Novel Efficient Asynchronous Cooperative Co-evolutionary Multi-Objective Algorithms

Sune S. Nielsen*, Bernabé Dorronsoro†, Grégoire Danoy*, Pascal Bouvry*

*Faculty of Sciences, Technology and Communication, University of Luxembourg, 6 rue Richard Coudenhove-Kalergi, L-1359, Luxembourg
Emails: {sune.nielsen, gregoire.danoy, pascal.bouvry}@uni.lu
†Interdisciplinary Centre for Security, Reliability, and Trust, University of Luxembourg, 6 rue Richard Coudenhove-Kalergi, L-1359, Luxembourg
Email: bernabe.dorronsoro@uni.lu

Abstract—This article introduces asynchronous implementations of selected synchronous cooperative co-evolutionary multi-objective evolutionary algorithms (CCMOEAs). The CCMOEAs chosen are based on the following state-of-the-art multi-objective evolutionary algorithms (MOEAs): Non-dominated Sorting Genetic Algorithm II (NSGA-II), Strength Pareto Evolutionary Algorithm 2 (SPEA2) and Multi-objective Cellular Genetic Algorithm (MOCell). The cooperative co-evolutionary variants presented in this article differ from the standard MOEAs architecture in that the population is split into islands, each of them optimizing only a sub-vector of the global solution vector, using the original multi-objective algorithm. Each island evaluates complete solutions through cooperation, i.e., using a subset of the other islands current partial solutions. We propose to study the performance of the asynchronous CCMOEAs with respect to their synchronous versions and base MOEAs on well known test problems, i.e. ZDT and DTLZ. The obtained results are analyzed in terms of both the quality of the Pareto front approximations and computational speedups achieved on a multicore machine.

Index Terms—Asynchronous parallel design, Cooperative co-evolutionary EAs, Multi-objective optimization, Superlinear speedup

I. INTRODUCTION

Multi-objective evolutionary algorithms deal with the simultaneous optimisation of multiple objective functions, usually in conflict, for a given problem. Improving one objective then implies decreasing (some of) the others. This gives rise to the definition of the Pareto-front composed of non-dominated solutions in the objective space (a solution dominates another one if the former is better or equal to the latter for all the considered objectives, but strictly better in at least one objective). The goal of any multi-objective algorithm is to find a diverse set of solutions in variable space which are non-dominated and as close to the optimal Pareto-front as possible in objective space.

Co-evolution is an approach where the population is split in so called sub-populations and the fitness evaluation depends on interactions with other sub-populations. Co-evolutionary algorithms can be competitive [1] or cooperative [2] depending on the sub-population components working together or against each other when optimizing their local fitness functions. This study is based on the cooperative co-evolution framework as introduced by Potter in [3] in which each sub-population, or island, evolves a subset of the decision variables with a standard EA. Complete solutions are evaluated in each sub-population by cooperation, i.e. by exchanging individuals with all other sub-populations.

The main contribution of this work is the development of a new asynchronous design of the three cooperative co-evolutionary multi-objective evolutionary algorithms (CCMOEAs) proposed in [4] which were demonstrated to achieve super-linear speedup compared to their respective MOEAs counterparts. The three CCMOEAs are based on the following state-of-the-art multi-objective evolutionary algorithms (MOEAs): Non-dominated Sorting Genetic Algorithm II (NSGA-II) [5], Strength Pareto Evolutionary Algorithm 2 (SPEA2) [6], and Multi-Objective Cellular Genetic Algorithm (MOCell) [7]. The framework implemented in [4] is synchronous and therefore does not fully take advantage of the parallelisation possibilities offered by the coevolutionary design. For this reason we propose in this work a new asynchronous design of the CCMOEAs, aiming at providing even larger speedups than the synchronous CCMOEAs, while still obtaining highly competitive results.

The remainder of this article is organised as follows. The next section introduces the literature found related to asynchronous GAs and the statements made w.r.t. possible time gains to be achieved with an asynchronous implementation. Then in Section III the multi-objective base algorithms used for the cooperative co-evolutionary implementations of this article are presented. Section IV, first introduces co-evolutionary algorithms followed by a more detailed description of the synchronous and asynchronous implementations providing the foundation for the experiments and tests in this article. The results of our experiments are detailed and analysed in Section V. Finally in Section VI we present our conclusions and perspectives.

II. RELATED WORK

In the vast field of evolutionary algorithms, the speedup effects achieved from parallel implementations have been explored thoroughly. Indeed, as outlined several times by Talbi in [8], synchronous versus asynchronous exchange of
individuals will have significant impact on the execution time in distributed systems. As an example, in [9] an asynchronous cellular genetic algorithm (cGA) is shown to provide superior performance on combinatorial problems: MAXSAT, MMDP and p-Median. Hart, et al. [10] also analyse the effect of relaxed synchronisation on Geometrically Structured GAs (GSGAs) in terms of asynchronous communication, or synchronous communication at a reduced frequency. However, the potential speed-up achievable by making a cooperative co-evolutionary synchronous genetic algorithm asynchronous has been seldom studied in the related literature. Potter and De Jong, whose work on cooperative co-evolution is the foundation of the developed algorithms in this article, characterize their algorithm as quasi-coevolutionary in [3] and [11]. This refers to the fact that their cooperative coevolutionary genetic algorithm (CCGA) synchronises after each generation. They state that a fully asynchronous version that communicates only occasionally and/or completely asynchronously would be possible and interesting for further studies - especially for scenarios where populations are run on separate CPUs. The work of Hart, et al. [10], also referenced by [12] in the context of a CCMOEAs, states that improvements could be done on the area of tuning and limiting solutions broadcast. In their work however, a synchronous approach is applied to study the effect of co-evolution rather than to study solution broadcast synchronisation. In [13] Craciun, et al. analyse aspects of synchronised and asynchronous cooperative co-evolution. The main focus is on scalability for large problems as well as on the number of evaluations required to converge to quality solutions rather than on synchron direct time gain resulting from the asynchronous implementation. Still they conclude that the asynchronous version produces slightly better results with less function evaluations.

The framework proposed in [14] and [4], which we develop further in this article, consists of synchronous CCMOEAs, which achieved super-linear speedup compared to their state-of-the-art MOEA counterparts on continuous and combinatorial problems. Most notably, a 41 times faster execution was achieved for a large Robust Static Mapping Problem (RSMP), consisting in reducing the makespan and increasing the robustness of a schedule of 2048 tasks on 64 machines.

In the field of CCMOEAs, the effect of changing the broadcast frequency has not been studied directly to our knowledge, which justifies this study comparing asynchronous with synchronous broadcasting in CCMOEAs.

III. Base Algorithms

The multi-objective base algorithms used in this work are the following state-of-the-art multi-objective evolutionary algorithms: Non-dominated Sorting Genetic Algorithm II (NSGA-II), Strength Pareto Evolutionary Algorithm 2 (SPEA2) and Multi-objective Cellular Genetic Algorithm (MOCell).

A. Non-dominated Sorting Genetic Algorithm (NSGA-II)

NSGA-II [5], is characterized mainly by the selection of best individuals sorted by dominance and crowding distance. The offspring is generated with usual genetic selection, crossover and mutation, then parent and offspring generations are combined to a set of size $2N$ and the best individuals are chosen to produce a new population $P_t$ of size $N$. This ensures elite solutions to survive and the crowding distance sorting ensures that chosen solutions are well spread.

B. Strength Pareto Evolutionary Algorithm (SPEA2)

SPEA2 [6], also uses genetic selection, crossover and mutation like NSGA-II. However in SPEA2, the fitness of an individual is assigned from the sum of individuals it dominates. All non-dominated solutions, kept in an archive, and the parent population are copied into the offspring population. If this results in a population greater than the allowed size, the population is truncated iteratively by removing individuals having smallest distance to their $k$ nearest neighbors. On the other hand, if the combined population is too small the best dominated solutions from the archive and parent population are copied to the offspring population.

C. Multi-objective Cellular Genetic Algorithm (MOCell)

MOCell [7] is characterized by the separation of solutions in cells interconnected in a toroidal grid. Only solutions in neighboring cells are allowed to interact, which prevents good solutions from spreading quickly and hence maintains solution diversity. In this article the MOCell variant aMoCell4 is used, where a dominant solution from the archive is used as a parent during the breeding loop.

IV. THE PROPOSED ASYNCHRONOUS CCMOGAS

This section describes co-evolution and the changes done to create asynchronous implementation of three synchronous CCMOEAs.

A. Co-evolutionary Algorithms

The principle of competitive co-evolution in evolutionary algorithms was introduced by Hillis [1] in the form of a parasite and host model also called the predator-prey model. Around the same time Husband and Mill [2] were the first to use cooperative co-evolution for job-shop scheduling. Shortly after, Potter and De Jong in [3] used cooperative co-evolution for continuous function optimization (CCGA-1), providing the basis framework we extend in this paper. The idea is to split the solution vector and run genetic algorithms on sub-sets of the solution variables. So called islands or sub-populations are then responsible for optimizing only a part of the global solution vector or in some cases only a single solution variable, using an evolutionary algorithm.

Each sub-population will then broadcast its individuals to the other islands after each generation following a selection scheme (e.g. using the best individual). This fully connected broadcast of the solution sub-vectors between all the islands enables each island to assemble and evaluate the resulting
global objective function which is essential for the local genetic algorithm. Fig. 1 shows broadcasting of individuals between three sub-populations of a standard CCEA.

This approach was shown to be especially effective when the problem to be solved contains little epistatic links between the variables [15], i.e., the effect on the fitness function of changing a variable depends little or not at all on the state of other variables in the global solution.

B. Synchronous CCMOEAs

The synchronous co-evolutionary approach described above was implemented in [4] for three state-of-the-art multi-objective algorithms (NSGA-II, SPEA2 and MOCell) and the pseudo code is shown in Algorithm 1. One notable difference between the CCMOEAs and a standard single-objective CCGA is the step that merges Pareto-fronts of the sub-populations to generate an archive of best combined Pareto-fronts, as can be seen in line 15. The merging process of the solution sets found by all the sub-populations is achieved by choosing one of them and adding to it all the solutions from the others. In case the resulting approximation set is full, the crowding policy is used to remove solutions based on the distance to surrounding individuals belonging to the same rank.

Besides the archives management in the subpopulations, another difference of the multi-objective design is the way in which complete solutions are built for evaluation. As previously mentioned, in the case of single-objective optimization, the evaluation of a partial solution in a sub-population is achieved by composing a complete solution with the best partial solutions from all the other subpopulations. In the case of multi-objective optimization, in most cases there will be more than a single best solution, i.e., a set of non-dominated ones. For the proposed CCMOEAs, every subpopulation shares a number $N_s$ of solutions randomly chosen from the non-dominated ones found so far. An example of how one subpopulation, $P_i$, shares its best solutions with the other populations is shown in Fig. 2, where $N_s = 4$. If the local Pareto-front contains less than $N_s$ non-dominated solutions, the set of $N_s$ solutions is completed by other individuals randomly taken from the rest of the population.

Referring to Algorithm 1, line 11, all islands synchronize after each generation to broadcast their selected individuals to the other islands during execution. This results in a short delay whenever any island takes longer time to finish its sub-population generation than the others. The delay can have various reasons depending on the algorithm itself, but simply the randomness in the basic genetic operators can produce a slight variation in the execution time of a generation in one island compared to another. Therefore we propose in this work a new asynchronous design to better take advantage of the parallel efficiency of the CCMOEAs by removing all synchronization points in the algorithm execution.

C. Asynchronous CCMOGAs

In Algorithm 2 the modified pseudo algorithm is displayed, showing the removal of the cross-thread synchronization point. Now the islands can independently evaluate their populations without waiting for all others to finish and send their best solutions. A result of this algorithmic change is that the islands will not share the total number of available objective function evaluations equally between them. Hence a fast executing island might continue on and ‘steal’ evaluations from the slower islands. But as suggested in the previous section, even if all islands do ‘consume’ the same amount of the total function evaluations, the asynchronous version should remove delays and allow the algorithm to finish faster. Another consequence is that an island may start evaluating its individuals before the other islands have migrated their individuals, resulting in eval-

---

**Algorithm 1** Parallel CCMOEAs framework

1: $t \leftarrow 0$
2: // $z$ means parallel run
3: $i \in [1, I] \triangleright \text{setup}(P^0, i)$ // Initialize every subpopulation
4: sync() // Synchronization point
5: // $\forall$ means sequential run
6: $\forall i \in [1, I] \triangleright \text{broadcast}(P^0, i)$ // Share random local partial solutions in every subpopulation
7: $i \in [1, I] \triangleright \text{evaluate}(P^0, i)$ // Evaluate solutions in every subpopulation
8: sync()
9: while not stoppingCondition() do
10: $i \in [1, I] \triangleright \text{generation}(P^t, i)$ // Perform one generation to evolve the population
11: sync()
12: $\forall i \in [1, I] \triangleright \text{broadcast}(P^t, i)$ // Share best local partial solutions in every subpopulation
13: $t \leftarrow t + 1$ // Increase generations counter
14: end while
15: mergeParetoFronts() // Merge the Pareto fronts found in the subpopulation into a single one

---
is analysed further in the experiments presented in the next section. How the issues discussed influence the final result of the compared algorithms presented here. To what extent the islands differ in the number of variables that are processed at the same time, and how this affects the performance of the algorithms, is not analyzed here. Clearly, the smaller island would result in one island having only two variables to process, while the larger island would have more variables to process.

Algorithm 2 Asynchronous CCMOEAs framework

1: \( t \leftarrow 0 \)
2: \( \begin{array}{l}
   \text{// \# means parallel run}
   \end{array} \)
3: \( i \in [1, I] \); setup( \( P^0, i \) ) \( \quad \) // Initialize every subpopulation
4: \( \text{sync()} \) \( \quad \) // Synchronization point
5: \( \begin{array}{l}
   \text{// \# means sequential run}
   \end{array} \)
6: \( \forall i \in [1, I] \); broadcast( \( P^0, i \) ) \( \quad \) // Share random local partial solutions in every subpopulation
7: \( i \in [1, I] \); evaluate( \( P^0, i \) ) \( \quad \) // Evaluate solutions in every subpopulation
8: \( \text{sync()} \)
9: \( i \in [1, I] \); \{ \)
10: \( \text{while not stoppingCondition()} \) do
11: \( \text{generation( } P^t, i \text{ )} \) \( \quad \) // Perform one generation to evolve the population
12: \( \text{broadcast( } P^t, i \text{ )} \) \( \quad \) // Share best local partial solutions in every subpopulation
13: \( t \leftarrow t + 1 \) \( \quad \) // Increase generations counter
14: \} \text{end while}
15: \( \text{mergeParetoFronts()} \) \( \quad \) // Merge the Pareto fronts found in the subpopulation into a single one

The experiments were run on a dedicated node at the HPC facility of the University of Luxembourg. The node used was a HP ProLiant BL2x220c G6 (10U) with 2 Intel L5640 CPU’s having 6 cores each at 2.26 GHz. The experiments were run in a way ensuring that the whole node was reserved and that the experiment was run strictly on the first 4 cores of the second CPU. Hence the delay effects studied stem from the individual islands different generational execution times, rather than from communication bandwidth limits or network latencies. This effect is especially important for multi-objective algorithms because of the extra load related to archiving and sorting solution sets and approximating the Pareto-front.

A. Test problems

The three multi-objective algorithm variants NSGA-II, SPEA2 and MOCell were tested with the standard test suites ZDT and DTLZ.

1) Zitzler-Deb-Thiele (ZDT): The ZDT test suite presented in [16] contains 6 test problems with convex, non-convex and non-continuous Pareto-fronts as well as deceptive local fronts and non-uniformly distributed solutions on the optimal Pareto-front. For the experiments we used 32 decision variables for ZDT1 to ZDT3 and 12 for the rest.

2) Deb-Thiele-Laumanns-Zitzler (DTLZ): The DTLZ test suite presented in [17] contains test problems with linear, but mostly spherical, Pareto-fronts, also with variants containing non-linear solution density and deceptive local fronts. The final test problem, DTLZ7, is unique in that its Pareto-front is a combination of a line and a hyper-plane. For the experiments we run with 8 decision variables for DTLZ1 and 12 for the rest.

B. Experiment configurations

The experiments were conducted with 50 independent runs for each algorithm and test problem. From the 50 runs, 5% of the fastest and slowest runs were removed and not regarded in the plots and tables. Termination condition used is 100,000 function evaluations for all algorithms.

The CCMOEAs use 4 subpopulations, each of them being evolved in parallel in a dedicated thread running on one of the 4 CPU cores used. The population size for the state-of-the-art MOEAs is 100, as usually in the literature, and every subpopulation is composed of 100 solutions in the studied CCMOEAs. Islands are exchanging 20 randomly chosen local non-dominated solutions, and to evaluate a given solution, it is built with random sub-solutions from those shared by the other subpopulations. This migration policy was proposed in [4] after intensive experimentations.

Finally, the rest of the parameters are shared amongst all the compared algorithms, and their values are commonly used in the literature, as the random initialization of the population, binary tournament selection, simulated binary crossover (SBX, applied with probability 0.9 and index 20), and polynomial mutation (with probability one over the number of variables and index 20). All parameters are summarized in table I.
TABLE I
Configuration of the co-evolutionary algorithms NSGA-II, SPEA2 and MOCell.

<table>
<thead>
<tr>
<th>Number of subpopulations</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores used</td>
<td>4 (1 for canonical MOGAs)</td>
</tr>
<tr>
<td>Number of threads</td>
<td>1 per subpopulation</td>
</tr>
<tr>
<td>Population size</td>
<td>100</td>
</tr>
<tr>
<td>Archive size*</td>
<td>100 per subpopulation</td>
</tr>
<tr>
<td>Final archive size</td>
<td>100, chosen among all subpops.</td>
</tr>
<tr>
<td>Migration policy **</td>
<td>20 rand [4]</td>
</tr>
<tr>
<td>Max. evaluations</td>
<td>100,000</td>
</tr>
<tr>
<td>Population initialization</td>
<td>Random</td>
</tr>
<tr>
<td>Selection</td>
<td>Binary tournament</td>
</tr>
<tr>
<td>Recombination</td>
<td>SBX</td>
</tr>
<tr>
<td>Probability</td>
<td>0.9</td>
</tr>
<tr>
<td>Crossover index</td>
<td>20</td>
</tr>
<tr>
<td>Mutation</td>
<td>Polynomial</td>
</tr>
<tr>
<td>Probability</td>
<td>( \frac{1}{\text{number of variables}} )</td>
</tr>
<tr>
<td>Mutation index</td>
<td>20</td>
</tr>
<tr>
<td>Independent runs</td>
<td>50</td>
</tr>
</tbody>
</table>

* Not applicable for NSGA-II
** Not applicable for MOEA

C. Quality measures used

To evaluate the found solution sets, the following quality measures are used. This is to ensure that the speed-up achieved does not come at the price of solution quality degradation.

1) **Hypervolume (HV):** The Hypervolume metric [18], defined in Eq. 1, is a calculation combining the hyper-cubes spanning from each solution to a dominated reference point in solution space.

\[
HV = \text{volume} \left( \bigcup_{i=1}^{Q} v_i \right) \tag{1}
\]

The reference point used contains all the worst objective values found and one hyper-cube is constructed by using the reference point and a solution point on \( Q \) in opposite diagonal corners. By taking the union of all hyper-cubes \( v_i \), the hypervolume is constructed, which gives a strong indication of the volume spanned by the solution set \( Q \). The larger the value, the closer the solution set found is to the Pareto front.

2) **Inverted Generational Distance (IGD):** The IGD presented in [19] and defined in Eq. 2 is the average Euclidean distance from the found solutions to the Pareto-front.

\[
IGD = \sqrt{\frac{\sum_{i=1}^{k} d_i^2}{n}} \tag{2}
\]

where \( d_i \) is the Euclidean distance from point \( i \) in the Pareto front approximation found to the closest one in the optimal Pareto front, and \( n \) is the number of solutions in the front. If \( IGD=0 \), all solutions are actually on the Pareto front.

D. Results

In this section, we discuss the experimental results in terms of time gains, and impact on the quality measures compared to the original algorithm.

1) **Time gains:** The main goal of this work was to achieve a speedup in execution time, resulting from the asynchronous algorithmic modifications discussed above. Fig. 3 graphically shows some selected results w.r.t. execution time by comparing the previously developed synchronous CCMOEAs (prefixed \( s \)) and the newly developed asynchronous CCMOEAs (prefixed \( a \)) with the base MOEAs (NSGA-II, SPEA2 and MOCell). Table II summarizes the average execution time in milliseconds of all the experiments conducted as well as the resulting speedup factors. The factors are calculated from the values in the table:

\[
\frac{t_{MOEA}}{t_{Async}} / \frac{t_{MOEA}}{t_{Sync}} / \frac{t_{Sync}}{t_{Async}}
\]

In each speedup column of Table II, the highest speedup is emphasized with dark grey background and the lowest speedup with light gray background. The speedup factors of the asynchronous compared to the synchronous implementation range from 1.13 for SPEA2 on DTLZ6 to 1.90 for NSGAI1 on ZDT3. In average the speedup is 1.41 times with a standard deviation of 0.16. For asynchronous versions the speedup achieved compared to the corresponding original MOEA was from 3.90 to 7.78 times for SPEA2 on DTLZ6 and MOCell on ZDT1, respectively. The synchronous version achieved 2.71 to 6.44 times faster execution for NSGAI1 on ZDT3 and MOCell on ZDT2, respectively.

The average speedup for asynchronous and synchronous implementations are 5.54 and 3.99 with standard deviations 1.08 and 1.13 respectively. Therefore, the proposed asynchronous algorithms perform in average super-linear speed-ups (remember that we are using four islands and, consequently, four cores) while their synchronous counterparts offer linear speedups in general.

These results demonstrate a significant speedup for all algorithms and all test problems, both comparing the original MOEA with the sCCMOEA [4] and the sCCMOEA with the developed aCCMOEA. We presume that these results could be still further seriously improved when solving computationally intensive problems, as it is the case for the scheduling problem addressed in [4]. This will be considered in some future work.

2) **Solution Quality:** This section presents the performance of the developed algorithms in terms of the solutions quality calculated using the previously mentioned quality indicators Hypervolume (HV) and Inverted Generational Distance (IGD). In order to assess statistical confidence in the comparisons with each performance indicators, we perform the Wilcoxon test [20] with standard confidence level (0.95). This enables us to tell if the found indicator values are significantly different, and if this is the case, the difference in their mean value is used to indicate improvement or decrease. Table III shows this as ‘▲’ and ‘▼’ if the indicator showed performance increase, decrease or ‘.' if no significant difference can be assessed.
The two ‘Delta’ columns show calculations of the percentage of change between the two preceding data value columns. For example, the first Delta column under HV shows increase from MOEA with respect to the synchronous CCMOEAs results, the next Delta the increase from synchronous to asynchronous CCMOEAs values. The same is the case for Delta columns under the IGD indicator. Note again that high HV and low IGD values are preferred, refer to section V-C for details. The best delta-values are marked with a dark gray background and the worst with light gray.

When interpreting the table, it is clear that for both HV and IGD indicators there is an overall slight increase in favor of the synchronous CCMOEAs version with respect to the base MOEA. When comparing the synchronous CCMOEAs with the developed asynchronous CCMOEAs, there are a lot more result sets with insignificant change. This implies that the results found are less significantly different than those of the CCMOEAs version with respect to the base MOEA, which was expected. However the asynchronous version shows many results significantly worse than the synchronous version on the HV measure, especially for ZDT test problems, though even for these results the difference in the mean value found remains relatively small, mostly below $|0,05|$. On IGD the asynchronous and synchronous versions show very similar performance still with a slight favor for the synchronous version.

VI. CONCLUSION

In this paper, we introduced a fully asynchronous cooperative co-evolutionary implementation for multi-objective optimization (aCCMOEA) and compared it to the synchronous versions (sCCMOEAs) previously based on three state-of-the-art multi-objective algorithms, i.e. Non-dominated Sorting Genetic Algorithm II (NSGA-II), Strength Pareto Evolutionary Algorithm 2 (SPEA2) and Multi-objective Cellular Genetic Algorithm (MOCell) [4]. The improvement in terms of speedup from the asynchronous implementation compared to the synchronous one was found to be 1.41 on average and up to 1.90. This is an additional speedup on top of the one originating from the cooperative co-evolutionary implementation previously shown in [4], hence a total speedup of up to almost 8 times, 5.5 on average, was achieved on the continuous functions of ZDT and DTLZ compared to the standard state-of-the-art MOEA implementation. The speedups are achieved not only without a significant general deterioration of the approximated Pareto fronts, but with significant improvements in some cases.

As future works, we plan to compare the performance of the proposed parallel aCCMOEAs to the sCCMOEAs on a real-world combinatorial problem, namely the robust static mapping problem of independent tasks on Grids (RSMP).

ACKNOWLEDGMENT

This work has been funded by the Fonds National de la Recherche Luxembourg, CORE project Green-IT (C09/IS/05) and and the UL-EvoPerf project. Experiments presented in this paper were carried out using the HPC facility of the University of Luxembourg.
### TABLE III

<table>
<thead>
<tr>
<th>Test</th>
<th>Alg.</th>
<th>MOEA</th>
<th>Sync</th>
<th>Delta</th>
<th>Async</th>
<th>IGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTLZ1</td>
<td>NSGAII</td>
<td>4.9277E-01</td>
<td>4.9355E-01</td>
<td>0.16 ▲</td>
<td>4.9369E-01</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>4.9426E-01</td>
<td>4.9303E-01</td>
<td>-0.07 V</td>
<td>4.9295E-01</td>
<td>-0.33</td>
</tr>
<tr>
<td>DTLZ2</td>
<td>NSGAII</td>
<td>4.1078E-01</td>
<td>4.1192E-01</td>
<td>0.54 ▲</td>
<td>4.2194E-01</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>2.1187E-01</td>
<td>2.1208E-01</td>
<td>0.01 ▲</td>
<td>2.1206E-01</td>
<td>-0.02</td>
</tr>
<tr>
<td>DTLZ3</td>
<td>NSGAII</td>
<td>2.0989E-01</td>
<td>2.0755E-01</td>
<td>-1.32 ▲</td>
<td>2.0653E-01</td>
<td>-0.01</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>2.0962E-01</td>
<td>1.9870E-01</td>
<td>-5.21 ▲</td>
<td>2.0071E-01</td>
<td>1.01</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>2.1159E-01</td>
<td>2.0893E-01</td>
<td>-2.63 ▲</td>
<td>2.0844E-01</td>
<td>-0.23</td>
</tr>
<tr>
<td>DTLZ4</td>
<td>NSGAII</td>
<td>1.6735E-01</td>
<td>1.8699E-01</td>
<td>11.74 ▲</td>
<td>1.7763E-01</td>
<td>-5.01</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>1.4951E-01</td>
<td>1.4034E-01</td>
<td>-6.13 ▲</td>
<td>1.3706E-01</td>
<td>23.32</td>
</tr>
<tr>
<td>DTLZ5</td>
<td>NSGAII</td>
<td>2.1087E-01</td>
<td>2.1195E-01</td>
<td>0.51 ▲</td>
<td>2.1198E-01</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>2.1187E-01</td>
<td>2.1207E-01</td>
<td>0.09 ▲</td>
<td>2.1207E-01</td>
<td>0.00</td>
</tr>
<tr>
<td>DTLZ6</td>
<td>NSGAII</td>
<td>1.8136E-01</td>
<td>2.0974E-01</td>
<td>15.65 ▲</td>
<td>2.064E-01</td>
<td>0.43</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>1.8652E-01</td>
<td>2.1176E-01</td>
<td>13.53 ▲</td>
<td>2.1012E-01</td>
<td>-0.77</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>1.7335E-01</td>
<td>1.8255E-01</td>
<td>5.31 ▲</td>
<td>1.8052E-01</td>
<td>-1.12</td>
</tr>
<tr>
<td>DTLZ7</td>
<td>NSGAII</td>
<td>3.3365E-01</td>
<td>3.3433E-01</td>
<td>0.21 ▲</td>
<td>3.3434E-01</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>3.3419E-01</td>
<td>3.3431E-01</td>
<td>0.04 ▲</td>
<td>3.3427E-01</td>
<td>-0.01</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>3.2675E-01</td>
<td>3.2415E-01</td>
<td>-0.79 ▲</td>
<td>3.3190E-01</td>
<td>2.39 ▲</td>
</tr>
<tr>
<td>ZDT1</td>
<td>NSGAII</td>
<td>6.0602E-01</td>
<td>6.1612E-01</td>
<td>0.21 ▲</td>
<td>6.160E-01</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>6.0618E-01</td>
<td>6.1618E-01</td>
<td>0.02 ▲</td>
<td>6.1618E-01</td>
<td>-0.02</td>
</tr>
<tr>
<td>ZDT2</td>
<td>NSGAII</td>
<td>6.0207E-01</td>
<td>6.1628E-01</td>
<td>0.05 ▲</td>
<td>6.1621E-01</td>
<td>-0.02</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>3.2874E-01</td>
<td>3.2838E-01</td>
<td>0.02 ▲</td>
<td>3.2838E-01</td>
<td>-0.02</td>
</tr>
<tr>
<td>ZDT3</td>
<td>NSGAII</td>
<td>5.1534E-01</td>
<td>5.1584E-01</td>
<td>0.08 ▲</td>
<td>5.1584E-01</td>
<td>-0.01</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>5.1569E-01</td>
<td>5.1576E-01</td>
<td>0.02 ▲</td>
<td>5.1576E-01</td>
<td>-0.01</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>5.1601E-01</td>
<td>5.1600E-01</td>
<td>0.00 ▲</td>
<td>5.1598E-01</td>
<td>0.00</td>
</tr>
<tr>
<td>ZDT4</td>
<td>NSGAII</td>
<td>3.9083E-01</td>
<td>4.0067E-01</td>
<td>0.66 ▲</td>
<td>4.0058E-01</td>
<td>-0.02</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>4.0092E-01</td>
<td>3.9979E-01</td>
<td>-0.24 ▲</td>
<td>3.9915E-01</td>
<td>-0.21</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>4.0091E-01</td>
<td>4.0133E-01</td>
<td>0.01 ▲</td>
<td>4.0110E-01</td>
<td>-0.06</td>
</tr>
</tbody>
</table>

### REFERENCES