Parallel global optimisation meta-heuristics using an asynchronous island-model

Dario Izzo, Marek Ruciński and Christos Ampatzis

Abstract— **We propose an asynchronous island-model algorithm distribution framework and test the popular Differential Evolution algorithm performance when a few processors are available. We confirm that the island-model introduces the possibility of creating new algorithms consistently going beyond the performances of parallel Differential Evolution multi starts. Moreover, we suggest that using heterogeneous strategies along different islands consistently reaches the reliability and performance of the best of the strategies involved, thus alleviating the problem of algorithm selection. We base our conclusions on experiments performed on high dimensional standard test problems (Rosenbrock 100, Rastrigin 300, Lennard Jones 10 atoms), but also, remarkably, on complex spacecraft interplanetary trajectory optimisation test problems (Messenger, Cassini, GTOC1). Spacecraft trajectory global optimisation problems have been recently proposed as hard benchmark problems for continuous global optimisation. High computational resources needed to tackle these type of problems make them an ideal playground for the development and testing of high performance computing algorithms based on multiple processor availability.**

I. Introduction

The implementation of global optimisation meta-heuristic algorithms on multiple processor platforms is an active research field that is increasingly gaining interest as the trend in processor development is drifting steadily from faster single-core to multi-core technology. At the same time, access to high performance computing platforms, be it either single machines composed of tens or even hundreds of cores, or thousands of machines distributed over the internet, is progressively getting easier and cheaper. These trends apart from ushering the research community into a new realm of possibilities, introduce challenges related to the emergence of a novel programming paradigm whose implications are deep and still need to be fully understood.

Parallel implementations of algorithms have been proven to achieve the intuitive speed-up with respect to task completion time due to the distribution of the workload to several processing elements operating in parallel. Interestingly, certain parallel implementations seem to also contribute to improving the quality of the solution obtained for a given overall amount of work [1]. In this paper we describe a generic parallelisation framework for global optimisation meta-heuristics and we study its performance against several standard large-scale test functions and other, less standard test problems, rooted in spacecraft global trajectory optimisation. The latter problems can be considered complex testbeds for state-of-the-art optimisation algorithms, due to the high problem complexity and computational requirements. Typically, for such problems, the global optimum is not known in advance.

All authors are with the Advanced Concepts Team of the European Space Agency (contact email: dario.izzo@esa.int). Marek Ruciński is also supported by Poznań University of Technology

II. Parallelisation strategy

The choice of the particular distribution strategy adopted when parallelising optimisation algorithms is very important and ultimately affects the speed-up, the obtained results and the scalability of the architecture [2]. The distribution strategy we study in this paper is based on the island-model paradigm for distributed evolutionary algorithms (see [3] for a first classification of different parallelisation strategies). This model typically assumes a global population that is subdivided in subpopulations, which occasionally communicate between them, with a process called migration. Research in this field is typically looking at the role of migration rates, the number of islands (i.e., subpopulations), and the nature of the communication network (network connectivity and topology) in the optimisation process. The island-model, inspired by the theory of punctuated equilibria introduced by Eldredge and Gould in 1972 [4], has been proven to be an efficient parallelisation strategy for genetic algorithms (GAs), that introduces—de facto—a new algorithm achieving superior performances with respect to the sequential version [1]. More recently, Tasoulis et al. [5] applied the island-model to the parallelisation of differential evolution (DE) [6], reporting on reduced computational time and improved performance for a series of low-dimensional optimisation problems. Apolloni et al. [7] showed that their implementation of a distributed DE outperforms the CEC'05 state-of-the-art algorithms for rather high dimensional problems.

Our approach has the ambition to serve as a generic framework for distributing heterogeneous meta-heuristic global optimisation algorithms. In particular, we implement an island-model coarse-grained parallelisation strategy, where islands can be running the same or a different meta-heuristic algorithm. The motivation behind this choice lies in the fact that different algorithms have different convergence and search properties; combining these algorithms appropriately could potentially lead to better solutions in shorter time. This is an open research question we want to address by proposing a novel distributed strategy. In this article, we report only on results obtained by using different variants of the DE algorithm [6]. However, any different optimisation algorithm (e.g., Genetic Algorithm, Particle Swarm Optimisation, Adaptive Simulated Annealing, etc.) can be deployed in an island and thus its ability to cooperate with the

other algorithms can be assessed. The implementation chosen uses asynchronous communication between the islands. The reason for this choice is twofold: (i) recent research works suggested that the use of an asynchronous implementation of the island-model is advantageous with respect to its synchronous counterpart in terms of result quality [8]; and (ii) an asynchronous implementation is more intuitive and suitable for distributed computing over TCP/IP where resources might become available/unavailable any time.

We hereby detail the parallelisation strategy we have implemented. Each different processor is assigned a population $\mathcal{P}_i$ of individuals and a global optimisation algorithm $\mathcal{A}_i$ that will evolve the population for a number of generations.[1] Whenever a CPU terminates $\mathcal{A}_i$, the best individual in $\mathcal{P}_i$ migrates with probability ϕ toward another node of the network. We use a ring network topology to implement such a migration operator. When all the algorithms $\mathcal{A}_i = \mathcal{A}$ are equal and $\phi = 0$ this architecture is equivalent to running in multi-start $\mathcal{A}$ over different processors. To implement this algorithm in a purely asynchronous fashion (i.e. no synchronization between different evolutions is required and the island can thus perform migrations at different times) we use one CPU as a central server and the other CPUs as clients.

The operation of the server and the clients is described by the following pseudo-algorithms:

The server:

1: initialise all $\mathcal{P}_i$
2: for each CPU
3: start a new optimisation on the CPU using $\mathcal{A}_i$ and $\mathcal{P}_i$
4: while not *convergence-criterion*
5: wait for any CPU to become available
6: update $\mathcal{P}_i$ with received $\mathcal{P}'_i$
7: perform migration with probability ϕ
8: start a new optimisation on the CPU using $\mathcal{A}_i$ and $\mathcal{P}_i$

and the clients:

1: receive $\mathcal{P}_i$ and $\mathcal{A}_i$
2: evolve $\mathcal{P}_i$ using $\mathcal{A}_i$, obtaining $\mathcal{P}'_i$
3: return $\mathcal{P}'_i$ to the server
4: signal availability

The *convergence-criterion* (step 4 of the server pseudo-algorithm) may be different for different global optimisation problems. In the experiments described in this paper, we used two different criteria depending on the test problem considered. In the first three experiments, the computation is stopped after a solution with an objective function value close enough to a putative optimal value is found. In this case, the number of performed objective function evaluations and the frequency of success (i.e. the global optimum is reached) will measure the algorithm performance. In a second set of three experiments, where the test problems considered are more difficult and solutions take longer to be found, the algorithm is allowed to perform a fixed number of objective function evaluations. In these cases, the algorithms' performances are quantified by the quality of the obtained solution.

After a client CPU finishes the evolution of the population $\mathcal{P}_i$, the resulting population $\mathcal{P}'_i$ is used to update the individuals in the original population $\mathcal{P}_i$ (step 6 of the server pseudo-algorithm). A greedy strategy of performing the update has been implemented – if an individual from the population $\mathcal{P}'_i$ has a better objective function value than the corresponding individual from $\mathcal{P}_i$, the former replaces the latter in the next generation of $\mathcal{P}_i$.

After the population $\mathcal{P}_i$ is updated, a migration of the best individual from that population to the target population is performed with probability ϕ (step 7 of the server pseudo-algorithm). The migration is assumed to be an unconditional replacement of a randomly selected individual in the target population with the migrating individual.

III. Implementation using the POSIX Threads library

The proposed algorithm has been implemented in C++ programming language, using the POSIX Threads application programming interface [9]. The library has been chosen because it provides a good trade-off between the robustness, portability, complexity and ease of use when compared to other possible choices, like operating system-specific mechanisms, PVM, MPI and OpenMP.

In the algorithm implementation, the server tasks are performed by the main program, while client tasks are realised by an appropriate number of concurrently running *threads*. Communication between the server and the threads has been achieved using the shared memory model. As in every other case of the multithreaded programming with shared memory, the following issues had to be addressed:

- synchronization between the server and the client threads;
- mutual exclusion of access to the shared memory area.

In the proposed algorithm, client-server synchronisation involves mainly the notification of the server about the completion of the client threads (step 5 of the server pseudo-algorithm). In order to achieve this goal, a *condition variable* mechanism, provided by the POSIX Threads library, has been used.

The shared memory mechanism is used to exchange data between the clients and the server, that is for:

- obtaining a copy of the population to evolve by the client (step 1 of the client pseudo-algorithm);
- updating the population after the optimisation is completed (step 6 of the client pseudo-algorithm);
- performing the migration (step 7 of the client pseudo-algorithm).

[1] We use the term "evolution"and "population" somehow improperly as, depending on the algorithm $\mathcal{A}_i$ the applied paradigm can be other than that of Darwinian evolution. We will thus refer to population also in the case, for example, of Particle Swarm Optimisation or Adaptive Simulated Annealing, where other terms are typically used.

Operations mentioned above must be performed in a thread-safe manner, that is, in a way that guarantees that at any point in time no more than one thread (including the server) is accessing the shared memory. This is achieved by guarding the shared memory area using the *mutex* mechanism, also provided by the POSIX Threads library. Access to the shared memory area is allowed only when an ownership of the mutex object is granted to the thread performing the access. The library guarantees that no more than one thread at a time is granted access to this object. Special care is also taken to ensure that all third-party libraries and modules used by the application are also thread-safe. including the implementation of the random number generator.

IV. Test problems

The test problems we use here to assess proposed computation framework are taken from the global optimisation literature and constitute an unusual mixture of rather standard large-scale problems and difficult engineering problems taken from the spacecraft global trajectory optimisation literature.

A. Rosenbrock 100

This test function has the mathematical representation:

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2\right]$$

and admits the global optima $f^* = 0$ at $x_i = 1, \forall i = 1, ..., 100$. We use the search domain $-5 < x_i < 10$ and the dimension $n = 100$.

B. Schwefel 300

This test function has the mathematical representation:

$$f(\mathbf{x}) = Vn - \sum_{i=1}^{n} \left[x_i \sin \sqrt{|x_i|}\right]$$

and admits the global optima $f^* = 0$ at $x_i = 1, \forall i = 1..n$ for $V = \sin 1$. We use the search domain $-500 < x_i < 500$ and the dimension $n = 300$.

C. Lennard-Jones 10

This test function expresses the total Lennard-Jones potential [10] in a cluster of N atoms through the expression:

$$V = 4 \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left(\frac{1}{r_{ij}^{12}} - \frac{1}{r_{ij}^{6}}\right)$$

where r_{ij} represents the distance between the i-th and the j-th atom. Cartesian coordinates are used to express the absolute atom positions and fundamental symmetries are used to reduce the problem dimension to $3N - 6$. We use the search domain $-3 < x_i < 3$ and the $N = 10$ atoms case resulting in a problem dimension of $n = 27$. This particular problem instance admits the putative global optima $f^* = -28.422532$ as reported in [11], where solutions to all problem instances, obtained by a monotonic basin hopping algorithm, are presented up to $N = 110$ atoms.

D. Spacecraft trajectory problems

This class of test functions for global optimisation has been recently introduced as such by Vinkó et al. [12] in 2006. They express a spacecraft trajectory performance index (often the final spacecraft mass at arrival) as a function of the strategy adopted to reach the mission goals (e.g. launch date, thrusting location and magnitude, fly-by sequence etc.). A large number of different test function instances can be derived from a rather general description, by specifying different goals or possible strategies (a rendezvous mission to Jupiter will look radically different from a fly-by mission to Pluto, depending on the spacecraft propulsion system adopted, the launcher used or the launch window). It is beyond the scope of this paper to give the exact expression of each function; instead, we just stress here that both its minimal value and its taxonomy vary considerably by changing the function domain, the planet ephemerides and a large number of parameters values (e.g. the strength of the Sun gravity). Thus, it is extremely important, when performing any test, to use standardized instances of the problem. In our experiments, we used some of the problem instances posted in the Advanced Concepts Team web pages (www.esa.int/gsp/ACT/inf/op/globopt.htm) together with the C++ code there provided [12]

1) Messenger and Cassini2 problems: A first important subgroup of spacecraft trajectory problems is the "simple MGA-DSM" (Multiple Gravity Assists with one Deep Space Maneuver). In these instances, a spacecraft is required to perform a given mission in the interplanetary medium using impulsive thrusters that are allowed to fire only once between two subsequent planets of a preassigned fly-by sequence. This creates an unconstrained global optimisation problem, as far as further constraints are not explicitly specified. In the case of the Messenger problem, the target planet is Mercury and the overall planetary sequence is Earth-Earth-Venus-Venus-Mercury resulting in a problem dimension $D = 18$. The mission target is to rendezvous with Mercury with the highest possible final spacecraft mass. This problem instance is rather academic, as multiple resonant fly-bys at Mercury are known to be able to substantially increase the performance index. In the case of Cassini2, the target planet is Saturn and the assigned planetary sequence is Earth-Venus-Venus-Earth-Jupiter-Saturn resulting in a problem dimension $D = 22$. The mission goal is the insertion around Saturn in an orbit similar to the orbit the Cassini spacecraft achieved in 2004. This problem instance reproduces quite accurately a real preliminary mission design problem.

2) GTOC1 problem: Another important subgroup of spacecraft trajectory problems is the "MGA" (Multiple Gravity Assists). In these instances, a spacecraft is required to perform a given mission in the interplanetary medium using impulsive thrusters that are allowed to fire only at the arrival planet or at the pericenter of the planetocentric hyperbolae. This creates a constrained global optimisation problem where constraints need to be put on the planetocentric hyperbolas pericenters as to impose a safe distance from the planet.

Such a problem was proven to allow a very efficient space pruning strategy[2] resulting in algorithms such as GASP [14], that essentially are able to solve the problem using a good amount of problem knowledge. In the case of the GTOC1 problem, the target body is the asteroid TW-229 and the overall imposed planetary sequence is Earth-Venus-Earth-Venus-Earth-Jupiter-Saturn-TW229 resulting in a problem dimension $D = 8$. The mission target is to maximise the asteroid semi-major axis change due to the final inelastic impact of the spacecraft with the asteroid. This problem instance is related to the 1st Global Trajectory Optimisation Competition (GTOC1) [15] and is a possible first step to arrive to the final solution of the competition problem.

V. Algorithms

The proposed parallelisation strategy was used to run different variants of the Differential Evolution algorithm. The original version of the algorithm was first described in [6], and since then it has been successfully applied to many problems in different frameworks, including parallel (see for instance [5], [16], [7]). Because the algorithm is very well described in already existing literature, we provide only a brief description of its five variants which have been used in experiments discussed in this paper.

The distributed framework described in section II has been tested with the following variants of the Differential Evolution algorithm (using the notation introduced in [6]):

- $DE/best/1/exp$, further simply referred to as DE1;
- $DE/rand/1/exp$, further referred to as DE2;
- $DE/rand{-}to{-}best/1/exp$, further referred to as DE3;
- $DE/best/2/exp$, further referred to as DE4;
- $DE/rand/2/exp$, further referred to as DE5.

Every iteration of the Differential Evolution algorithm consists of three phases: mutation, crossover and selection. During the mutation phase, for every individual in the population, a so-called mutant individual is calculated. For the five DE variants, the formulae used to calculate the mutant $v_{i,G+1}$ of the i-th individual used to create the generation $G + 1$ are the following:
For DE1:

$$v_{i,G+1} = x_{\star,G} + F \cdot (x_{r_2,G} - x_{r_3,G});$$

for DE2:

$$v_{i,G+1} = x_{r_1,G} + F \cdot (x_{r_2,G} - x_{r_3,G});$$

for DE3:

$$v_{i,G+1} = x_i + F \cdot (x_i - x_{\star,G}) + F \cdot (x_{r_1,G} - x_{r_2,G});$$

for DE4:

$$v_{i,G+1} = x_{\star,G} + F \cdot (x_{r_1,G} + x_{r_2,G} - x_{r_3,G} - x_{r_4,G});$$

and for DE5:

$$v_{i,G+1} = x_{r_5,G} + F \cdot (x_{r_1,G} + x_{r_2,G} - x_{r_3,G} - x_{r_4,G}),$$

where G is the iteration number, $x_{\star,G}$ is the best individual found up to iteration G, r_{1-5} are 5 different, randomly selected indices of individuals, and $F > 0$ is the constant amplification factor (one of the algorithm parameters).

All these five variants of the Differential Evolution algorithm use the exponential crossover strategy. As in all variants of DE described in [6], the selection method resembles the greedy approach – a new individual replaces the old one in the population if and only if it has a better value of the objective function.

In the experiments, the following values of the Differential Evolution algorithm parameters were assumed:

- Amplification factor $F = 0.8$;
- Crossover constant $CR = 0.8$;
- Population size $NP = 20$.

Values of these parameters were not fine-tuned because the goal was to compare the performance of the proposed distributed framework against sequential versions of the algorithms, rather than obtaining state-of-the-art solutions for covered optimisation problems.

VI. Experiment results

In this section, we report the results we obtained in the two different sets of experiments introduced in the previous section. In the first set we consider three "easy" problems: Rosenbrock (dimension 100), Schwefel (dimension 300), Lennard-Jones (10 atoms, dimension 24). We first test the performance of different algorithms in their standard sequential version and then we test the performance of distributing the different algorithms in our asynchronous implementation of the parallel island model. In this first set of experiments we stop the search when the algorithm arrives to the known global optima or if it performs more than 2,000,000 function evaluations. We record the number of successes (global optima reached) and the average number of function evaluations performed in all successful trials. Algorithms are referred to using designations introduced in the previous section. The naming for the parallel versions of the algorithms explains the strategies adopted, so that 5xDE1 will be a five islands run with DE1 running on each island, while a 5xDE1,...,DE5 will be a five islands run with a different DE strategy running on each island, and so on. The migration happens on a ring network topology at the end of each sub-population evolution (which lasts 500 generations). Note that the number of function evaluations reported are stated per-processor.

In Table I, we report the result of the first experimental setup on the 100 dimensional Rosenbrock problem. The problem seems to be quite simple for Differential Evolution in general, as all sequential versions of the algorithm locate the global optima within a reasonable number of function evaluations. Yet, DE1 and DE3 are sometimes too greedy and get trapped in local minima. In successful runs these are also the fastest algorithms among the ones chosen. As soon as we run the parallel algorithms, we get that already with two islands, reliability and convergence speed are increased. Note that all the tried combinations of algorithms running in

[2]During the Ariadna project performed by the University of Reading and the Advanced Concepts Team (see Myatt et al. [13]), later revised and published by Izzo et al. [14].

TABLE I

ROSENBROCK 100

Sequential

$\mathcal{A}$	Trials	Success	Evaluations
DE1	20	15	1335000
DE2	20	20	1832000
DE3	20	17	1491176
DE4	20	20	1760000
DE5	20	20	2540000

Parallel

$\mathcal{A}$	Trials	Success	Evaluations (per CPU)
5xDE1	20	20	1173500
5xDE2	20	20	1507400
5xDE3	20	20	1238500
5xDE4	20	20	1325200
5xDE5	20	20	1958600
5xDE1,...,DE5	20	20	1245000

TABLE II

SCHWEFEL 300

Sequential

$\mathcal{A}$	Trials	Success	Evaluations
DE1	20	0	xx
DE2	20	13	350000
DE3	20	0	xx
DE4	20	0	xx
DE5	20	20	627000

Parallel

$\mathcal{A}$	Trials	Success	Evaluations (per CPU)
5xDE1	20	20	320000
5xDE2	20	20	387500
5xDE3	20	20	880000
5xDE4	20	20	668000
5xDE5	20	20	765400
5xDE1,...,DE5	20	20	315000

TABLE III

LENNARD-JONES 10 ATOMS

Sequential

$\mathcal{A}$	Trials	Success	Evaluations
DE1	10	0	xx
DE2	10	0	xx
DE3	10	0	xx
DE4	10	0	xx
DE5	10	0	xx

Parallel

$\mathcal{A}$	Trials	Success	Evaluations (per CPU)
5xDE1	20	8	182000
5xDE2	20	12	302500
5xDE3	20	12	332000
5xDE4	20	9	460000
5xDE5	20	9	585500
5xDE1,...,DE5	20	12	250000

the islands improve the result of the sequential algorithms. We stress that the number of function evaluations reported are referred to each island (thus each CPU) so that the price to pay, for example, to obtain the performances indicated for the 5xDE1,...,DE5 algorithm is, on average, two times the function evaluations reported. Having two separate sequential DE1 running on each of the two processors would require, on average, more function evaluations and be less reliable. Embracing this logic we conclude, for this problem, that the parallel algorithms are better than the sequential ones. The same conclusion can be drawn from the data reported in Table II relative to the 300 dimensional Schwefel problem. This problem seems to be more difficult for the different differential evolution algorithms, as only DE2 and DE5 were able to locate the global optima in at least one of twenty runs. Interestingly, the use of multiple islands changes radically the situation as the algorithms are able to consistently find the global optima using less function evaluations (per island). Even more convincing are the results reported in Table III for the Lennard-Jones problem with 10 atoms. In their sequential implementations, no version of the differential evolution algorithm was able to find the putative global minima, while in the island model parallel runs, all the different strategies adopted (5xDE1, ..., 5xDE1,...,DE5) were able to solve the problem in at least one of the runs.

The second set of experiments deals with complex problems where the global optima is unknown. We cannot use here the same experimental setup as before, since no definition of a successful run is possible. We allow a maximum number of function evaluations (summing over all islands) and we check, if the highest standard deviation among all islands populations $\mathcal{P}_i$ is larger than $\epsilon = 1e-05$. This last condition is usually not met except for greedy algorithms, in which case the computational speed gained employing such a stopping criteria can be substantial. At the end of each test we record the mean, the standard deviation, the maximum and minimum of the objective function values obtained along the different runs.

The results, shown in Table IV, V, VI, show a clear trend of the island-model to significantly improve the algorithm performance. Better averages (and better solutions) are reached using less objective function evaluations (as opposed to the first experiment here the total number of function evaluations is counted summing over all CPUs). More specifically, we compared the means and standard deviations of the parallel implementations of each of the five DE algorithms with their sequential counterparts, by using Welch's unpaired t-test. We have found that all parallel versions of the algorithms outperform the sequential ones with a confidence level of 99%, with the exception of 5xDE1 in the case of Messenger (p-value=91.98) and 5xDE2 in the case of GTOC1 (p-value=17.19).

In order to gain insight in the performance of the co-operative algorithms in the three trajectory optimisation problems, we ranked the performance of all the paral-

TABLE IV
MESSENGER

Sequential

$\mathcal{A}$	Trials	Mean	Std	Min	Max
DE1	20	11.497	1.001	10.419	13.781
DE2	20	12.325	1.130	10.229	14.319
DE3	20	12.002	1.102	10.090	13.996
DE4	20	13.155	0.789	10.943	14.054
DE5	20	13.514	0.749	12.162	14.906

Parallel

$\mathcal{A}$	Trials	Mean	Std	Min	Max
5xDE1	20	10.901	1.094	8.653	13.165
5xDE2	20	10.601	1.319	8.675	13.006
5xDE3	20	10.757	0.883	9.067	12.311
5xDE4	20	10.744	0.975	8.730	12.567
5xDE5	20	11.570	1.026	9.810	13.591
5xDE1,...,DE5	40	11.070	1.020	8.654	13.095

TABLE V
CASSINI2

Sequential

$\mathcal{A}$	Trials	Mean	Std	Min	Max
DE1	20	18.723	1.598	15.718	22.050
DE2	20	21.946	1.291	18.537	23.763
DE3	20	20.993	1.985	16.538	24.479
DE4	20	20.566	2.286	16.405	23.968
DE5	20	20.320	2.270	15.846	24.329

Parallel

$\mathcal{A}$	Trials	Mean	Std	Min	Max
5xDE1	20	16.398	2.127	12.819	20.235
5xDE2	20	16.797	1.549	14.312	18.916
5xDE3	20	16.252	2.019	12.972	19.726
5xDE4	20	16.322	2.262	12.055	20.150
5xDE5	20	18.073	1.443	15.292	21.159
5xDE1,...,DE5	20	15.950	2.386	10.256	19.604

TABLE VI
GTOC1

Sequential

$\mathcal{A}$	Trials	Mean	Std	Min	Max
DE1	20	1,058,460	236,139	603,276	1,469,395
DE2	20	868,849	168,719	584,613	1,216,370
DE3	20	876,979	90,838	728,125	1,123,748
DE4	20	962,246	107,002	740,653	1,159,833
DE5	20	940,618	114,446	648,416	1,130,297

Parallel

$\mathcal{A}$	Trials	Mean	Std	Min	Max
5xDE1	20	900,156	169,106	516,824	1,179,027
5xDE2	20	859,030	108,992	631,650	1,025,194
5xDE3	20	745,970	111,439	559,483	918,494
5xDE4	20	800,954	138,776	439,895	1,029,943
5xDE5	20	737,464	106,663	491,843	863,669
5xDE1,...,DE5	20	769,490	122,712	586,141	1,131,791

lel implementations—those running the same algorithm in each island (5xDE1, 5xDE2, 5xDE3, 5xDE4, 5xDE5), and 5xDE1,...,DE5. In particular, we compared their means and standard deviations (see Tables IV, V, VI) for each problem, using Welch's unpaired t-test. For Messenger and Cassini2, we found that 5xDE5 is performing significantly worse than all the other algorithms, with a confidence interval of 99%. The performance of all the other algorithms is not significantly different. For GTOC1, we found that 5xDE1 and 5xDE2 perform significantly worse than all the other algorithms, with a confidence interval of 99%,[3] while the performance of the rest does not present statistically significant differences. These results show that the cooperative algorithms always place themselves in the top of the ranking, outperforming "bad" algorithms and obtaining performances that are comparable to the "best" ones. This observation suggests that a cooperative algorithm might be able to tune itself towards the best performing individual algorithms, avoiding the influence of the worse ones. This may prove to be a very important step towards the automatization of the selection of good algorithms for a certain optimisation problem.

VII. Conclusions

In this article, we have presented an environment for distributing heterogeneous meta-heuristic global optimisation algorithms. More specifically, we have implemented an island-model coarse-grained parallelisation strategy. Our results confirm that such a model improves the performance of a sequential implementation of the DE optimisation algorithm, reporting on results obtained when testing on high dimensional benchmark test problems, but also on complex trajectory optimisation problems. In the case of the former problems, the improvement is observed with respect to both reliability and speed of the algorithm. In the case of the trajectory optimisation problems, the parallel algorithms are able to consistently find significantly better solutions than their sequential counterparts. Moreover, our results show that our framework for distributing heterogeneous algorithms is a promising approach to global optimisation problems, since the cooperative algorithms seem to be able to combine desirable properties derived from the best performing contributing algorithms. This approach can be considered the first step towards automatizing the algorithm selection in global optimisation tasks. Future work will focus on a more in-depth analysis of the role of the implementation of the migration operator on the final result of the optimisation process, in particular with respect to the migration rate, the network topology and the strategy of the migration update. Finally, we will thoroughly investigate the integration of different optimisation algorithms into a common optimisation framework, a non-trivial endeavor due to the intrinsically different operational principles of different algorithms, but also due to their varying search and convergence properties.

[3]The only exception is that DE2 is not significantly worse than DE4 (p-value=0.1493).

References

[1] W. N. Martin, J. Lienig, and J. P. Cohoon, *Island (migration) models: evolutionary algorithms based on punctuated equilibria*. Institute of Physics Publishing, Bristol, UK, 1997, ch. C6.3:1-C6.3:16.

[2] E. Cantú-Paz and D. E. Goldberg, "Efficient parallel genetic algorithms: Theory and practice," in *Computer Methods in Applied Mechanics and Engineering*. Elsevier, 2000, pp. 221–238.

[3] C. Gagne, M. Parizeau, and M. Dubreuil, "Distributed beagle: An environment for parallel and distributed evolutionary computations," in *Proc. of the 17th Annual International Symposium on High Performance Computing Systems and Applications (HPCS) 2003*, May 11-14 2003, pp. 201–208.

[4] N. Eldredge and S. Gould, "Punctuated equilibrium prevails," *Nature*, vol. 332, no. 6161, pp. 211–212, 1988.

[5] D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis, "Parallel differential evolution," in *IEEE Congress on Evolutionary Computation (CEC)*, 2004.

[6] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *J. of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[7] J. Apolloni, G. Leguizamòn, J. Garcìa-Nieto, and E. Alba, "Island based distributed differential evolution: An experimental study on hybrid testbeds," in *Hybrid Intelligent Systems, International Conference on*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 696–701.

[8] F. F. de Vega, G. G. Gil, and J. A. G. Pulido, "Comparing synchronous and asynchronous parallel and distributed genetic programming models," in *EuroGP '02: Proceedings of the 5th European Conference on Genetic Programming*. Springer-Verlag, 2002, pp. 326–336.

[9] D. R. Butenhof, *Programming with POSIX threads*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[10] J. Lennard-Jones, "Cohesion," *Proceedings of the Physical Society*, vol. 43, pp. 461–482, 1931.

[11] D. Wales and J. Doye, "Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms," *Journal of Physical Chemistry A*, vol. 101, no. 28, pp. 5111–5116, 1997.

[12] T. Vinkó, D. Izzo, and C. Bombardelli, "Benchmarking different global optimisation techniques for preliminary spacecraft trajectory design," in *Proceedings of the 58th International Astronautical Congress, Hyderabad, India*, 2007, paper IAC-07-A1.3.01.

[13] D. R. Myatt, V. M. Becerra, S. J. Nasuto, and J. M. Bishop, "Advanced global optimisation tools for mission analysis and design," European Space Agency, the Advanced Concepts Team, Ariadna Final Report 03-4101a, 2004, available on line at www.esa.int/act.

[14] D. Izzo, V. Becerra, D. Myatt, S. Nasuto, and J. Bishop, "Search space pruning and global optimisation of multiple gravity assist spacecraft trajectories," *Journal of Global Optimisation*, vol. 38, pp. 283–296, 2007.

[15] D. Izzo, "1st ACT global trajectory optimisation competition: Problem description and summary of the results," *Acta Astronautica*, vol. 61, pp. 731–734, 2007.

[16] W. Kwedlo and K. Bandurski, "A parallel differential evolution algorithm for neural network training," in *Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)*, 2006, pp. 319–324.