fitness values (i.e. the problem they refer to).

The optimization algorithm $\mathscr{A}$ is any optimization process supporting an evolution operator $P' \leftarrow \mathscr{A}(P, \mu)$, where μ denotes the migration interval (i.e. the number of allowed algorithmic iterations before a migration is allowed). Algorithms may or may not have the following distributive property:

$$\mathscr{A}(\mathscr{A}(P, \mu_1), \mu_2) = \mathscr{A}(P, \mu_1 + \mu_2) \tag{3}$$

if they do not they are referred to as *adaptive* . Note that $\mathscr{A}$ can also operate on populations containing one only individual, in which case we write $|P| = 1$ (simulated annealing is one of such algorithms)

The migration-selection policy $\mathscr{S}$ determines the deme $\mathbb{M} \subseteq \mathbb{P}$ to be sent to other islands via migration. We write this as $\mathbb{M} \leftarrow \mathscr{S}(P)$. In turn, given a deme $\mathbb{M}$, the migration-replacement policy $\mathscr{R}$ specifies how $\mathbb{M}$ could be inserted into a population P. We write this as $P' \leftarrow \mathscr{R}(P, \mathbb{M})$.

We are now ready to specify the flow of the general coarse-grained optimization algorithm. On every island I_i, processing follows the pseudo-code presented on listing 1. Optimization on all islands is performed in parallel. The final result of the optimization is the best solution found over all islands.

Listing 1. Pseudo-code for the island I_i

```
initialize $P$
**while** !stop_criteria
    $P' \leftarrow \mathscr{A}_i(P, \mu_i)$
    $\mathbb{M} \leftarrow \mathscr{S}_i(P')$
    /* Send $\mathbb{M}$ to islands adjacent to $I_i$ in $\mathscr{T}$ */
    /* Let $\mathbb{M}'$ be the set of solutions received from adjacent islands */
    $P'' \leftarrow \mathscr{R}(P', \mathbb{M}')$
    $P \leftarrow P''$
```

2.2 *Parameters*

The generalized island model just introduced has a number of parameters which require further clarification and discussion.

2.2.1 Implementation of Migration

The formal definition given above does not specify all details involved in the implementation of the migration. One question is whether the migration should be *synchronous* or *asynchronous*, or in other words: should the communication event happening in line 5 of 1 block all islands until communication is completed?. Both approaches have advantages and disadvantages. Asynchronous communication seems to be an obvious choice for distributed computing architectures, because no or little global control over the execution of the code on islands is required (just initialization of the tasks and gathering of the final results). It provides good scalability and maximally utilizes available computing resources, as communication overheads are limited to the necessary minimum. On the other hand, because processing time on islands is usually unpredictable, it is also not possible to predict when migration events occur, an thus the resulting algorithm is non-deterministic, non-repeatable and hard to control or debug. These issues are solved if the communication between all islands is synchronized. Under this assumption it is possible to obtain a deterministic process. The price that one has to pay, however, is a potentially significant overhead resulting from the synchronization of all islands and the fact that the slowest island dictates the execution time of the whole process. The presence of a global synchronization mechanism thus limits the speed-up and the scalability of the system.

Should the asynchronous migration be the choice, there are still certain decisions to be made about its implementation. Two opposite approaches could be called migration *initiated by source* and migration *initiated by destination*. The former is the most intuitive implementation of a simple message passing protocol. As soon as an island reaches line 5 of the island model code introduced above, the migrating individuals are sent to adjacent islands. It is very likely that a destination island will not be able to insert incoming individuals immediately to the local population because of being for the moment busy executing the optimization step in line 3, thus every island needs a buffer in which incoming migrating individuals are stored in between two events of execution of line 7 of the pseudo-code. We call this approach initiated by source, because the *transmission* of solutions from one island to another is performed when the *source* island reaches the communication step. In contrast, in the migration initiated by destination, when an island reaches line 5 of the island model code, the solutions are not sent to individuals just yet, put placed in a local buffer or “database”. Then, when an island is about to execute the line 7 of the code, it contacts adjacent islands and fetches individuals available in their “databases” at the moment. The individuals are thus *transmitted* when the *destination* island ask for them. At the first glance this may seem to be just an implementational detail, however these two strategies differ quite significantly in a way that may affect the performance of the optimization. If one consideres a topology with one island acting as a "hub" in the center, in the case of migration initiated by source this island will receive much more individuals than in the case of migration initiated by the destination. Or, one can imagine a situation where one island is much slower than its neighbors: here migration initiated by destination could mean that the neigbors would receive the same individuals several times, as the slow island would not manage to update its "database" in between communication events initiated by fast neighbors.

Another detail of the migration implementation that is not explicitly specified in our definition is the *method of distribution* of the migrating solutions. Migration topology $\mathscr{T}$ specifies which islands are adjacent and thus are allowed to exchange solutions. However when an island has more than one neighbor, there is plenty of different scenarios possible. One strategy could be to send the migrating individuals to all adjacent islands. On the other side of the spectrum of available choices, solutions could be sent only to one neighbor – for example selected randomly or in an algorithmic fashion (e.g. round-robin). It is likely that the choice will have an impact on the performance of the optimization process. It is reasonable to expect that in the former case, the effect of the choice of the migration topology should be more pronounced than in the latter. The frequency of communication between two particular islands in the latter case is lower, and also the total amount of information being exchanged in the whole archipelago is smaller, what may be significant for densely connected archipelagos with many islands. The method of distribution of individuals is moreover connected with other parameters of the island model, especially the migration-selection policy $\mathscr{S}$: a common strategy of migration is to allow the whole local population to be sent to the neighbors, but partitioned in such a way that each neighbor receives a distinct part of it [5].

2.2.2 Number of Islands n

The *number of islands* n is one of the most straightforward parameters of an archipelago. There may be many factors influencing the choice of n for a particular application: the computing platform used, the number of available computing machines, or the choice of the migration topology. Along with the progress in computing technology, this choice becomes now less and less constrained by the hardware. More and more researchers have access to large numbers of relatively powerful computers (for example on a university or company network) that could be used to perform distributed optimization in their idle time. Common sense prediction is that the more islands are involved in the computations, the better the final result of the optimization should be, as more computational resources are employed. It is reasonable to expect that using an archipelago with many islands will provide various benefits. For example this can remove the necessity (or at least reduce the importance of) fine-tuning of the parameters of the employed optimization algorithm. One can just create an archipelago in which different islands use different combinations of parameters and hope that islands with "good" parameters will drive the optimization process, compensating for the other, underperforming islands [11]. One should keep in mind however, that the bigger the number of islands, the more important the choice of the migration topology becomes. For instance, comparing archipelagos with 8 and 1024 islands, for a fully-connected topology this means nearly 20000 more connections between islands, while for the hypercube topology, the increase is only 500-fold.

2.2.3 Migration Topology $\mathscr{T}$

Another parameter of the archipelago is the *migration topology* $\mathscr{T}$. In the past, the migration topology was often dictated by the hardware on which the island model was implemented. Dedicated parallel machines without shared memory were used and the topology of the connections between available processors was thus used also when implementing an island model. Distributed computing architectures available today provide much greater flexibility, and the migration topology can be selected appropriately to the task at hand. The choice of the migration topology was the focus of a previous study [17] where the effect of the topology choice was related to the quality of the final results. A selection of migration topologies that can be found in the literature about the island model and other parallelization schemes is presented in figure 1. How to choose the migration topology for a given problem and optimization algorithm(s) remains an open question. Probably the most significant way in which the migration topology affects computation in the island model is shaping the information flow in the archipelago. From this point of view, most relevant parameters of the topology graph are: the average length of a path between two islands (or its diameter, i.e. the longest of the shortest paths between any two islands), the number of edges (which can be viewed as the measure of the communication overheads), and the degree of connectivity (average or maximum number of neighbors per island).

2.2.4 Migration Interval μ

The *migration interval* μ specifies the iterations of $\mathscr{A}$ before a migration occurs (an equivalent term sometimes found in the literature is *migration frequency*). The value of the migration interval should be chosen by taking into account the properties of the considered optimization algorithm $\mathscr{A}_i$, in particular, the convergence rate of the algorithm in relation to $\mathscr{P}$. It is reasonable to assume that the algorithm should be allowed to achieve a sensible progress in optimization in between two migration events. It has been shown that in certain cases too short migration interval may cause the algorithm to stop working as expected [21]. This is why the choice of the migration interval should be proceeded at least by experimental observation of the optimization progress on an isolated island. Another idea is to use a dynamically changing migration interval, and performing migration only after the algorithm achieves stagnation [4].

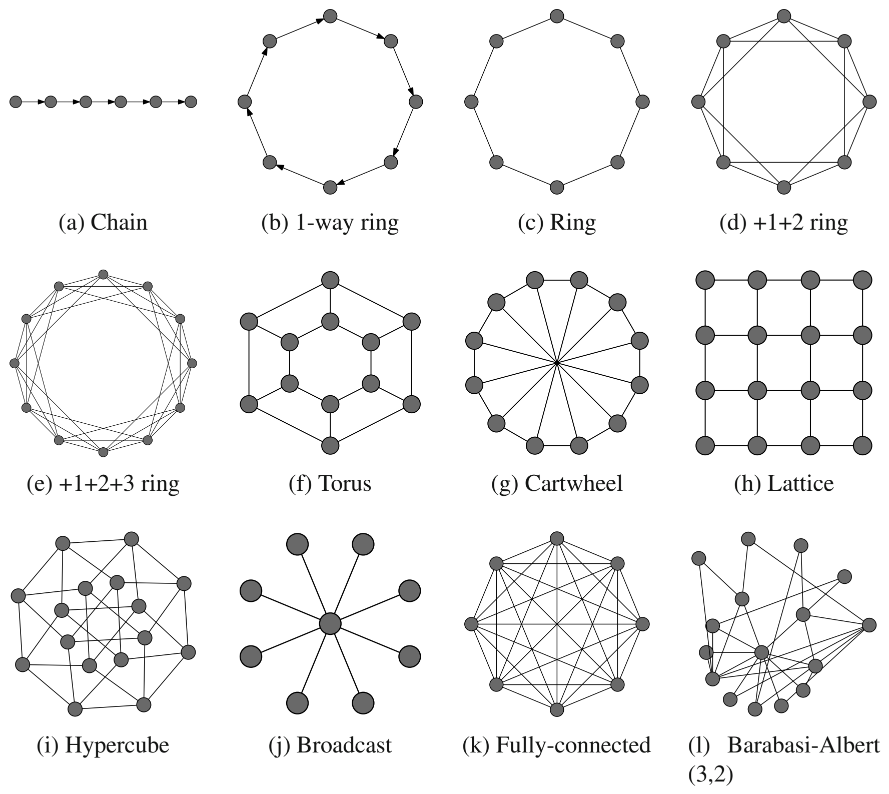

Fig. 1 Common migration topologies

2.2.5 Migration-Selection Policy $\mathscr{S}$ and Migration-Replacement Policy $\mathscr{R}$

The island model requires specification of two operations: the strategy of the *selection* of the solutions from the local population to be sent to adjacent island(s), as well as the method of *integration* of incoming individuals into the local population. In our generalised island model, these details are specified by the migration-selection policy $\mathscr{S}$ and migration-replacement policy $\mathscr{R}$. Note that among the details of selection and replacement there is the number of solutions to send or accept (usually called the *migration rate*), and that in principle $\mathscr{S}_i$ and $\mathscr{R}_i$ could differ from island to island. The spectrum of available reasonable choices for $\mathscr{S}$ and $\mathscr{R}$ could be narrowed down to *randomness* and *elitism* [18], however also more sophistcated mechanisms could be used, for example the Metropolis criterion.

3 The Code: PaGMO

All the experiments described in this chapter have been performed using a software platform for global optimization called PaGMO [3]. PaGMO has been developed within the Advanced Concepts Team of the European Space Agency, originally with special focus on spacecraft trajectory optimization problems, and later extended to be a general-purpose optimization framework.

The generalized island model is implemented within PaGMO using different threads of execution and a shared memory model, in order to take full advantage of contemporary multicore architectures. Each island executes the optimization algorithm in a separate thread, whose scheduling is managed asynchronously by the operating system (which will typically migrate the threads among the available computing cores as needed to maintain a balanced workload). Locking primitives such as mutexes, condition variables and thread barriers are used to avoid contention issues and manage access to resources shared among different threads. The parallelization of optimization tasks through the coarse-grained approach of the island model results in an "embarrassingly parallel" workload, since most of the CPU time is typically spent in the execution of the optimization algorithm in separate threads. Synchronization points, mostly due to the implementation of the migration operator, are sparse and lightweight, and as result PaGMO's performance scales linearly with the number of available cores.

Although not used in the experiments described in this chapter, PaGMO also has the capability of implementing the island model over a network of connected computers (e.g., a high-performance computing cluster) using the Message Passing Interface (MPI) [19]. In this operative mode, each island on the master node first serializes the objects representing the algorithm, the optimization problem and the population, and then transmits them over the network to another computer in the cluster, a slave node, where the optimization is actually executed in a local process. When the optimization process finishes, the new (optimised) population is serialised on the slave node and sent back to the master node, where it replaces the original population in the island. With respect to the multithreaded, shared memory version of the island model, this implementation incurs in the additional overheads and latencies of object serialization and network transmission. On the other hand, the ability to run on computer clusters greatly enhance the parallelisation potential which would otherwise be limited by the number of cores available on a single machine.

Written in C++ with the availability of Python bindings for increased ease of use and user-friendliness, designed with portability in mind and tested on GNU/Linux, OSX and Windows, PaGMO is free/libre/open-source software licensed under the GNU Public License. It can be downloaded from the website `http://pagmo.sourceforge.net`

4 Experiments

4.1 Problems

We consider some standard multimodal mathematical function with diverse properties (separability, etc.) together with "real world" problems. The problems have been selected to pose a real challenge to the algorithms studied and to maintain the overall CPU time for the entire test reasonable. The standard test functions we selected are: Rastrigin, Rosenbrock, Griewank, Ackley, De Jong, Levy5 (exact details on the implementation of these common mathematical functions can be found directly in the PaGMO code [3]. All problems are here instantiated with a dimension $d = 50$. We also consider two examples of the Lennard-Jones test function [23] corresponding to 11 and 17 atoms. We then extract some cases from the GTOP database [22], a European Space Agency's repository of difficult global optimization test functions related to interplanetary space travel. Out of such a database we have selcted the three problems named Rosetta, Cassini 2 and Messenger Full, also implemented and available in PaGMO [3].

4.2 Algorithms

We consider a number of global optimization algorithms based on diverse paradigms. Our selection is certainly not exaustive but it includes some of the arguably most popular algorithms that have proved their value extensively in the past decades.

1. DE: Differential Evolution – This algorithm by Storn and Price [20] has a rather standard implementation we here test. In particular we consider two variants of the algorithm commonly called rand/1/exp and rand/1/bin, that differ, essentially, in the crossover type (binomial or exponential). The algorithm parameters [20] are set to be $CR = 0.9$ and $F = 0.8$. We allow for 500 generations over a population of dimension 20 for each algorithmic call, corresponding to 2000 function evaluations.
2. PSO: Particle Swarm Optimization – The original algorithm proposed by Kennedy and Eberhart [13] was later improved by the introduction of a so called constriction factor [6]. We here consider this version of the algorithm together with the variant named Fully Informed Particle Swarm [15]. For both algorithms we select rather canonical values for the different coefficients [6]. In the canonical algorithm a ring topology with two neighbours is used, while for the FIPS we use a lattice topology, as suggested in [15].
3. SA: Simulated Annealing – There are many different implementations of simulated annealing available in the literature, in this work we consider the adaptive neighbourhood simulated annealing as introduced by Corana et al. [8]. The main algorithm parameters are the starting and final temperatures (T_s, T_f) and the number of total iterations n (these can be approximately be taken as the number of function evaluation made). Other parameter also control the performances of the we use $N_s = 20$ and $N_T = 1$, where N_s is the number of cycles and N_T the number of step adjustments (see Corana et al. [8]). The cooling schedule implemented is a geometric cooling schedule. For all problems we use $T_s = 0.1$, $T_f = 0.001$.
4. HS: Harmony Search – This metaheuristic algorithm [9] is inspired by the improvisation process of musicians. In the HS metaphor, each decision variable is seen as a musician which, through improvisation, generates new notes together

with the other musicians in order to find a good “harmony” (i.e., the global best). More specifically, in the HS algorithm new candidate solutions are generated either by randomly choosing components of existing solutions and adjusting them by increasing/decreasing their values, or by generating components of the candidate solution completely randomly. The parameters of the algorithm include the population size, the probability of generating new components by adjusting existing components instead of by random selection, and the amount of adjustment. In this study, we have adopted the canonical HS algorithm described in the original paper. To our knowledge, this is the first time that the island model paradigm is tested on a HS algorithm.

5. GA: Genetic Algorithm – This class of algorithms [10] is wide and contains many different variants that do not have an agreed common implementation. Here we consider a rather straight forward implementation with tournament selection, exponential crossover and elitism of one individual (i.e. the best among the parents is reinserted in the new generation substituting the worst of the offsprings if better). As for the mutation, we consider two different algoritmic variants, one with gaussian mutation (with the gaussian bell having a standard deviation of $\sigma = 0.1$ with respect to the width of the linear bounds on that particular component) and one having random mutation implemented. The crossover coefficient $c = 0.95$ and the mutation rate is $m = 0.05$ applied to each component of the cromosome. As to allow for a number of iterations (migration interval) $mu = 2000$ we let the
6. ABC: Artificial Bee Colony – This rather new metaheuristics still did not have the time to “mutate” much from the original version proposed by Karaboga [12] and we thus here consider that original algorithm. The parameters are the iteration number n which we set to $n = 50$ as to allow 2000 function evaluations for each algorithmic call, and the number of tries m after which a source of food is dropped if not improved. We here use $m = 20$.

For all algorithms we set the migration interval μ_i so that 2000 function evaluation are allowed in between migrations.

4.3 *Experiment #1: The Migration Dilemma*

Consider the pair $< \mathscr{P}, \mathscr{A} >$ containing an optimization algorithm $\mathscr{A}$ and an optimization problem $\mathscr{P}$. A common approach to use $\mathscr{A}$ to solve $\mathscr{P}$ is to run $\mathscr{A}$ indipendently N times over $\mathscr{P}$ and record, at the end, the best result found x. Alternatively, one could run N times $\mathscr{A}$ over $\mathscr{P}$ allowing solutions to be exchanged (migrate) among the N runs of $\mathscr{A}$ and and record, at the end, the best result found y. The two approaches require the exact same computational effort (i.e. it takes the same CPU time to get y or x) if one neglects the overhead of migrating and exchanging information among algorithmic runs (i.e. CPUs) Accumulating statistical evidence on the difference between x and y helps us in evaluating the benefits of the generalized island model and to answer the simple question “should I migrate?” and thus will be the focus of the results here presented. For each pair $< \mathscr{P}, \mathscr{A} >$

and for $N = 8, N = 32$ we build samples (containing 200 instances) of the stochastic variables x and y at different points along the process. We indicate these samples with bold symbols $\mathbf{x}_i$ and $\mathbf{y}_i$, where $i = 1 \dots 30$ indicates at point of the process the value is recorded. In other words:

1. (Without Migration - unconnected topology) We run the algorithm independently N times (on different CPUs) and we record in the variable x_i, $i = 1 \dots 30$ the best solution found across the N algorithmic runs each 2000 function evaluation
2. (With Migration - two way ring topololgy) We run the algorithm N times (on different CPUs) and we record the best solution found across the N algorithmic runs each 2000 function evaluation in the variable y_i, $i = 1 \dots 30$ when we also let solutions migrate along a two-way ring topology. Thus the index i can be seen as the number of migrations occured and will so be interpreted in the following.

At the end of this process we have the samples $\mathbf{x}_i$ and $\mathbf{y}_i$, $\forall i = 1 \dots 30$ each containing 200 instances of the stochastic variables x_i and y_i.

We first detect if there is any statistical difference in the samples for $i = 30$ (i.e. after the very last migration), in case there is none we set $i = i - 1$ and repeat the test until a difference is found. When a dfference is detected at $i = m$ we sort the solutions in $\mathbf{x}_m$ and $\mathbf{y}_m$ and compare the best 60. If they all but one are better in one sample we conclude that that solution strategy is better, otherwise we try again with $i = i - 1$. If we get to $i = 1$ without having found any result we conclude that there is not enough statistical evidence to choose between the two solution strategies for the particular couple $< \mathscr{P}, \mathscr{A} >$ under consideration.

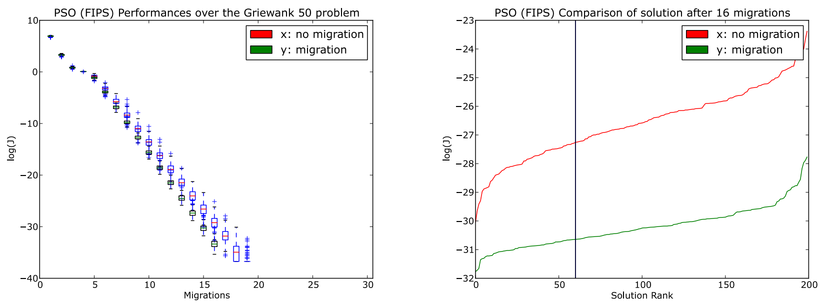

Fig. 2 Comparison in the case of PSO (FIPS) applied to Griewank 50

Example: Assume the problem $\mathscr{P}$ is Griewank 50 and the algorithm $\mathscr{A}$ is a 100 generation Fully Informed Particle Swarm algorithm (PSO FIPS) with population size of 20. In Fig. 2, on the left, we show, after each of the 30 migrations, the box plot in logarthmic scale of the samples $\mathbf{x}_i$ and $\mathbf{y}_i$. Clearly at $i = 30$ there is no difference between the results, as in both cases the global optimum of the problem (i.e. J=0) has been found. Proceeding backwards, we then consider $i = 16$, where there actually is a statistical difference between the two samples. At that stage of the runs, we show in Fig. 2, on the right, the ranked solutions. As 59 out of the best 60 solutions of the sample $\mathbf{x}_{16}$ are better than the best 60 solutions of sample $\mathbf{y}_{16}$ we conclude that for this pair $< \mathscr{P}, \mathscr{A} >$ migration is actually better.

NOTE: in this example the comparison is quite trivial and a visual inspection of the results is enough to conclude, however there are cases where a mathematical formalization of the comparison is necessary to decide conclusively whether migration helped or not.

In the method outlined above, one must be very careful when detecting the statistical difference between the samples (in the exmaple it is rather obvious, but things do get much more complicated). Here we use a method based on random resampling, essentially a jacknifing method [7]. The use of this method is not too common within the global optimization community and we thus briefly explain it in the following. We create a new sample $\mathbf{r}$ joining $\mathbf{x}$ and $\mathbf{y}$. We then randomly create two disjoint new samples of size equal to the original $\mathbf{x}$, $\mathbf{y}$ out of $\mathbf{r}$ and we evaluate, for these two samples, the difference d in a chosen statistics (in our case the mean). We repeat this process $n = 10000$ times, thus building an experimantal distribution of d for two samples of equal size extracted at random from $\mathbf{r}$. We then evaluate, according to the built distribution, the probability of d being as big as the one calcualted from the two original samples $\mathbf{x}$ and $\mathbf{y}$ and we use this as our confidence level that the two distribution actually are different. Only if this is larger than 99.97% we conclude that they indeed are. Note that if $p = 0.9997$ approximate the probability that the two samples come from different random processes, our actual confidence level that the whole optimization process is different is smaller as we need to compare the samples 30 times (after each migration). Thus, our true confidence level will be $p^{30} = 0.99$.

4.3.1 Results

We apply the methodology described above to approach "the migration dilemma" for each possible problem-algorithm couple $< \mathscr{P}, \mathscr{A} >$ one can form starting from the algorithms and problems introduced. We repeat the same experiment using $N = 8$ and $N = 32$ islands. This results in a total of 240 experiments. Simple calculation show that for each one of those where $N = 8$, $24,000,000$ objective function evaluations are performed, while for $N = 32$ the number of function evaluation per experiment is $96,000,000$ (we report these number with the sole purpose of giving a feel for the amount of computations involved in our tests). These allow the construction of our statistical samples $\mathbf{x}_i$ and $\mathbf{y}_i$. Making use of the inherent parallelization offered by PaGMO, the 240 experiments take roughly 96 hours to complete on a linux gentoo system installed on top of a dual quad core OSX XServe machine.

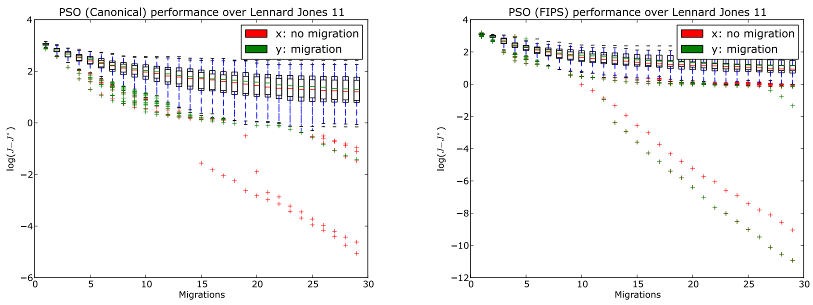

Fig. 3 Two non obvious cases

The results are summarized in Table 1 where we report, for each experiment, the outcome of our comparison criteria. In reporting only this final information we are synthezising in a single table a very complex procedure producing a great amount of data, thus many of the details on what is actually happening in a case to case basis are inevitabley lost. As an example, we here only investigate in more detail what happens in the case of $N = 8$ for the pairs, Lennard-Jones 11, PSO (canonical) and PSO (FIPS). This case is interesting as our comparison concludes that migrating is actually harmful in the case of PSO, (FIPS) and undecidable in the case of the PSO (canonical), a strange conclusion that is worth further investigation. A closer look to the data reveal more details on the rationale for such a conclusion. In Figure 3 we report the boxplots for $\mathbf{x}_i$ and $\mathbf{y}_i$. In the case of PSO (canonical) we observe how after some migrations one lucky trial, not using migration, finds the optimal solution and is shown as an oulier (or flier) in the boxplot. For this particular experiment, this fact and the generally lower median of the sample relative to the optimization without migration (see the median red line in the boxplot) is not considered as no statistical differenc is found between the samples at any of the migration steps. For this reason our comparison criteria does not choose among the two approaches. Looking at the other case, the one considering the PSO (FIPS) algorithm, we note from Figure 3 that even though a lucky shot finds the optimal solution in the case of migration (shown as an outlier, or flier, in the boxplot), our comparison criteria concludes that not migrating is actually better for this case and gives to the unconnected topology the "winner" cup. This clearly implies that there is statistical significance between the samples at some point during the optimization and that, except that lucky run, the other 59 best solutions found in the case of the unconnected topology are actually better than the ones found by migrating (as evaluated at the point where statistical difference is found between the samples). Other cases are much easier to judge (as shown for example in Figure (2) and here we picked up one of the most trouble-some cases in order to show how our comparison criteria is able to make intelligent choices also in difficult situations where, probably, also humans would argue on what conclusion to reach.

Table 1 Generalized Island Model Results: migration occurs every 2000 function evaluations, algorithms are stopped after 30 migrations, samples are made of 200 instances. M = Migration outperforms non migration. U = Non migration outperforms migration, - = no conclusion is possible as the results are not significantly different

	8 Islands										32 Islands									
	PSO (Canonical)	PSO (FIPS)	Corana's SA	HS	HS (Improved)	GA (Gaussian)	GA (Random)	DE (rand/1/exp)	DE (rand/1/bin)	Artificial Bee Colony	PSO (Canonical)	PSO (FIPS)	Corana's SA	HS	HS (Improved)	GA (Gaussian)	GA (Random)	DE (rand/1/exp)	DE (rand/1/bin)	Artificial Bee Colony
Griewank ($d = 50$)	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
Rastrigin ($d = 50$)	-	U	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
Rosenbrock ($d = 50$)	U	M	M	M	M	M	M	M	M	M	U	U	M	M	M	M	M	M	M	M
Ackley ($d = 50$)	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
De Jong ($d = 50$)	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
Levy 5 ($d = 50$)	M	U	M	M	M	M	M	M	M	M	M	U	M	M	M	M	M	M	M	M
Schwefel ($d = 50$)	M	U	M	M	M	M	M	M	M	M	M	U	M	M	M	M	M	M	M	M
Lennard-Jones (11 Atoms)	M	U	M	M	M	M	M	M	M	M	M	U	M	M	U	M	M	M	M	M
Lennard-Jones (17 Atoms)	-	U	M	M	M	M	M	M	M	M	M	U	M	M	M	M	M	M	M	M
Rosetta	-	-	-	M	M	M	-	M	M	M	-	-	-	M	M	M	M	M	M	M
Cassini 2	-	U	-	M	-	M	-	M	M	M	-	U	-	U	M	M	M	M	M	M
Messenger Full	-	U	-	M	U	M	M	M	M	M	-	U	M	M	U	M	M	M	M	M

Let us now take a look at Table 1. We may observe a number of things that appear quite strongly from the reported data. We list them in the following as they all are observation worth further studies.

- Not surprisingly (the no free lunch theorem applys here) the answer to the question "should I migrate or not?" is "it depends from the algorithms and the problem".
- As a general rule-of-thumb, our generalized migration operator help algorithms find better solutions.
- PSO (and in particular for its FIPS variant) is an exception to the above rule being unable to take consistently advantage of our generalized migration operator.
- Problems such as Griewank, Ackley and De Jong are solved more efficiently making use of migration, regardless of the employed algorithms.
- Increasing the problem complexity, it is more difficult to find statistically significant results using a sample size of 200, as shown by the higher number of inconclusive tests for problems such as Lennar-Jones, Rosetta, Messenger Full and Cassini.
- Increasing the number of islands the conclusions do not change. Statistical significance is easier to be found in the comparisons made using a higher number of islands.

Table 2 Heterogeneous tests results: parameters as in the previous experiment. Number in every cell reports how many of the 5 heterogeneous archipelagos performed better than best of the two homogeneous algorithms from experiment 1 (with or without migration) for the given problem and algorithm.

	8 Islands										32 Islands									
	PSO (Canonical)	PSO (FIPS)	Corana's SA	HS	HS (Improved)	GA (Gaussian)	GA (Random)	DE (rand/1/exp)	DE (rand/1/bin)	Artificial Bee Colony	PSO (Canonical)	PSO (FIPS)	Corana's SA	HS	HS (Improved)	GA (Gaussian)	GA (Random)	DE (rand/1/exp)	DE (rand/1/bin)	Artificial Bee Colony
Griewank ($d = 50$)	4	0	5	5	5	5	5	5	5	5	0	0	5	5	5	5	5	5	5	5
Rastrigin ($d = 50$)	5	5	4	5	5	5	5	4	5	5	5	5	5	5	5	5	5	5	5	5
Rosenbrock ($d = 50$)	1	3	5	5	5	5	5	5	5	1	0	5	2	5	5	5	5	5	5	0
Ackley ($d = 50$)	4	0	5	5	5	5	5	5	5	5	0	0	5	5	5	5	5	5	5	5
De Jong ($d = 50$)	4	0	5	5	5	5	5	5	5	4	0	0	5	5	5	5	5	5	5	5
Levy 5 ($d = 50$)	5	5	0	5	5	4	5	0	5	5	5	5	2	5	5	5	5	3	5	5
Schwefel ($d = 50$)	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
Lennard-Jones (11 Atoms)	5	5	2	5	5	5	5	1	1	5	5	5	5	5	5	5	5	0	1	5
Lennard-Jones (17 Atoms)	5	5	0	5	5	5	5	0	5	5	5	5	0	5	5	5	5	0	5	5
Rosetta	2	0	5	5	5	5	5	0	0	5	5	3	5	5	5	5	5	0	0	5
Cassini 2	2	5	5	5	0	5	5	0	3	5	5	5	5	5	5	5	5	0	5	5
Messenger Full	0	0	5	5	4	5	5	0	0	5	0	5	5	5	5	5	5	0	0	5

4.4 *Experiment #2: Can They Cooperate?*

In the previous section we have established that the influx of migrants help populations in islands to converge faster to good solutions for the majority of the evolution startegies tried. We now ask ourself the question: does it matter whether these migrants come from populations being evolved with the same paradigm? In other words: does it help to evolve populations using different algorithms while still exchanging migrants among them? In order to answer this question we compared the results obtained in the previous section (where we used archipelagos of 8 and 32 homogeneus islands, i.e. islands containing the very smae algorithm) with the result obtained using archipelagos of 8 and 32 island containing heterogeneous algorithms. As we have determined that PSO is, as an exception, not taking advantage of our generalized migration operator, we do not allow migrants to islands containing PSO, only from. Different heterogeneous archipelagos can be created out of mixing the 10 different algorithm instances considered, thus we perform our comparison with respect to five archipelagos containing different permutations of the chosen algorithms (a simple round robin startegy is implemented to select the algorithms to instantiate in the different islands). These are then compared pairwise (see the previous section for the comparison criteria) to the corresponding homogeneous archipelago. In Table 2 we report for each pair problem-algorithm the number of heterogeneous archipelagos that performed better than the corresponding homogeneous ones (using migration or not according to the best choice outlined in Table 1).

One would expect that the heterogeneous archipelago performs either a) as good as or b) worse than an homogeneous archipelago having the best algorithm across all islands. This is infact the case, as an example, for the problem Griewank. In this case PSO FIPS is the best performing algorithm and no heterogeneous archipelago (also the ones containing islands with PSO FIPS) can perform better. The surprising result comes from problems such as Rastrigin or Schwefel, where we can conclude that the cooperation between different algorithms via our generalized operator creates defacto a new meta-algorithm that improves over the performance of all its algorithmic components.

5 Conclusions

We propose a generalization of the island model to obtain a coarsed grain parallelization strategy valid across global optimization algorithms. The new model, essentially, allows for information exchange among different instances of algorithms. We show how, for algorithms such as Particle Swarm optimization, Differential Evolution, Simulated Annealing, Bee Clolony Search, Harmony Search and Genetic Algorithms such an information exchange is indeed beneficial in a large number of cases. We then test the same migration strategy across heterogeneous algorithms to study whether solutions generated by one algorithm could improve over the performance of a second algorithm and vice versa. We find that on some problems, migration among a set of heterogenous algorithms allows for a better search than all possible set ups where migration occur among different instances of the same algorithm.

References

1. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. IEEE Transactions on Evolutionary Computation 6(5), 443–462 (2002)
2. Aydin, M.E., Yiğit, V.: Parallel simulated annealing. Wiley Online Library (2005)
3. Biscani, F., Izzo, D., Yam, C.H.: A global optimisation toolbox for massively parallel engineering optimisation. In: International Conference on Astrodynamics Tools and Techniques - ICATT (2010)
4. Braun, H.: On Solving Travelling Salesman Problems by Genetic Algorithms. In: Schwefel, H.-P., Männer, R. (eds.) PPSN 1990. LNCS, vol. 496, pp. 129–133. Springer, Heidelberg (1991)
5. Cantú-Paz, E.: Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers, Norwell (2000)
6. Clerc, M., Kennedy, J.: The particle swarm explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation 6(1), 58–73 (2002)

7. Cohen, P.R.: Empirical methods for artificial intelligence, vol. 55. MIT press (1995)
8. Corana, A., Marchesi, M., Martini, C., Ridella, S.: Minimizing multimodal functions of continuous variables with the ”simulated annealing” algorithm Corrigenda for this article is available here. ACM Transactions on Mathematical Software (TOMS) 13(3), 262–280 (1987)
9. Geem, Z.W., Kim, J.H., Loganathan, G.V.: A new heuristic optimization algorithm: Harmony search. SIMULATION: Transactions of The Society for Modeling and Simulation International 78, 60–68 (2001)
10. Goldberg, D.E.: Genetic algorithms in search, optimization, and machine learning. Addison-wesley (1989)
11. Izzo, D., Rucinski, M., Ampatzis, C.: Parallel global optimisation meta-heuristics using an asynchronous island-model. In: IEEE Congress on Evolutionary Computation, CEC 2009, pp. 2301–2308. IEEE (2009)
12. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of Global Optimization 39(3), 459–471 (2007)
13. Kennedy, J., Eberhart, R.C.: Particle Swarm Optimization. In: Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, vol. 4, pp. 1942–1948. IEEE Press (1995)
14. Konfrst, Z.: Parallel genetic algorithms: advances, computing trends, applications and perspectives. In: Proceedings of the 18th International Parallel and Distributed Processing Symposium 2004, p. 162. IEEE (2004)
15. Mendes, R., Kennedy, J., Neves, J.: The fully informed particle swarm: simpler, maybe better. IEEE Transactions on Evolutionary Computation 8(3), 204–210 (2004)
16. Price, K.V., Storn, R.M., Lampinen, J.A.: Differential evolution. Springer, Berlin (2005)
17. Ruciński, M., Izzo, D., Biscani, F.: On the impact of the migration topology on the Island Model. Parallel Computing 36(10-11), 555–571 (2010)
18. Schwehm, M.: Parallel population models for genetic algorithms (1996)
19. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI: The Complete Reference. MIT Press, Cambridge (1995)
20. Storn, R., Price, K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11(4), 341–359 (1997)
21. Tanese, R.: Distributed genetic algorithms. In: Proceedings of the 3rd International Conference on Genetic Algorithms, pp. 434–439. Morgan Kaufmann Publishers Inc., San Francisco (1989)
22. Vinkó, T., Izzo, D.: Global optimisation heuristics and test problems for preliminary spacecraft trajectory design. Technical Report GOHTPPSTD, European Space Agency, the Advanced Concepts Team (2008)
23. Wales, D., Doye, J.: Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. Arxiv preprint cond-mat/9803344 (1998)