PaDe: A Parallel Algorithm Based on the MOEA/D Framework and the Island Model

Andrea Mambrini[1,⋆] and Dario Izzo[2]

[1] University of Birmingham, Birmingham, UK
[2] European Space Agency, Noordwijk, The Netherlands

Abstract. We study a coarse grained parallelization scheme (thread based) aimed at solving complex multi-objective problems by means of decomposition. Our scheme is loosely based on the MOEA/D framework. The resulting algorithm, called Parallel Decomposition (PaDe), makes use of the asynchronous generalized island model to solve the various decomposed problems. Efficient exchange of chromosomic material among islands happens via a fixed migration topology defined by the proximity of the decomposed problem weights. Each decomposed problem is solved using a generic single objective evolutionary algorithm (in this paper we experiment with self-adaptive differential evolution (jDE)). Comparing our algorithm to MOEA/D-DE we find that it is attractive in terms of performances and, most of all, in terms of computing time. Experiments with increasing numbers of threads show that PaDe scales well, being able to fully exploit the number of underlying available cores.

1 Introduction

In many real-world decision problems several conflicting criteria need to be optimized at the same time. Those problems can be modelled as Multi-objective Optimization Problems (MOPs). A MOP is defined as follow:

$$\begin{array}{ll}\text{Minimize} & F(x) = (f_1(x), \ldots, f_o(x)) \\ \text{subject to} & \qquad x \in \Omega\end{array}$$

where Ω is the *decision space*, $F(x) : \Omega \to \mathbb{R}^o$ consists in o *objective functions*, $\mathbb{R}^o$ being the *objective space*. In continuous problems $\Omega \subset \mathbb{R}^s$ and s is defined as the *problem size*.

Being $u = (u_1, \ldots, u_o)$ and $v = (v_1, \ldots, v_o)$ two objective vectors in Ω, we say that u dominates v if $u_i \leq v_i$ for $i = 1, \ldots, o$ and the strict inequality sign holds for at least one objective. A point $x^* \in \Omega$ is called *Pareto Optimal* if there isn't any $x \in \Omega$ such that $F(x)$ dominates $F(x^*)$. The set of all pareto optimal points in Ω is called *Pareto Set* and the set of the associated objectives is called *Pareto Front*. The aim of a multi-objective optimisation algorithm is to find a well spread set of points in the Pareto Set, or as close as possible to the Pareto Set.

⋆ Andrea Mambrini has been partially supported by EPSRC through grant no. EP/I010297/1.

Traditionally, multi-objective evolutionary algorithms (MOEAs) are dominance-based. In dominance-based algorithms, individuals are selected using the Pareto dominance notion and some auxiliary criteria aimed at maintaining a good distribution along the same non-dominated front. Examples of dominance-based MOEAs are NSGA-2 [1] and SPEA-2 [2]. Decomposition is another way to solve a MOP. MOEA/D [3] decomposed the original multi-objective problem into many single objective problems constructed in a way that the optimal solution to each of these subproblems is a point on the optimal Pareto Front of the original MOP. It then uses a single-objective optimization algorithm to solve concurrently the subproblems and returns as Pareto Set the union of the optimal solution to each single objective subproblem.

There are two main components of a MOEA/D. The first one is the mechanism to decompose the MOP into subproblems. Traditionally weight vectors are randomly generated and from each of them a single objective problem is obtained using weighted sum [4], Tchebycheff [4] or Boundary Intersection [5]. Some recent work propose a way to adapt the weight vectors throughout the run [6]. The second important component is the way to solve the single objective problems obtained from the decomposition. The original approach uses point crossover and standard mutation [3], while other approaches use different operators, e.g. from Differential Evolutions [7] or Particle Swarm Optimization [8]. As the decomposed problems are solved concurrently, it is natural to consider parallel implementations of a MOEA/D algorithm. In recent work [9] [10] a thread-based parallelization of MOEA/D-DE has been proposed. The approach, a fine-grained scheme, is that of parallelizing the original MOEA/D-DE algorithm by dividing the main population loop in different threads. In order for this fine-grained approach to balance the work load across threads, the number of threads must be kept small in comparison to the population size and the objective function evaluation must account for most of the computing time.

In this paper we introduce a new algorithm called Parallel Decomposition (PaDe) using a different approach to parallelize a MOEA/D algorithm. The idea is to solve each subproblem in a separate logical computational unit and to then use the island model [11] paradigm to introduce exchange of solutions across problems. In an island model, several subpopulations (islands) are evolved independently. Selected individuals are then sent to other islands during a process called *migration*. Island models are well suited for parallelization since group of islands can run in parallel on different computational units. Traditionally in an island model each island runs the same algorithm and solves the same problem. PaDe uses instead a generalised (or heterogeneous) island model where each island solves a different problem and can run a different optimisation algorithm [12]. A recent theoretical work studied how heterogeneous island models can find approximate solutions to NP-Hard problems [13]. Compared to fine-grained parallelization approaches the asynchronous island model at the core of PaDe, is suitable for modern multi-cores architectures as well as for heterogeneous parallel architecture (e.g. grid computing). Moreover the modularity of this approach allow to employ any single objective solver or even run different ones on each island, thus assigning different areas of the Pareto Front to different algorithms.

We made a C++ implementation of PaDe available as part of the open source scientific library PaGMO [14], and its python front-end PyGMO [15].

Firstly we compare PaDe (with a self-adaptive version of Differential Evolution) to the MOEA/D-DE algorithm [7]. We will show that the two approaches get similar pareto front's quality. We then investigate the parallel performances of PaDe showing how increasing the number of threads up to the population size, PaDe is able, unlike fine grained approach proposed elsewhere [9] [10], to fully take advantage of the underlying cores (linear speedup) even when the objective function has small computational cost.

2 Algorithm Definition

PaDe is a multi-objective evolutionary framework based on decomposition and parallelized using the island model. It first generates m weight vectors of dimension o (being o the number of objectives of the original multi-objective problem) using a weight vector generation method W. The o components of the weight vector must sum to 1, thus the vector must lie on the standard $(o-1)$-simplex. The weight vectors can be generated using one of the following methods:

- *GRID*: the weights are generated to optimally maximize their spread as described in [3]. Using this method it is not possible to generate any amount of weight vectors. In fact, for any fixed $H \in \mathbb{N}$ this method can generate $m = \binom{o-1}{H+o-1}$, where o is the dimension of the weight vectors.
- *RANDOM*: differently from GRID it can generate any amount of vectors. It generates each weight vector sampling uniformly at random between 0 and 1 each component. In order to enforce that the sum of all the components of a weight vector is equal to 1, each vector is projected to the o-dimensional standard simplex. This method cannot guarantee the optimality of the spread as the previous method.
- *LOW-DISCREPANCY*: a novel method to generate any amount of weight vectors with a good spread. An Halton sequence [16] of m points of size o is generated. As in the RANDOM method it is then projected to the standard $(o-1)$-simplex. This method is a good compromise between the maximum spread guaranteed by the GRID method, and the freedom to generate any amount of vectors guaranteed by RANDOM. We introduced this method as in our island model it is often useful to increase/decrease the population sizes adaptively, which would be not allowed by the standard weight generation method GRID.

After generating the weight vectors, PaDe decomposes the original multi-objective problem into m single objective problems $\mathcal{P}_i$ using one decomposition method between Weighted, Tchebycheff and Boundary Intersection, obtaining each problem from a weight vector w_i. It then assigns each subproblem to an island defined by a single-objective solver S and a population of size $T+1$. The evolution of the islands is executed by n threads, each one taking care of evolving m/n islands. In order to fully take advantage of the parallelism, n should be set at least as the number of available computational nodes, so that each core will execute at least one thread. Each island i, associated with the weight vector w_i, is connected to the T islands whose weight vectors are the closest to w_i according to the Euclidian distance: this way migration will exchange solutions between islands solving similar problems.

Then the following is repeated for G_P (PaDe's number of generations) times: each island is evolved using the single objective solver S for G_S generations (solver's number of generations) and, at the end of the evolution, the worst T individuals of each island are replaced from the best individuals from each of the T neighbouring islands (migration). Eventually the union of the best individuals from each island is returned as final population. Algorithm 1 provides an high level pseudo-code description of PaDe.

Algorithm 1. PaDe (Decomposition method D, single-objective solver S, weight generation method W, number of threads n)

```
Generate $m$ weight vectors $w_1, \dots, w_m$ using $W$
For each $w_i$ find the set $N_i$ containing the $T$ indices of the Euclidian closest weights

Generate $m$ problems $P_i$ using weights $w_i$ and decomposition method $D$
Create $m$ random populations $\mathcal{P}_i$ of dimension $T+1$
Assign each population $\mathcal{P}_i$ to a decomposed problem $P_j$
for $k = 0$ to $G_P$ do
  Set $s = 1$
  while $s < (m+n)$ do
    for $i = s$ to $\min(s+n, m)$ in parallel do
      Migrate solutions from $\mathcal{P}_j, j \in N_i$
      Evolve $\mathcal{P}_i$ using $S$ for $G_S$ generations
    $s = s + n$
```

In its current implementation PaDe needs a reference point z^* to be defined upfront and kept fixed throughout the entire run. Such a point could also be defined as the ideal point of the population and updated adaptively during the run, similarly to what done in MOEA/D-DE. In the implementation here discussed this is not done, and z^* is defined as the origin of the axis, which is appropriate as we experiment with DTLZ and ZDT problems. The online update of the z^* point is indeed an area of improvement for PaDE and it is a delicate issue as it must be implemented as to not break the island asynchronicity, while still providing an effective adaptation mechanism for z^*.

2.1 Comparison to a Traditional MOEA/D Implementation

PaDe has several advantages compared to a traditional MOEA/D implementation: the main one being its simple and effective parallelization. In fact, MOEA/D is a steady state algorithm as each individual must be evaluated in sequence. This limit the possibility to solve the subproblems in parallel as it requires synchronization between the nodes and communication at each generation. A simple transformation of MOEA/D-DE into a generational variant, on the other hand, simply degrades too much performances. In PaDe we use asynchronous migration to help each problem with solutions from the neighbours. This approach is easier to parallelize since each problem is independently solved by every island and communication between islands doesn't need to apply at each generation, while the original performances are kept.

Moreover the island model used by PaDe is asynchronous. That means that each island migrates when it is ready, without waiting for other islands, and each island can receive immigrants at any moment. This and the fact that communication doesn't need to happen at every generation, is particularly helpful when PaDe is deployed on an heterogeneous parallel architecture mixing slow and fast nodes, as for example a grid, where its performances, though, would be affected and need to be evaluated.

The main disadvantage of PaDE is in the overhead caused by the internal island model, and by the population based evolutionary algorithm used on each subproblem, an overhead that is only justified when the decomposed problems are hard, and thus the deployment of generic, state of the art, single objective evolutionary algorithm on the decomposed problems is justified.

Finally PaDe is a flexible and modular algorithm: the decomposition and the optimization of each single-objective subproblem are two phases that can be designed independently. In our implementation [14] we propose a novel method for the former (LOW-DISCREPANCY, see Section 2), while for the latter we provide many well known single objective solvers (Covariance Matrix Adaptation Evolutionary Startegy, Differential Evolution, PSO, Harmony Search, ...).

3 Experiments

The aim of the experiments is to investigate the parallel scalability of PaDe and whether this appraoch is competitive with other common MOEA/D implementations from a quality of the final Pareto Front point of view. First, we compare PaDe (using a self-adaptive version of Differential Evolution as a solver) with the MOEA/D-DE algorithm [7]. The implementations we used for both the algorithms, are available as part of the open source scientific library PaGMO [14]. We then investigate the parallel performances of PaDe showing how increasing the number of threads up to the population size PaDe is able to fully take advantage of the underlying cores.

3.1 Performance Measures

We have considered the following performance measures

- *p-distance*: measures the average distance between points on a non dominated front and the optimal Pareto Front [17]. The indicator can only be defined for ZDT and DTLZ problems and is zero if the non dominated front belongs to the Pareto Front. A smaller p-distance is better. A zero p-distance means that all points are on the Pareto Front.

– *Hypervolume*: the hypervolume of the Pareto Front calculated according to a reference point shared between both the algorithms and all the runs for the same benchmark problem. The reference point is chosen as the largest point to box all the final non dominated fronts, or equivalently as the nadir point of the union of the populations of all the runs and all the algorithms for the same benchmark problem. Since the reference point is different for each problem we report the hypervolumes normalized as follow $\widetilde{h_i} = (h_i - min(h_1, h_2))/max(h_1, h_2)$, where h_1 and h_2 are the hypervolume for PaDe and MOEA/D-DE respectively. When the normalized hypervolume is zero, it indicates that the algorithm achieved the smallest hypervolume. To know how much smaller, one has to read the normalized hypervolume of the other algorithm. Small values will indicate, essentially, that the same quality has been achieved.
– *Fitness Evaluations* this is simply the number of calls to the fitness functions throughout one run. In PaDe the number of fitness evaluations cannot be fixed in advance as the inner algorithm S may have multiple termination conditions (as jDE has). For a fair comparison G_P has been set to get,in almost all cases, a similar number of fitness evaluations between PaDe and MOEA/D-DE.
– *cpu-time* for both the parallel and the sequential experiments this is the wall-clock time for the algorithm to stop after evaluating the given number of generations.

3.2 Sequential Experiments

PaDe running on a single thread has been tested against MOEA/D-DE on the following continuous multi-objective benchmark problems ZDT1, ZDT2, ZDT3, ZDT4, ZDT6 (problem size equal to 30), and for the 3-objectives, 4-objectives and 5-objectives version of DTLZ1, DTLZ2, DTLZ3, DTLZ4, DTLZ5, DTLZ6, DTLZ7 (problem size equals to 30). For both the algorithms the decomposition method is Tchebycheff, and weight vectors are generated using GRID (see Section 2). The population size has been set to 100 for the 2-objectives problems, 105 for the 3-objectives ones, 120 for the 4-objectives ones and 126 for the 5-objectives one to accommodate the limitation imposed by the grid method (see Section 2). The number of neighbours is set to T=15.

PaDe runs for $G_P = 20$ generations on $n = 1$ threads, the solver S used is a self-adaptive differential evolution jDE [18] running for $G_S = 100$ generations. The best/1/exp variant is used. MOEA/D-DE uses a crossover probability equal to 1, it uses both the diversity preserving mechanism described in [7] and it runs for $G_D = 32000$ generation. This has been chosen to have a comparable number of generations between PaDe and MOEA/D. In fact PaDe needs to run the solver on m populations of size $T + 1$. That means that approximately, the number of fitness evaluations required by PaDe is $G_P \cdot m \cdot (T+1) \cdot G_S$, while MOEA/D-DE will perform approximately $m \cdot G_D$ fitness evaluations. We run the two algorithms 30 times on each problem. We report the median of the measures presented in Section 3.1 on Table 1.

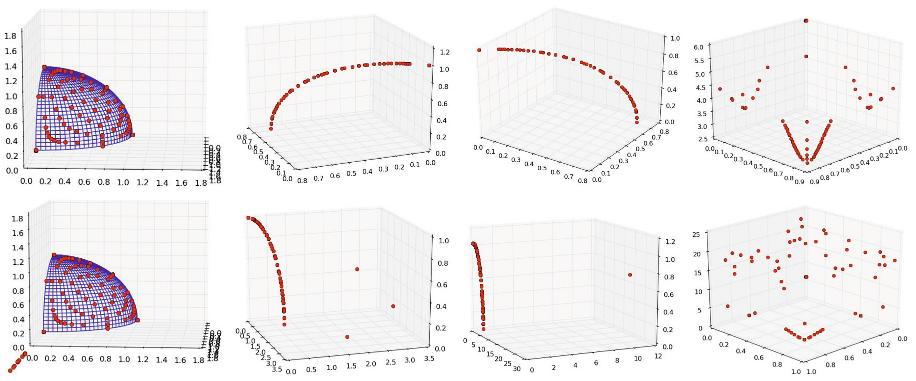

Fig. 1. Comparison between the Pareto Fronts found by MOEA/D-DE (first row) and PaDe (second row) on the 3-objective problems in which PaDe gets a worse p-distance than MOEA/D-DE [DTLZ4,5,6,7 respectively in 1st, 2nd, 3rd and 4th column]

The two algorithms have comparable performances from a quality of the fronts point of view. In Fig. 1 we show the Pareto Fronts for the problems in which PaDe achieves higher p-distance than MOEA/D. Excluding DTLZ7 for which MOEA/D-DE is much better, for the other problems the gap between MOEA/D-DE and PaDe in term of p-distance is due to few point which don't converge to the optimal front rather than to the whole Pareto Front not converging. We suspect this happens because MOEA/D-DE adapts the reference point throughout the run, while PaDe fixes it to the origin. We don't plot the Pareto Fronts for the problems in which PaDe is better because they look very similar to the the ones obtained by MOEA/D. From Table 1 we can also notice that the way we set G_S and G_P is appropriate, since it leads to a comparable number of fitness evaluations between MOEA/D and PaDe. Hypervolumes are still very similar between the two methods, while in terms of cpu-time PaDe is faster even if it is executed on a single thread.

3.3 Parallel Experiments

The scalability of PaDe has been tested running it on a 8-core machine with hyperthreading for increasing n (number of threads). Results for the ZDT, DTLZ-3obj, DTLZ-4obj, DTLZ-5obj problems are summarized in Fig. 2. We define the speedup as the ratio between the running time using an increasing number of threads and the running time using just one thread. As all the problems tested do not require to access neither memory nor other peripherals, hyperthreading is not expected to help, thus the maximum theoretical speed-up achievable on the tested architecture is $s_M = 8$.

We see in Fig. 2 how increasing the number of threads up to the population size PaDe is able to fully take advantage of the underlying cores and to get very close to the linear speedup of 8 for most of the problems tested.

Table 1. Sequential experimental results. In grey the best results among the two algorithms. See Section 3.1 for a description of the performance measures.

	p-distance		normalized hypervolume		cpu-time		Fitness evaluations	
Prob/Alg	PaDe	MOEA/D-DE	PaDe	MOEA/D-DE	PaDe	MOEA/D-DE	PaDe	MOEA/D-DE
ZDT1	1.56e-06	2.04e-05	7.69e-06	0.00e+00	5.83	12.34	3151320	3200000
ZDT2	9.77e-07	9.94e-06	0.00e+00	1.77e-05	5.70	12.31	3150480	3200000
ZDT3	2.26e-01	2.73e-05	0.00e+00	8.65e-05	5.33	13.01	2523240	3200000
ZDT4	8.89e-09	8.01e-04	2.01e-04	0.00e+00	15.39	21.95	2967360	3200000
ZDT6	4.15e-01	2.38e-04	0.00e+00	9.89e-05	6.48	13.43	2801200	3200000
DTLZ-3obj	1.23e-09	3.77e-03	0.00e+00	1.20e-06	20.42	30.45	3050240	3360000
DTLZ2-3obj	1.38e-11	3.27e-04	0.00e+00	2.02e-05	11.34	20.38	2976920	3360000
DTLZ3-3obj	5.18e-10	1.42e-03	0.00e+00	2.56e-05	21.08	32.56	2968840	3360000
DTLZ4-3obj	3.00e-01	8.59e-04	0.00e+00	1.22e-05	12.41	21.95	2952000	3360000
DTLZ5-3obj	7.16e-02	3.53e-05	0.00e+00	4.31e-05	10.37	20.88	2692640	3360000
DTLZ6-3obj	5.39e-01	6.93e-06	0.00e+00	4.37e-04	20.46	29.89	3341920	3360000
DTLZ7-3obj	4.44e-01	2.29e-04	0.00e+00	5.84e-03	6.29	20.43	1774400	3360000
DTLZ1-4obj	3.68e-10	2.73e-03	0.00e+00	1.89e-05	23.49	40.49	3518080	3840000
DTLZ2-4obj	3.14e-11	1.43e-04	0.00e+00	9.62e-05	14.60	30.06	3422600	3840000
DTLZ3-4obj	2.99e-10	8.12e-04	0.00e+00	6.16e-05	26.12	44.63	3411480	3840000
DTLZ4-4obj	6.37e-01	6.71e-04	0.00e+00	2.13e-05	16.61	31.82	3308120	3840000
DTLZ5-4obj	1.19e+00	9.07e-01	0.00e+00	1.33e-05	12.82	33.21	2938800	3840000
DTLZ6-4obj	3.62e+00	3.13e+00	0.00e+00	5.26e-04	25.33	76.12	3811040	3840000
DTLZ7-4obj	1.42e+00	8.65e-04	0.00e+00	7.74e-03	8.76	28.47	2421680	3840000
DTLZ1-5obj	2.51e-10	2.20e-03	0.00e+00	2.13e-05	25.06	46.64	3752200	4032000
DTLZ2-5obj	7.29e-13	4.98e-05	0.00e+00	1.07e-04	17.75	38.20	3691240	4032000
DTLZ3-5obj	1.90e-12	2.22e-03	0.00e+00	5.76e-05	30.10	53.63	3673960	4032000
DTLZ4-5obj	1.06e+00	9.13e-05	0.00e+00	1.94e-05	21.25	41.26	3566560	4032000
DTLZ5-5obj	2.20e+00	1.84e+00	9.66e-04	0.00e+00	14.08	33.54	2802120	4032000
DTLZ6-5obj	6.70e+00	6.47e+00	0.00e+00	1.06e-05	28.78	75.43	3968080	4032000
DTLZ7-5obj	1.46e+00	1.42e-03	0.00e+00	5.47e-03	10.04	35.14	2669200	4032000

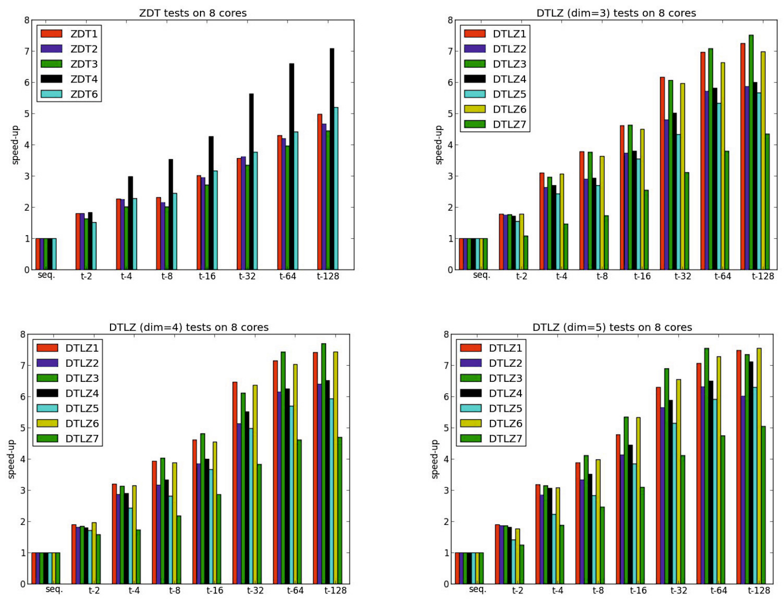

Fig. 2. The number of threads used versus the speedup (ratio between the cpu-time obtained with one thread and the cpu-time obtained with that amount of threads)

4 Conclusions

We have introduced a new multi-objective evolutionary approach, based on the MOEA/D framework and on the island model. The new algorithm, called PaDe is compared to MOEA/D-DE. Experiments show that PaDe can find good approximations to the Pareto Fronts on the tested problems in shorter time than MOEA/D-DE even when deployed on one single CPU. When deployed on multiple CPU architectures, and, unlike fine-grained parallelization approaches for MOEA/D-DE, PaDe is able to provide considerable (close to linear) speed ups also with non CPU intensive fitness landscapes. Moreover the asynchronous island model at the core of PaDe, make the algorithm suitable for modern multicores architectures as well as for heterogeneous parallel architecture in which slow nodes are mixed with fast ones (e.g. grid computing).

As future work PaDe should be tested on harder problems, where it could perform better than MOEA/D from a quality of the Pareto Front point of view, and a parallel-safe mechanism to adapt the reference point z^* throughout the run should be implemented to make PaDe competitive also for problems for which setting z^* to the origin is not a sensible choice.

References

1. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast elitist multi-objective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6, 182–197 (2000)
2. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In: Proceedings of the EUROGEN 2001 Conference (2001)
3. Zhang, Q., Li, H.: MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. IEEE Trans. Evolutionary Computation 11(6), 712–731 (2007)
4. Miettinen, K.: Nonlinear Multiobjective Optimization. International series in operations research & management science. Kluwer Academic Publishers (1999)
5. Das, I., Dennis, J.: Normal-boundary intersection: An alternate method for generating pareto optimal points in multicriteria optimization problems (1996)
6. Jiang, S., Cai, Z., Zhang, J., Ong, Y.S.: Multiobjective optimization by decomposition with pareto-adaptive weight vectors. In: Seventh International Conference on Natural Computation, ICNC 2011 (2011)
7. Li, H., Zhang, Q.: Multiobjective optimization problems with complicated pareto sets, MOEA/D and NSGA-II. IEEE Transactions on Evolutionary Computation 13(2), 284–302 (2009)
8. Al Moubayed, N., Petrovski, A., McCall, J.: A novel smart multi-objective particle swarm optimisation using decomposition. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6239, pp. 1–10. Springer, Heidelberg (2010)
9. Nebro, A.J., Durillo, J.J.: A Study of the Parallelization of the Multi-Objective Metaheuristic MOEA/D. In: Blum, C., Battiti, R. (eds.) LION 4. LNCS, vol. 6073, pp. 303–317. Springer, Heidelberg (2010)
10. Durillo, J.J., Zhang, Q., Nebro, A.J., Alba, E.: Distribution of Computational Effort in Parallel MOEA/D. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 488–502. Springer, Heidelberg (2011)
11. Tomassini, M.: Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time. Springer (2005)
12. Izzo, D., Ruciński, M., Biscani, F.: The generalized island model. In: Fernandez de Vega, F., Hidalgo Pérez, J.I., Lanchares, J. (eds.) Parallel Architectures & Bioinspired Algorithms. SCI, vol. 415, pp. 151–170. Springer, Heidelberg (2012)
13. Mambrini, A., Sudholt, D., Yao, X.: Homogeneous and heterogeneous island models for the set cover problem. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part I. LNCS, vol. 7491, pp. 11–20. Springer, Heidelberg (2012)
14. PaGMO: Parallel Global Multiobjective Optimizer, `http://pagmo.sourceforge.net/pagmo/`
15. PyGMO: Python Parallel Global Multiobjective Optimizer, `http://pagmo.sourceforge.net/pygmo/`
16. Halton, J.H.: Algorithm 247: Radical-inverse quasi-random point sequence. Commun. ACM 7(12), 701–702 (1964)
17. Märtens, M., Izzo, D.: The asynchronous island model and NSGA-II: study of a new migration operator and its performance. In: Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference, pp. 1173–1180. ACM (2013)
18. Brest, J., Zumer, V., Maucec, M.S.: Self-adaptive differential evolution algorithm in constrained real-parameter optimization. In: IEEE Congress on Evolutionary Computation, CEC 2006, pp. 215–222. IEEE (2006)