The Asynchronous Island Model and NSGA-II: Study of a New Migration Operator and its Performance

Marcus Märtens
Advanced Concepts Team
European Space Agency (ESTEC)
Noordwijk, The Netherlands
marcus.maertens@esa.int

Dario Izzo
Advanced Concepts Team
European Space Agency (ESTEC)
Noordwijk, The Netherlands
dario.izzo@esa.int

ABSTRACT

This work presents an implementation of the asynchronous island model suitable for multi-objective evolutionary optimization on heterogeneous and large-scale computing platforms. The migration of individuals is regulated by the crowding comparison operator applied to the originating population during selection and to the receiving population augmented by all migrants during replacement. Experiments using this method combined with NSGA-II show its scalability up to 128 islands and its robustness. Furthermore, the proposed parallelization technique consistently outperforms a multi-start and a random migration approach in terms of convergence speed, while maintaining a comparable population diversity. Applied to a real-world problem of interplanetary trajectory design, we find solutions dominating an actual NASA/ESA mission proposal for a tour from Earth to Jupiter, in a fraction of the computational time that would be needed on a single CPU.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; G.1.6 [**Optimization**]: Global Optimization

General Terms

Algorithms

Keywords

multi-objective optimization; island model; migration; distributed algorithms; evolutionary algorithms

1. INTRODUCTION

The need for algorithms exploiting parallel architectures has become more important than ever before. Today, supercomputers are not the only platforms having multiple CPUs: every modern PC is a parallel machine to be exploited. Connecting these machines via the internet creates another gigantic parallel computing platform whose potentials are still largely unexplored.

Due to their inherent suitability to such multiple CPU architectures, evolutionary optimization algorithms have been extensively studied in various parallel frameworks [1]. The main ideas to parallelize an optimization task are a) the master-slave model, b) the island model and c) the diffusion model. Past studies, mainly focused on homogeneous CPUs and static networks, proved clear benefits of all these schemes. However, in large, dynamic and heterogeneous CPU networks a coarse-grained approach like the island model is arguably most promising. For single-objective problems, a generalized version of the asynchronous island model has been recently studied in connection to a large number of islands (up to 1024), proving the importance of migration in general [12] and its topology (migration paths) in particular [16].

For multi-objective evolutionary algorithms (MOEA) the implementation on modern computing platforms is, instead, rather unexplored as noted also in [17]. While most works focus only on small numbers of homogeneous CPUs, a few groups [14, 15, 19] looked into grid computing architectures, heavily multithreaded environments or GPGPU computing. Some works study simple multi-start strategies as they are inherently scalable, asynchronous, robust to churn, well suited for heterogeneous environments and easy to implement, making them also the first choice for most practitioners. Besides, it can always be argued how having multiple runs with different initial seeds of an evolutionary algorithm may be more valuable than one single longer run. However, such an argument is already flawed in the case of single objective continuous optimization as shown in [12].

This work presents an implementation of the asynchronous island model suitable for multi-objective evolutionary optimization on heterogeneous and large-scale computing platforms. The main idea is that by using the crowding comparison operator [8] during migration we select solutions from a non crowded area of the Pareto-front of one population and inject them on a different population only when this improves its diversity. The importance of selecting suitable migration policies was already discussed in general by Coello et al. [5] whereas our goal is to give a analysis of the crowding comparison operator in particular. Unlike past implementations of the island model for multi-objective optimization [20, 10, 17], we do not assign a particular part of the search space nor of the Pareto-front to each island.

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

All islands are identical as they search the same space and focus on the entire the Pareto-front. This allows computational resources to be added and removed easily (during the optimization) and thus to be less sensitive to churn and heterogeneity. At Second, we make another contribution by finding the Pareto-front for the EJSM interplanetary trajectory (the NASA/ESA proposal for an Europa Jupiter System Mission), proving our approach suitable for real world applications.

2. ASYNCHRONOUS ISLAND MODEL

The island model is a coarse-grained model for parallel optimization, where N subpopulations $P_1, \ldots, P_N$ are distributed onto N islands $I_1, \ldots, I_N$ and evolve using algorithm A. An island is an abstraction of any basic parallel entity, i.e. a CPU-core, a thread or a computer in a large network. Islands evolve mostly independent, but are allowed to exchange individuals after a certain number of iterations, a process called *migration*. Migration is only allowed on migration paths between certain islands which are defined by a graph commonly called *migration topology*. By restricting these paths, each island maintains a niche where evolution can proceed (exploitation) but might also get new genetic material, which helps to maintain diversity and prevent premature convergence (exploration).

Migration has been shown to be helpful in improving and speeding up optimization in a more general context than evolutionary algorithms, i.e. when connected to different techniques such as simulated annealing, artificial bee colony, improved harmony search and many more meta-heuristics [12, 16]. The set of all islands together with the migration topology is what we call *archipelago*. Every individual in an archipelago corresponds to a solution to the optimization problem (decision vector) forming the *global Pareto-front* in contrast to the *local Pareto-front* consisting just of the solutions on a single island.

2.1 Migration

Migration is determined per island by several parameters: the migration frequency m_f is the number of iterations of the algorithm between two migrations and the migration rate m_r determines how many members of the subpopulations are selected and exchanged. Selection and replacement of migrants are governed by a selection policy m_{sel} and a replacement policy m_{rep}. Finally, a topology $\mathcal{T}$ defines the migration paths on which islands are allowed to exchange individuals.

Migration is often implemented as a synchronization point so that an island has to wait until all other islands have finished their iterations to migrate and proceed. Instead of this, we use an asynchronous implementation of the island model that is better suited for large-scale and heterogeneous computing architectures: each island I_i stores a list L_i containing the migration candidates. Before starting iterations of algorithm A, the replacement policy m_{rep} modifies the current population by taking into account all individuals in the list L_j of a randomly selected island I_j connected directly to I_i in the topology $\mathcal{T}$. According to the underlying computing architecture, shared memory (protected by the necessary mutexes), MPI or XMPP can be used by island I_i to access the information contained in L_j. Then, the algorithm A is executed for m_f iterations on the modified population. After that, the selection policy m_{sel} selects individuals from the current population, empties L_i and copies them into the list. This process repeats until a termination criterion is satisfied. Algorithm 1 summarizes the whole process for a single island.

Algorithm 1 Migration and evolution on an Island I_i

```
Initialize initial population P_i
Initialize L_i to an empty list
repeat
    Select a random island I_j directly connected by T
    P_i = m_rep(P_i ∪ L_j)
    for k = 1 → m_f do
        P_i = A(P_i)
    end for
    L_i = m_sel(P_i)
until termination criterion is satisfied
```

2.2 The island model for MOEAs

The asynchronous island model described in the previous subsection can be applied to parallelize any evolutionary algorithm as a generic scheme. According to the type of optimization problem to be solved (i.e. single/multiple objective continuous/integer, constrained/unconstrained) different choices for the selection and the replacement policy influence the archipelagos performance significantly. A common approach for MOEAs is to select and replace individuals randomly, which has the advantage of a straightforward implementation (see [20, 10]).

Instead of this, we suggest the use of a partial order $\prec$ to rank and select individuals both in m_{rep} and m_{sel}. The selection policy m_{sel} of an island I_i orders all p individuals in its current population with respect to $\prec$ and puts the first m_r into L_i. The replacement policy m_{rep} works on the union of the current population with the individuals in the selected list L_j and sorts it with respect to $\prec$. Afterwards, the first p individuals become the new population whereas the last m_r individuals are discarded. For $\prec$ we use the *crowding comparison operator* described and used in the popular non-dominated sorting genetic algorithm NSGA-II [8]. Let i_{rk} and i_{cd} be the non-domination rank and the crowding distance of individual i. Then $\prec$ is defined as:

$$i \prec j \text{ if } (i_{rk} \leq j_{rk}) \vee (i_{rk} = j_{rk} \wedge i_{cd} \geq j_{cd})$$

This partial order favours at first members with high non-domination ranks and, if individuals belong to the same front, members with the highest crowding distances. When used in the replacement and selection policies detailed above, it has the advantage that it incorporates individuals which help to improve the spreading of the Pareto-front: because a migrant that would be inserted in an already densely populated area would have a very small crowding distance, it is thus much more likely that by using $\prec$ it will be discarded in comparison to a replacement policy that discards just uniformly at random. Vice versa, the acceptance of a migrant in a sparsely populated area becomes more likely.

3. EXPERIMENTS

In this section we describe the experimental setups used to evaluate the performance of our new migration strategy in the asynchronous island model. As noted before, our scheme

problem	dimension	objectives	problem properties
ZDT1	30	2	convex
ZDT2	30	2	concave
ZDT3	30	2	disc. fronts, multimodal
ZDT4	10	2	convex, multimodal
ZDT6	10	2	concave, multimodal, biased
DTLZ1	30	3	linear, multimodal
DTLZ2	90	3	concave
DTLZ3	30	3	concave, multimodal
DTLZ4	90	3	concave, biased
DTLZ7	90	3	disc. fronts, multimodal

Table 1: Properties and used parameters of the benchmark problems.

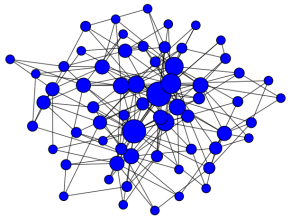

Figure 1: Example of a Barabasi-Albert topology with 64 nodes, generated with $m_0 = 3$ and 3 new random connections per added node. The size of a node is proportional to its degree.

can be used in connection to any multi-objective optimization algorithm, but here we only establish its performances when all islands are running an instance of NSGA-II. In this respect our result can also be seen as a study on coarse-grained parallelization of NSGA-II, implementing a research idea similar to one proposed by Coello et al. [5]. All experiments were conducted on a platform having sixteen Intel(R) Xeon(R) CPU - X5560 @ 2.80GHz with 8192 KB of cache. As we will simulate archipelagos with up to 128 islands, there are unpredictable effects on the wall-clock time due to cache efficiency considerations, especially when running more islands than cores (i.e. 32, 64, 128). In all comparative results, these effects are all factored out. The asynchronous island model, the new migration operator and the NSGA-II implementation used are all open source contributions to the parallel optimization platform PyGMO[1] under development at the European Space Agency.

3.1 Benchmark Problems

We use selected problems out of two well-known benchmark suites, namely the ZDT [21] and the DTLZ [9] problems. These problems cover various features like concave geometries, disconnected Pareto-fronts, biased and multimodal problems (see Table 1). The problem dimensions were chosen as to create an interesting complexity level.

3.2 Performance measures

Each of our test-problems is constructed as a composition of functions that describe the position and the shape of the Pareto-front, which is thus known beforehand. One of these functions, which we call *distance function* and indicate with g, is minimized by any Pareto-optimal solution. This fact can be used to define a convergence metric as follows: let g be the distance function of a problem, P a set of solutions and x^* any Pareto-optimal decision vector, we define

$$\Upsilon(P) = \frac{1}{|P|} \sum_{x \in P} g(x) - g(x^*) \qquad (1)$$

as *convergence metric* of the corresponding problem. The lower the $\Upsilon(P)$, the closer the solutions of P are to solutions of the Pareto-front. Although there are already well-established convergence metrics in literature, Υ has the advantage of not relying on a reference set of Pareto-optimal solutions which allows for a more efficient computation while reflecting the same behaviour as most of the other convergence metrics.

For the ZDT problems we also use the diversity metric Δ introduced by Deb et. al [8, 6] and defined as:

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{p-1} \left| d_i - \bar{d} \right|}{d_f + d_l + (p-1)\bar{d}}$$

where p is the population size, d_i is the Euclidean distance between two consecutive solutions $i-1, i$ in the fitness space, $\bar{d}$ the average distance over all these distances, and d_l, d_f are the distances from the border solutions to the known extreme solutions of the Pareto-front.

3.3 The archipelago setups

To illustrate the effect of migration in experiments, different underlying migration topologies $\mathcal{T}$ can be considered. The extreme cases are the unconnected graph which does not include any migration paths at all. We call the setup based on the unconnected graph the *unconnected setup*. It corresponds to a multi-start strategy where an algorithm is executed several times on different initial populations and using different random seeds. In contrast to this, one can consider the fully-connected graph as a topology, which allows migration between arbitrary islands directly. While preliminary experiments with NSGA-II showed that the best results were achieved by using this topology, there are some heavy drawbacks. By using a fully-connected topology, each new island has to be linked to any existing island, producing a large communication overhead and a rather inefficient scalability. To be more applicable in practice, we decided to base our migration experiments on another graph instead, which is called the Barabasi-Albert network [2]. This network is a scale-free network observed in many growing systems including the internet, citation networks and social networks. Its degree of separation is in general small enough to allow a quick migration between any two islands but keeps the communication overhead much lower than a fully-connected topology. Moreover, this topology is known for its robustness against random node failures, characteristics of particular interest for heterogeneous large CPU networks. We grow our Barabasi-Albert networks by starting with $m_0 = 3$ nodes and adding 3 new random connections for each additional node. Figure 1 depicts an example of an archipelago with 64 islands. Based on the Barabasi-Albert network topology, we examine two other setups: in the *random migration*

[1] http://pagmo.sourceforge.net/pygmo/index.html

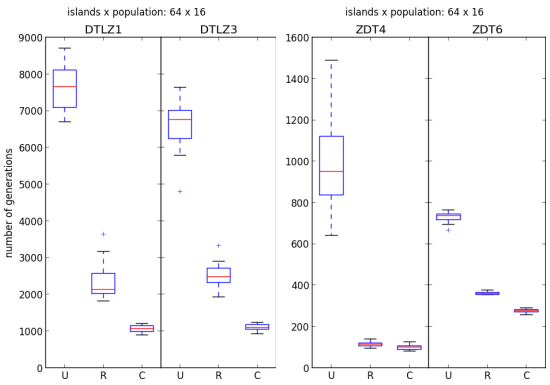

Figure 2: Number of generations till one island reaches τ for unconnected setup U, random migration setup R and crowding migration setup C.

setup selection and replacement of individuals happens uniformly at random and in the *crowding migration setup* we use our suggested scheme $\prec$ based on the crowding distance operator.

For each setup, the initial population is initialized at random, but is kept the same for all three setups. We always apply NSGA-II with a cross-over probability of $p_c = 0.9$, a mutation probability of $p_m = 0.01$ and distribution indices $\eta_c = 10$ and $\eta_m = 10$. The migration frequency is set to $m_f = 5$ generations and the migration rate to $m_r = 0.2 \cdot p$, rounded accordingly. No significant qualitative difference on the reported results was, though, observed when experimenting with different values.

The termination criterion is based on Υ and a problem-dependent threshold τ. We stop all islands in a setup as soon as there is one island with population P for which $\Upsilon(P) < \tau$ holds, where the τ is 0.01 for the ZDT and 0.1 for the DTLZ problems, taking into account their higher number of objectives. Reaching the threshold means that all individuals of a population in one island have converged close enough to a Pareto-optimal solution of the Pareto-front. We say that the population has *converged* to the Pareto-front in this case. While being important, note that Υ does not make any statement about whether the converged population is well-spread over the whole front or not.

3.4 Results

We conducted our experiments for all three setups, varying the number of islands $N \in \{16, 32, 64, 128\}$ and the number of individuals per island $p \in \{8, 16, 32, 64, 128\}$, using 25 repetitions for every combination of p and N as to accumulate enough statistical significance. During our experiments we recorded, for each of the 25 runs of every setup, wall-clock times (per island) and the generations necessary to converge. For the ZDT problems we also computed Δ. We report a summary of the whole experimental campaign, highlighting the most interesting results and observations. Statistical significance for all experiments was consistently found as standard deviations are all very small, which is why we omit an exhausting statistical analysis at this point.

First, we looked at the comparison between the unconnected setup and the migration setups. On every combination of p and N the migration setups clearly outperform the

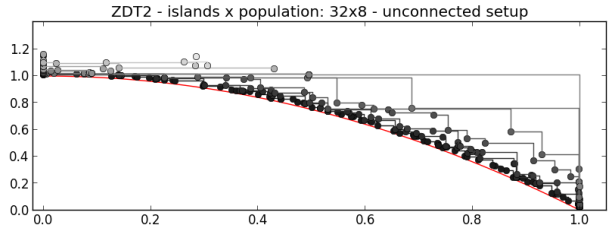

Figure 3: Sample of the solutions of an archipelago in an unconnected setup when $\tau = 0.01$ was reached by one island. Convergence after 170 generations, $\Delta = 0.680254$. Clearly, some islands need additional generations to converge.

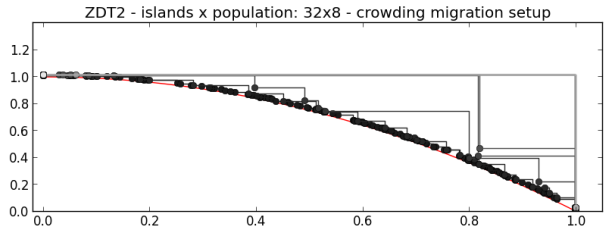

Figure 4: Sample of the solutions of an archipelago in a crowding migration setup when $\tau = 0.01$ was reached by one island. Convergence after 175 generations, $\Delta = 0.638917$. All islands approach the Pareto-front with a similar convergence speed.

unconnected setup both in terms of generations and of wall-clock time. This effect was strongest for multimodal problems like ZDT4, ZDT6, DTLZ1 or DTLZ3 and for small population sizes p as shown in Table 2. Multi-start approaches for NSGA-II can thus be improved by exchanging information among independent runs. The question of whether to migrate or not thus always has a positive answer. We extend thereby the result acquired for single-objective algorithms [12] to NSGA-II.

Afterwards, we looked at the comparison between the two migration setups as to assess the performance of our proposed migration strategy based on $\prec$. We observed that the crowding migration setup outperforms by a factor of 2-3 the random migration setup in terms of average number of generations for the harder DTLZ1 and DTLZ3 problems and by smaller amounts for the easier ZDT problems. For the latter it does not need many less generations on average to converge as shown in Figure 2 for an archipelago with $p = 16$ and $N = 64$. Random migration may even have a slightly smaller average wall-clock runtime in those cases (such as ZDT4) where almost the same number of generations is needed for convergence. This is due to the implementation overhead of the crowding migration operator compared to the more easier random migration operator.

We then looked at the diversity of the global Pareto-front (i.e. $\Delta(\bigcup_{i=1}^{N} P_i)$) when convergence was reached. No significant difference was found between all the setups. The only exception is ZDT2 for small p where migration appears to help in finding the border solutions (observed for all but $p = 128$). Figure 3 and 4 show the global Pareto-fronts for the unconnected and crowding migration setup as an example. One can clearly see how in the unconnected setup many islands proceed slower and their individ-

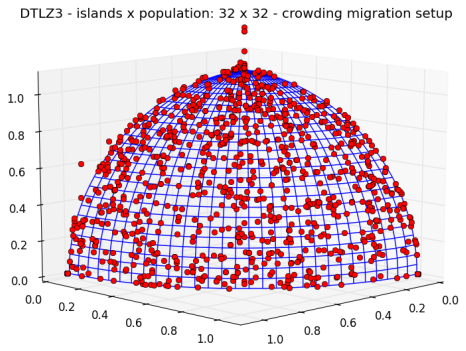

Figure 5: Sample of a global Pareto-front of a crowding migration setup till $\tau = 0.01$ was reached by one island after 970 generations.

uals have higher domination ranks on the archipelago level. Figure 5 shows the spreading of the global convergence over the Pareto-front for the DTLZ3 problem after one island reached $\tau = 0.01$. All in all, the diversity at the archipelago level is maintained when migrating, while migration helps islands in dominated areas to proceed faster towards the Pareto-front as they would by themselves.

Table 3 shows the generations and wall-clock times per island (averaged over the 25 runs) for all combinations of p and N in the crowding migration setup. By looking at the anti-diagonals we are comparing runs where the population number is equal (constant $N \cdot p$). When a fixed population size at the archipelago level is desired, the crowding migration setup encourages to have setups with a high number of islands evolving small populations. Reading the table by column we note how there is mostly just a very small speed-up in adding islands to an already existing archipelago. However, The advantage in this case is, that by increasing the product $p \cdot N$ we are getting more solutions at the archipelago level allowing for a finer approximation of the actual Pareto-front in roughly the same amount of time.

We finally observed that NSGA-II occasionally produces degenerated fronts on DTLZ4. The algorithm is known to perform in that way on this particular problem (see [9]) which will result in approximations consisting of just a single point or only a line of the border from the spherical shape of the Pareto-front. Using more than $N > 32$ islands, we observed that the island that reached τ first always had such a degenerated front in the unconnected setup but never in the crowding migration setup. We thus conclude that the crowding migration helps NSGA-II to deal better with strongly biased problems like DTLZ4.

4. TRAJECTORY OPTIMIZATION

The design of an interplanetary trajectory with deep space manoeuvres and multiple fly-bys is a challenging task. In order to maximize the payload of the on-board scientific instruments, the propellant mass has to be minimized. On the other hand, the transfer time needs to be short as to allow the mission to reach its objectives without having years of unnecessary delay.

NSGA-II has already been applied by Deb et. al [7] to a simplified version of this problem, allowing only 3 planetary

problem	number of islands N				population per island p
	16	32	64	128	
DTLZ1	6.29	8.46	9.15	11.14	8
	4.98	5.70	7.21	7.83	16
	4.01	4.74	5.37	5.74	32
	3.12	3.38	3.70	3.88	64
	2.53	2.59	2.78	2.84	128
DTLZ3	5.62	6.68	7.76	7.98	8
	4.29	5.43	6.02	6.54	16
	3.30	3.70	4.00	4.20	32
	2.57	2.69	2.88	2.98	64
	1.94	2.15	2.12	2.24	128
ZDT4	10.74	16.61	22.68	22.31	8
	7.44	8.67	9.68	10.07	16
	3.08	4.47	4.65	4.78	32
	2.50	2.55	3.20	3.19	64
	1.90	1.86	2.05	1.99	128
ZDT6	2.94	3.28	3.68	3.86	8
	2.19	2.46	2.67	2.77	16
	1.79	1.94	2.05	2.12	32
	1.51	1.60	1.69	1.78	64
	1.35	1.43	1.58	1.69	128

Table 2: Ratio of average number of generations until τ between unconnected and crowding migration setup.

parameter	unit	lower bound	upper bound
t_0	MJD2000	6210	8400
T	days	1000	3000
u	-	0	1
v	-	0	1
V_∞	km/s	1.0	3.5
$\eta^1, \ldots, \eta^\ell$	-	$1 \cdot 10^{-5}$	0.99999
$\alpha^1, \ldots, \alpha^\ell$	-	$1 \cdot 10^{-5}$	0.99999
$r_p^1, \ldots, r_p^{\ell-1}$	Planet Radii	1.1	100
$\beta^1, \ldots, \beta^{\ell-1}$	rad	-2π	2π

Table 4: Encoding of a MGA-1DSM tour with ℓ legs.

fly-bys and using a simplified dynamics not allowing for deep space manoeuvres. In this section, we use the MGA-1DSM trajectory model [11] to design a planetary tour using a predefined fly-by sequence of an arbitrary length and allowing the spacecrafts propulsion system to perform up to one DSM between each two planetary encounters. By the aid of our asynchronous island model, we construct approximating Pareto-fronts for various different sequences starting with Earth and ending with Jupiter and compare these results with the actual mission baseline of the NASA/ESA joint proposal for an Europa Jupiter System Mission (EJSM) [4].

4.1 Problem Description

Fixing a sequence S of planets for fly-by manoeuvres, we divide our trajectory into ℓ legs, where each leg corresponds to a transfer between two successive planets in S. We use the encoding and bounds described in Table 4 to model our optimization problem allowing for up to one DSM on each leg as follows.

The parameter t_0 describes the date of launch, u, v the launch direction and V_∞ is the departure velocity of the spacecraft. The parameter η^i describes the fraction of the

problem	number of islands N				population per island p
	16	32	64	128	
ZDT1	222.0 / 0.15	190.6 / 0.12	168.8 / 0.10	150.0 / 0.08	8
	186.6 / 0.21	167.0 / 0.17	151.0 / 0.13	138.4 / 0.10	16
	154.4 / 0.34	144.0 / 0.26	133.8 / 0.18	125.8 / 0.14	32
	125.0 / 0.51	121.0 / 0.38	112.6 / 0.30	104.4 / 0.25	64
	100.4 / 0.81	98.0 / 0.65	88.4 / 0.52	80.4 / 0.45	128
ZDT2	208.2 / 0.15	178.0 / 0.12	156.6 / 0.10	138.6 / 0.08	8
	172.4 / 0.20	153.0 / 0.16	138.0 / 0.12	126.2 / 0.10	16
	136.2 / 0.31	128.0 / 0.24	118.4 / 0.17	109.4 / 0.13	32
	110.8 / 0.47	103.8 / 0.33	96.0 / 0.26	89.0 / 0.22	64
	88.6 / 0.71	86.6 / 0.59	76.2 / 0.48	66.6 / 0.40	128
ZDT3	229.8 / 0.16	199.4 / 0.13	179.0 / 0.11	159.4 / 0.09	8
	189.8 / 0.23	174.6 / 0.18	158.2 / 0.15	145.8 / 0.11	16
	154.8 / 0.35	144.6 / 0.27	133.4 / 0.19	124.8 / 0.14	32
	133.0 / 0.55	129.0 / 0.41	118.6 / 0.32	108.8 / 0.26	64
	115.4 / 0.90	114.0 / 0.76	103.6 / 0.62	93.2 / 0.53	128
ZDT4	591.2 / 0.36	423.0 / 0.25	198.6 / 0.11	137.8 / 0.07	8
	292.2 / 0.30	183.4 / 0.16	154.4 / 0.13	95.8 / 0.07	16
	169.2 / 0.35	111.4 / 0.19	91.6 / 0.12	76.8 / 0.09	32
	117.6 / 0.49	82.2 / 0.26	73.2 / 0.18	65.0 / 0.16	64
	76.2 / 0.65	69.8 / 0.51	59.4 / 0.40	53.8 / 0.34	128
ZDT6	426.0 / 0.24	371.0 / 0.21	322.8 / 0.17	290.0 / 0.15	8
	363.4 / 0.34	325.8 / 0.27	293.0 / 0.23	270.6 / 0.18	16
	309.2 / 0.57	286.2 / 0.44	263.0 / 0.31	248.0 / 0.25	32
	250.8 / 0.98	236.6 / 0.68	221.6 / 0.50	204.0 / 0.42	64
	203.2 / 1.57	195.4 / 1.23	173.2 / 0.98	157.8 / 0.85	128
DTLZ1	2632.4 / 2.61	1844.0 / 1.60	1383.0 / 1.10	1140.4 / 0.78	8
	1878.2 / 3.33	1321.0 / 1.93	1091.6 / 1.27	889.8 / 0.88	16
	1312.2 / 4.32	1079.6 / 2.53	896.6 / 1.63	794.2 / 1.37	32
	1076.4 / 5.49	898.8 / 3.66	775.8 / 2.86	666.0 / 2.38	64
	757.8 / 9.09	686.0 / 6.71	624.4 / 5.58	596.8 / 4.61	128
DTLZ2	543.4 / 0.71	464.2 / 0.52	416.2 / 0.42	386.4 / 0.31	8
	368.4 / 0.83	335.2 / 0.60	311.6 / 0.43	294.0 / 0.34	16
	264.8 / 1.02	250.2 / 0.70	236.4 / 0.53	219.2 / 0.43	32
	200.8 / 1.19	194.2 / 0.95	182.6 / 0.78	168.8 / 0.69	64
	157.8 / 1.85	154.8 / 1.55	145.2 / 1.36	133.4 / 1.18	128
DTLZ3	2883.2 / 3.02	2045.4 / 1.86	1637.0 / 1.37	1311.2 / 0.93	8
	1780.8 / 3.26	1383.2 / 2.07	1144.2 / 1.38	931.2 / 0.94	16
	1091.8 / 3.77	970.8 / 2.36	853.2 / 1.59	745.6 / 1.33	32
	809.4 / 4.28	748.2 / 3.20	656.2 / 2.52	602.2 / 2.12	64
	589.6 / 6.14	582.2 / 5.49	516.6 / 4.35	480.6 / 3.81	128
DTLZ4	363.8 / 0.49	320.2 / 0.37	278.4 / 0.29	255.4 / 0.21	8
	293.2 / 0.69	269.8 / 0.50	245.6 / 0.34	222.2 / 0.26	16
	226.6 / 0.90	210.8 / 0.59	194.4 / 0.44	183.0 / 0.37	32
	176.8 / 1.07	170.2 / 0.86	158.4 / 0.69	144.2 / 0.60	64
	139.6 / 1.72	136.6 / 1.40	128.2 / 1.22	116.0 / 1.04	128
DTLZ7	495.6 / 0.62	420.6 / 0.46	358.0 / 0.35	321.2 / 0.25	8
	373.2 / 0.82	327.6 / 0.57	301.0 / 0.41	279.8 / 0.32	16
	300.0 / 1.12	276.4 / 0.76	255.4 / 0.56	234.8 / 0.45	32
	238.6 / 1.38	229.4 / 1.11	214.0 / 0.89	196.8 / 0.79	64
	201.0 / 2.33	195.6 / 1.94	182.6 / 1.67	168.4 / 1.45	128

Table 3: Average number of generations until τ for the crowding migration setup / average wall-clock time per island until τ for the crowding migration setup.

i-th leg on which the DSM is taken, whereas β^i, r_p^i model the fly-by radius and the orientation of the fly-by plane during the i-th fly-by. Thus we have in total $3 + 4 \cdot \ell$ decision variables by using this encoding. The upper bound on V_∞ is set to the maximum velocity allowed by the EJSM proposal. Also t_0 and T are bounded in a way that the EJSM solution with $t_0^* = 7364$ and $T^* = 2122$ is in the decision space.

Our first objective is to minimize the cumulative velocity change Δv, which is linked to the fraction of mass used for propellant via the Tsiolkovsky rocket equation [18]. The second objective is the total transfer time T, which we directly encode as a decision variable into the chromosome. The transfer times $T_1, \ldots, T_\ell$ per leg can be computed by the parameters $\alpha^1, \ldots, \alpha^\ell$ using the following equation:

$$T_i = \frac{\alpha^i \cdot T}{\sum_{j=1}^{\ell} \alpha^j} \qquad i = 1, \ldots, \ell.$$

4.2 Optimization Technique and Results

We observed that by using NSGA-II on any fixed planetary sequence with a random initialization, we only achieve poor results that miss large parts of the fitness space, especially in the area of low Δv that is of particular interest in the domain. To overcome this limitation, we initialized the population with solutions obtained by running a single-objective optimizer optimizing for low Δv only. We have chosen jDE [3] for this purpose which we also embedded into the asynchronous island model in order to improve the speed of computation and quality of solution, using $N = 8, p = 40, m_f = 100, m_r = 2$ and a ring topology, migrating the individuals with the best fitness values, replacing the ones with the worst. We also used a minor modification in the encoding of the problem. Instead of using T and $\alpha_i, \ldots, \alpha_\ell$ we put the transfer times per leg $T_1, \ldots, T_\ell$ directly into the chromosome, setting appropriate bounds. This produces better fitness values as with the original encoding, which is, however, better suited for multi-objective optimization allowing to have box bounds on the total time of flight (i.e. one of the objectives).

Evolution of jDE was stopped after the Δv did not decrease by more than 0.1 during 1000 generations and the best solution was extracted. In order to maintain a diverse set of solutions, this procedures was repeated 1024 times. Running NSGA-II for at least 10000 generations on this population produced better results than our first attempt, but took us more than $4h$ to be completed[2]. By using the island model in a crowding migration setup with $N = 16$, $p = 64$ we were able to lower this time dramatically to 140s per 10000 generations on average.

To allow for a stronger exploitation in the NSGA-II algorithm, we used $m_f = 10$ whereas all other parameters of the setup were the same as in subsection 3.1. As a final step, we put all individuals from all islands into one population and used NSGA-II for just 50 more generations on them, allowing every individual to advance to a single non-dominated front while maintaining a good spread of solutions.

[2]The original implementation of NSGA-II we used scales quadratically with the number of individuals of a population. Using a more sophisticated implementation like in [13] could improve the runtime, but would not change the quality of the overall solution. Moreover, we still need a fairly large amount of generations as NSGA-II is just making small improvements for every evolutionary step.

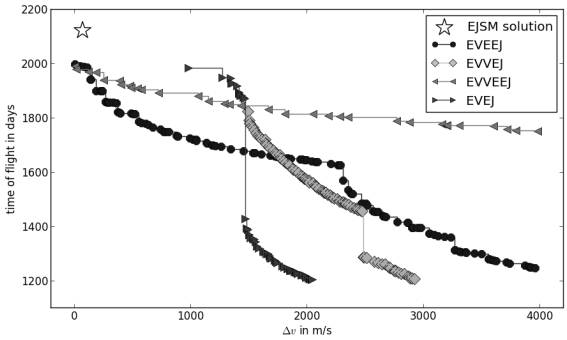

Figure 6: Pareto-fronts for different fly-by sequences (E=Earth, V=Venus, J=Jupiter) between 1200 and 2200 days of transfer using at most $\Delta v = 4000$ m/s. Solutions outside these bounds are excluded.

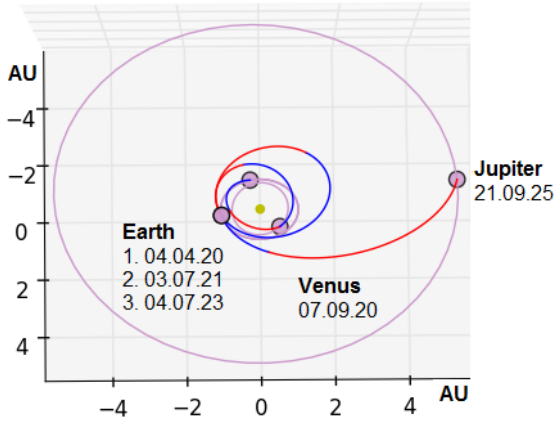

Figure 7: Alternative EVEEJ transfer found by NSGA-II with $\Delta v \approx 9.33$ m/s and a transfer time of 1996 days.

Using the outlined method, we analyzed several different sequences of planets (encoded with E for Earth, V for Venus and J for Jupiter) to obtain alternative trajectories from Earth to Jupiter[3]. Combining all trajectories we found creates a comprehensive overview on possible interplanetary transfer options in the given launch window of 6 years (01.01.2017 - 31.12.2022) and the possible trade-offs, shown in Figure 6. It turns out that the solutions of EVEEJ and the EVEJ trajectories dominate all other solutions, where at $\Delta v = 1452$ m/s a change between these two sequences can help saving over 249 days. Also, having an additional flyby on Venus is an alternative for very low Δv trajectories, as there are EVVEEJ solutions here that dominate a small part of the EVEEJ front.

Surprisingly, our optimization was able to find solutions for EVEEJ and EVVEEJ sequences that dominate the EJSM from NASA/ESA, which is also based on an EVEEJ sequence. The trajectory of the lowest Δv solution of our EVEEJ Pareto-front is shown in Figure 7 as an example. It uses a Δv of 9.33 m/s and a total transfer time of 1996 days instead of the EJSM proposal that uses a Δv of 67.95

[3]We also analyzed sequences containing Mars but omit these results here as they are far less competitive as sequences using Earth and Venus only.

m/s in 2122 days. A further investigation showed that this was coming at the price of a higher arrival velocity at the Jupiter system of 6.08 km/s instead of 5.5 km/s as in the EJSM. It would be interesting to look into the arrival velocity as a third objective to further explore the quality of the solution.

5. CONCLUSIONS

Motivated by the emergence of large heterogeneous computational networks, we have outlined how MOEAs can be embedded in the asynchronous island model while proposing a novel migration operator to select and replace solutions in sparsely populated areas. The experiments indicate that we can use a high number of islands with small populations to speed up convergence while maintaining a good diversity of the global solution. Furthermore, our approach is scalable and helps NSGA-II to not converge in degenerated fronts. As a second contribution, we have shown how our asynchronous island model together with an appropriate initialization technique can be used to produce a Pareto-front for a difficult interplanetary trajectory optimization task.

Future research might include additional MOEAs and new migration policies. Also combining different MOEAs in a single crowding migration setup could give interesting results. Using more sophisticated performance indicators might show how migration not only affects convergence speed but also the overall quality of the global approximation. Moreover, it would be interesting to see how the island model performs on more difficult problems that are non-separable or have a deceptive fitness landscape. We also think that the island model can be helpful in solving many-objective optimization problems with a high number of objectives. Algorithms for these problems usually need very large populations to construct a meaningful approximation of the Pareto-front. Since our experiments indicate that migration can help NSGA-II to converge even with small populations we can be optimistic.

6. REFERENCES

[1] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 6(5):443–462, 2002.

[2] A. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[3] J. Brest, V. Zumer, and M. Maucec. Self-adaptive differential evolution algorithm in constrained real-parameter optimization. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 215–222. IEEE, 2006.

[4] K. Clark, T. Magner, R. Pappalardo, M. Blanc, R. Greeley, J. Lebreton, C. Jones, and J. Sommerer. Jupiter europa orbiter mission study 2008: Final report. *The NASA Element of the Europa Jupiter System Mission Technical Document, Task Order# NMO710851*, 2009.

[5] C. A. C. Coello, G. B. Lamont, and D. A. V. Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007.

[6] K. Deb. Multi-objective optimization. *Multi-objective optimization using evolutionary algorithms*, pages 13–46, 2001.

[7] K. Deb, N. Padhye, G. Neema, and V. Adimurthy. Interplanetary trajectory optimization with swing-bys using evolutionary multi-objective optimization. *Lecture Notes in Computer Science*, 4683:26–35, 2007.

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[9] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable test problems for evolutionary multiobjective optimization. *Evolutionary Multiobjective Optimization*, pages 105–145, 2005.

[10] K. Deb, P. Zope, and A. Jain. Distributed computing of pareto-optimal solutions using multi-objective evolutionary algorithms. In *Proceedings of the Second Evolutionary Multi-Criterion Optimization (EMO-03) Conference, 8-11 April, Faro, Portugal*, pages 535–549. Proceedings of the Second Evolutionary Multi-Criterion Optimization (EMO-03) Conference, 8-11 April, Faro, Portugal, 2003.

[11] D. Izzo. *Global optimization and space pruning for spacecraft trajectory design*. Cambridge University Press, New York, NY, USA, 2010.

[12] D. Izzo, M. Ruciński, and F. Biscani. The generalized island model. *Parallel Architectures and Bioinspired Algorithms*, pages 151–169, 2012.

[13] M. Jensen. Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms. *Evolutionary Computation, IEEE Transactions on*, 7(5):503–515, 2003.

[14] N. Melab, M. Mezmaz, and E. Talbi. Parallel cooperative meta-heuristics on the computational grid.: A case study: the bi-objective flow-shop problem. *Parallel computing*, 32(9):643–659, 2006.

[15] S. Mostaghim, J. Branke, A. Lewis, and H. Schmeck. Parallel multi-objective optimization using master-slave model on heterogeneous resources. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1981–1987. IEEE, 2008.

[16] M. Ruciński, D. Izzo, and F. Biscani. On the impact of the migration topology on the island model. *Parallel Computing*, 36(10):555–571, 2010.

[17] E. Talbi, S. Mostaghim, T. Okabe, H. Ishibuchi, G. Rudolph, and C. Coello Coello. Parallel approaches for multiobjective optimization. *Multiobjective Optimization*, pages 349–372, 2008.

[18] D. Vallado. *Fundamentals of astrodynamics and applications*, volume 12. Springer, 2001.

[19] M. Wong. Parallel multi-objective evolutionary algorithms on graphics processing units. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2515–2522. ACM, 2009.

[20] N. Xiao and M. Armstrong. A specialized island model and its application in multiobjective optimization. In *Genetic and evolutionary computation GECCO 2003*, pages 213–213. Springer, 2003.

[21] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.