# An Efficient Parallel Version of Dynamic Multi-Objective Evolutionary Algorithm

Maroua Grid
Computer Science Department,
Barika University Centre, Algeria
maroua.grid@gmail.com

Leyla Belaiche
Computer Science Department,
Biskra University, Algeria
leila.belaiche@gmail.com

Laid Kahloul
Computer Science Department,
Biskra University, Algeria
l.kahloul@univ-biskra.dz

Saber Benharzallah
Computer Science Department,
Batna 2 University, Algeria
sbharz@yahoo.fr

**Abstract:** *Multi-Objective Optimization Evolutionary Algorithms (MOEAs) belong to heuristic methods proposed for solving Multi-objective Optimization Problems (MOPs). In fact, MOEAs search for a uniformly distributed, near-optimal, and near-complete Pareto front for a given MOP. However, several MOEAs fail to achieve their aim completely due to their fixed population size. To overcome this shortcoming, Dynamic Multi-Objective Evolutionary Algorithm (DMOEA) [20] was proposed. Although DMOEA has the distinction of dynamic population size, it still suffers from a long execution time. To deal with the last disadvantage, we have proposed previously a Parallel Dynamic Multi-Objective Evolutionary Algorithm (PDMOEA) [10] to obtain efficient results in less execution time than the sequential counterparts, in order to tackle more complex problems. This paper is an extended version of [10] and it aims to demonstrate the efficiency of PDMOEA through more experimentations and comparisons. We firstly compare DMOEA with other multi-objective evolutionary algorithms Non-Dominated Sorting Genetic Algorithm (NSGA-II) and Strength Pareto Evolutionary Algorithm (SPEA-II), then we present an exhaustive comparison of PDMOEA versus DMOEA and discuss how the number of used processors influences the efficiency of PDMOEA. As experimental results, PDMOEA enhances DMOEA in terms of three criteria: improving the objective space, minimizing the computational time, and converging to the desired population size. Finally, the paper establishes a new formula relating the suitable number of processes, required in PDMOEA, and the number of necessary generations to converge to the optimal solutions.*

**Keywords:** *Multi-objective problems, pareto front, multi-objective evolutionary algorithms, dynamic MOEA, parallel DMOEA.*

*Received April 10, 2022; accepted April 28, 2022*
*https://doi.org/10.34028/iajit/19/3A/2*

## 1. Introduction

Nowadays, real-world problems have more than one objective to be achieved because of their complexity. Such that, each objective is specified using a mathematical formula called the objective function. The resolution process of these problems requires performing a procedure of decision making because it looks simultaneously for a set of approximated solutions to each objective function, and these functions may be in conflict. To get the optimal decision, some trade-offs between the conflicting objectives are required. This kind of problem is called a Multi-objective Optimization Problem (MOPs). Multi-objective Evolutionary Algorithms (MOEAs) are one of the heuristic methods which are intended for MOPs.

MOEAs share the common purpose which is searching for a uniformly distributed, near-optimal, and well-extended Pareto front for a given MOP. Figure 1 [7] depicts graphically the concept of Pareto optimal solutions. In this case, two variables *x1*, *x2* and two objective functions *f1*, *f2* are considered. Thus, the objective space is defined in the space of objective functions and the space of variables defines what we call the decision space. Indeed, many researchers have exploited MOEAs to resolve several kinds of MOPs, such as works in the optimization of reconfigurable manufacturing systems [4, 13, 19].

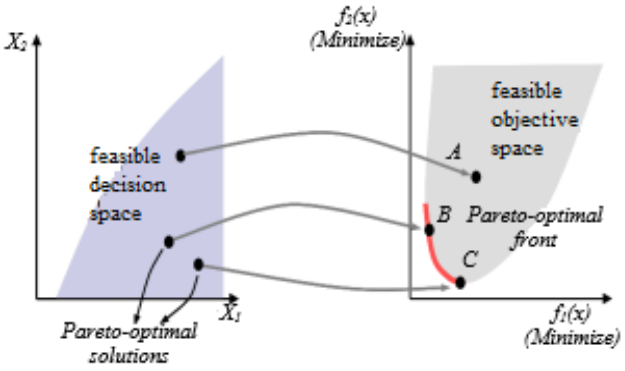



Figure 1. Graphical depiction of pareto optimal solution [10].

However, finding a suitable Pareto front was difficult to reach by several of the existing MOEAs. Their most important reason for not achieving their aim completely is the fixed population size which makes evolutionary algorithms suffer from premature convergence if the population size is too small, whereas an overestimated population size will result in a heavy burden on computation and a long computational time for fitness improvement. For this reason, the dynamic multi-objective evolutionary algorithm (i.e., DMOEA) [20] was proposed, in the literature, with population size varying dynamically during the run-time of the algorithm using adaptive cell-based rank density estimation.

DMOEA was proposed to obtain a Pareto front with a desired resolution because the shape and size of the true Pareto front are unknown a priory for most of the MOPs. Although DMOEA gives excellent results compared to other MOEAs Non-Dominated Sorting Genetic Algorithm (NSGA-II) and Strength Pareto Evolutionary Algorithm (SPEA-II), being an evolutionary algorithm means that it will certainly be characterized by a long execution time. It cannot be ignored that the user is always looking for the best results in less time. On another hand, MOEAs algorithms (including DMOEA) require many cycles to reach their convergence. To overcome these shortcomings, one issue is to propose parallel versions of those algorithms. One of the main reasons for using Parallel Evolutionary Algorithms (PEAs) is to obtain efficient results with an execution time less than one of their sequential counterparts in order to tackle more complex problems. This naturally leads to measuring the speedup of the PEA. PEAs have sometimes been reported to provide super-linear performances for different problems.

In our previous published paper [10], we have proposed a new parallel version of DMOEA (i.e., PDMOEA). Indeed, PDMOEA enhances DMOEA in terms of three criteria (i.e., improving the objective space, minimization of computational time, and converging to the desired population size) for satisfying the user requirements (i.e., getting optimal solutions in minimum execution time). This paper is an extended version of [10] which aims to provide more experimentations on the use of DMOEA and its parallel version PDMOEA. We provide firstly a comparison between DMOEA and two other existing algorithms NSGA-II [8] and SPEA-II. [21]. after that, we provide an experimental comparison between the sequential version DMOEA and the parallel version PDMOEA. In this comparison, we work on changing the number of processes (2, 4, and 8) implied in the parallel version, in order to study how the performance of PDMOEA depends on the number of processes and which gain will be better. Finally, this paper establishes a new formula that relates the suitable processes number to be used in PDOMA to the generation's number required for the convergence of the algorithm.

The rest of this paper is organized as follows. Section 2 presents the most relevant related work. Section 3 introduces DMOEA. Section 3 presents the proposed parallel version of DMOEA (i.e., PDMOEA). After that, the results and an experimental study are discussed in Section 5. Finally, Section 6 provides the conclusion that evaluates the results and discusses some perspectives.

## 2. Literature Review

Many researchers were interested to propose, implement and parallelize multi-objective algorithms [2, 3, 4]. Indeed, in the literature, many state-of-the-art multi-objective evolutionary algorithms were parallelized. In the following paragraphs, we discuss some of the most relevant work in the parallelization of MOEA.

González-Álvarez *et al*. [9], address the discovery of repeated common patterns as a multi-objective optimization problem by means of a hybrid MPI/Open MP [1] (i.e., Message Passing Interface/Open Multi-Processing) approach which parallelizes a well-known multi-objective meta-heuristic, the fast NSGA-II. Their main objective was to combine the benefits of shared-memory and distributed-memory programming paradigms to discover patterns in an accurate and efficient manner. They have proposed a hybrid parallel approach based on MPI and Open MP to parallelize a modified version of NSGA-II when finding common patterns on a set of amino acid sequences. This new modified algorithm, named Hybrid Non-Dominated Sorting Genetic Algorithm (H-NSGA-II), combines the evolutionary properties of the well-known NSGA-II with a local search function specialized in improving the quality of the predictions. The complexity of the addressed optimization problem Nondeterministic Polynomial Time Hardness (NP-hard: ppv) has motivated this research, mainly designed to take advantage of current hardware architectures Symmetric Multiprocessing (SMP) clusters with many cores.

In [17], two parallel multi-objective meta-heuristics, NSGA-II and Strength Pareto Evolutionary Algorithm (SPEA2), have been applied to tackle the inference of phylogenetic trees considering two criteria: maximum parsimony and maximum likelihood. These algorithms were implemented with Open MP to take advantage of multi-core processor systems, with the aim of distributing time-consuming inference/evaluation operations among execution threads. The resulting parallel approaches were evaluated by conducting experimentation on four real data sets, examining speedup factor and phylogenetic results by means of well-known parallel and multi-objective metrics.

Dagostino *et al*. [6], presented a fine-grained parallelization of the Fast NSGA-II for The Compute Unified Device Architecture (CUDA). In particular,

they discussed how this solution can be exploited to solve multi-objective optimization tasks in the field of computational and systems biology.

Most of the well-known MOEAs have been parallelized using different parallel technologies for improving their performance. Compared to state-of-the-art MOEAs, DMOEA is found to be competitive in terms of maintaining diversity and improving the objective space [20]. However, DMOEA suffers from a long computational time. In this work, we propose a parallel version of DMOEA (i.e., PDMOEA) to enhance DMOEA in terms of three criteria:

1. Improving the objective space.
2. Minimizing the computational time.
3. Converging to the desired population size.

For implementing PDMOEA, we adopted the threads model of parallel computing. Experimentations are provided in this work to demonstrate the performance of the parallel proposed version versus the sequential existing one.

## 3. Dynamic Multi-Objective Optimization Evolutionary Algorithm (Dmoea)

MOEAs common purpose is to find a set of optimal solutions by using several strategies, to balance the diversity, and the convergence to the Pareto-optimal solutions. However, this ultimate goal is far from being accomplished by all MOEAs. One of the negative aspects that made them move away from their goals, is the fixed population size to initiate the evolutionary process. Indeed, it is often hard and inefficient to solve MOPs by MOEAs with fixed population sizes, since they have to homogeneously distribute the predetermined computation resource to all the possible directions in the objective space [16]. In fact, MOEAs may suffer from premature convergence if the population size is too small, whereas if the population size is too large, undesired computational resources may be incurred and the computational time for a fitness improvement may be too long in practice [18].

Dynamic multi-objective evolutionary algorithm (DMOEA) is an evolutionary algorithm, which was proposed by Lu and Yen in [15]. DMOEA's aim is to obtain a Pareto front with a desired resolution because the shape and size of the true Pareto front are unknown a priory for most of the MOPs [20].

DMOEA is based on adaptive cell-based rank density estimation, where the objective space is considered as a grid; the number of its cells depends on the dimension of the MOP and the grid-scale in each dimension. DMOEA consists of four main functions; the first function is the initial cell-based rank density calculation scheme which initializes principal variables as initial population size to create a population of initial individuals, calculates cells width, initializes objective functions, and creates a grid that contains a number of cells defined in each dimension depending on the grid-scale given by the multi-objective problem. The second function is a population growth strategy which aims to make the population size bigger than the initial one to increase the probability of obtaining better individuals. After that, DMOEA uses crossover and mutation functions to create new offspring. These functions are good for creating more individuals which means an increase in the population size. However, this increase will not stop without the intervention of another function to reduce the continuous reproduction of individuals. This function is the population declining strategy which represents the third main function of DMOEA. It reverses the work of the population growth strategy by decreasing population size by using the killing technique to kill each individual that has an age bigger than the age threshold defined by the MOP, to get finally the desired population with the desired size. The last main function is the objective space compression strategy to get the real objective space of Pareto front individuals.

DMOEA should stop by examining the 3 stopping criteria [20]:

- The rank values of all cells are one,
- The objective space cannot be compressed anymore,
- And finally, each resulting non-dominated cell contains *ppv* individuals (*ppv* stands for a given population size per unit volume).

The reason for this step is due to the fact that another criterion, the Pareto ranks of all resulting individuals equal to one, should be satisfied as well to guarantee there is no dominance relationship among the resulting Pareto solutions at the final generation. Taking into consideration all procedures of DMOEA, Figure 2 shows the flowchart of DMOEA algorithm steps.

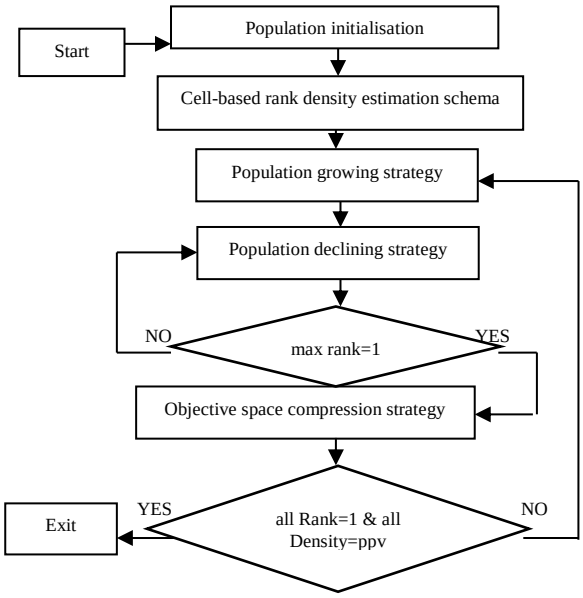



Figure 2. DMOEA algorithm flowchart [10].

## 4. A Parallel Version of Dynamic Multi-Objective Optimization Evolutionary Algorithm (Pdmoea)

The proposed Parallel Dynamic Multi-Objective Optimization Evolutionary Algorithm (PDMOEA) is based on a multiprocessing framework and master\slave technique. The master process creates *n* processes. As it is shown in Figure 3 PDMOEA divides the Generations Number (NbG) into *n* processes, and each process executes DMOEA with *NbG/n* generations. Then, the best results of one of the processes are selected.

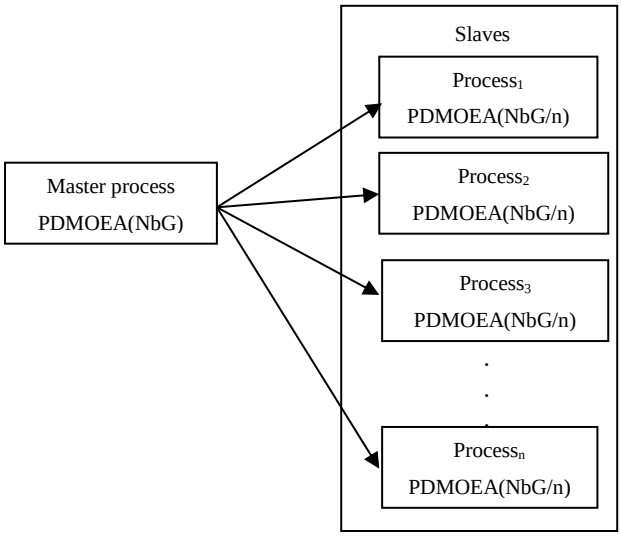



Figure 3. Division framework of generations' number of PDMOEA [10].

PDMOEA algorithm has the same main functions as DMOEA but the only difference is that PDMOEA divides generation's number into *n* threads. After dividing the generation number by the *n* threads each process will create an initial population and makes it grow and decline until it gets the desired population size with the best solutions. Figure 4 depicts the global architecture of the PDMOEA implemented system.

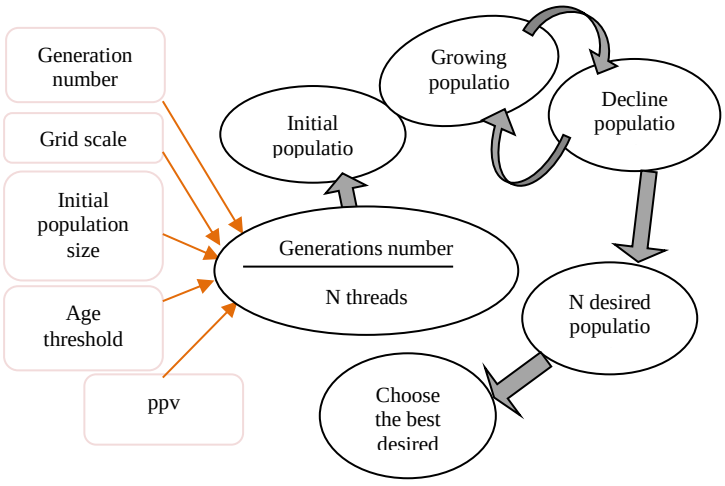



Figure 4. Parallel global design [10].

Indeed, in this work, we have already implemented a sequential version of DMOEA following an object programming style. Then, in the parallel version we have added a new method, called PDMOEA, to the class Population of the program (See Figure 5).

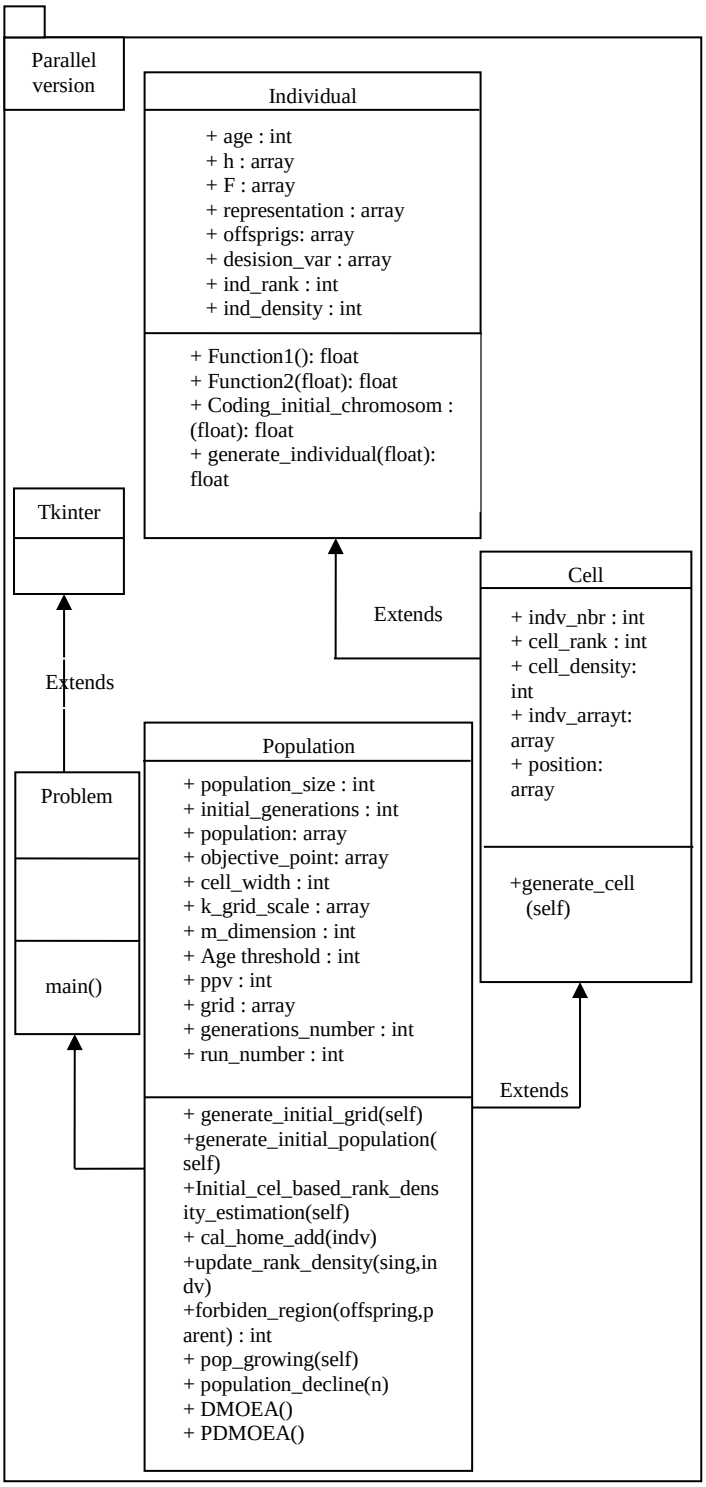



Figure 5. Parallel DMOEA version classes diagram [10].

*Algorithm 1 PDMOEA method*

```
q:=mp.Queue()
poolprocess:=[]
i:=1
while i<=n do
    p:=mp.Process(target=self.mainloop)
    p.start()
    poolprocess.append(q.get())
    p.join()
    i:=i+1
end while
```

## 5. Experimental Study

In this experimental study, we aim to compare the performance of the two versions of DMOEA; the sequential one versus the parallel one PDMOEA. For testing the proposed Parallel DMOEA (i.e., PDMOEA), the multi-objective problem presented in [30] is used as a case study. This problem was designed by Deb as the preliminary test function that has a discontinuous Pareto front. Equations 1 and 2 give the two objective functions of the considered problem. This problem represents a minimization problem. It aims to minimize $f1(x_1, x_2)$ and $f2(x_1, x_2)$, where:

$$f_1(x_1, x_2) = x_1 \tag{1}$$

$$f_2(x_1, x_2) = (1 + x_2) \times \left(1 - \frac{x_1}{1+x_2}\right)^2 - \frac{x_1}{1+x_2} \times 10\pi \frac{x_1}{1+x_2} \tag{2}$$

Subject to $0 \leq x1, x2 \leq 1$.

### 5.1. Parameter Settings

Both DMOEA and PDMOEA are implemented using the Oriented Object Programming (OOP) paradigm and using a set of software and hardware which are summarized in Table 1.

Table 1. Software/hardware versions [10].

| Software/Hardware | Version |
|---|---|
| OS | Microsoft Windows 10 Professional, 64bits, version version 10.0.17134 |
| CPU | Intel(R) Core(TM) i7-4500U CPU @1.80GHz 2.40GHz |
| RAM | 8.00Go |
| Python Interpreter | 3.7.0 |
| PyCharm | 2016.3.1 |
| matplotlib | 2.0.0 |
| Tkinter | 8.6 |

### 5.2. Comparative factors

Comparative factors mean indexes that we observe each time as outputs of each execution. These factors are, in fact, the results obtained every time we execute both versions. They will be mentioned in the following subsections in this experimental study.

1. *Desired population size*: as mentioned in paper [30] the desired population size at generation *n* noted *dps(n)*, with the desired population size per unit volume *ppv* (a user-specified parameter), and an approximated number of trade-offs hyper-areas $A_{to}$, is defined as:

$$lowbps < dps(n) = ppv \times A_{t0}(n) < upbps \tag{3}$$

Where *low bps* and *up bps* are the lower and upper bounds for the desired population size *dps(n)*, respectively. In paper [5], the authors applied a method used to estimate the approximated number of hyper-areas by:

$$A_{t0} \approx \frac{\frac{\pi(m-1)}{2}}{\frac{m-1}{2}!} \times \left(\frac{d(n)}{2}\right)^{(m-1)} \tag{4}$$

Where *m* is the number of objectives and *d(n)* is the diameter of the hypersphere at generation *n*. In DMOEA, instead of estimating the trade-off hyper-areas $A_{to}$ at each generation *n*, we concentrate on searching for a uniformly distributed and well extended final set of trade-off hyper-areas and ensure that each of these areas contains *ppv* number of non-dominated individuals. Therefore, by using DMOEA, the optimal population size and final Pareto front will be found simultaneously at the final generation.

Desired population size *dps(n)* is approximated and when we approach it will be better. To get a fair comparison, this value is calculated in the same way in both versions in DMOEA and PDMOEA.

2. *Improving the objective space*: MOEAs are often run with the goal of approximating the whole Pareto front and most MOEAs are designed to do this on problems of arbitrary parameter space and objective space dimension [11]. Today, some test functions are scalable in both parameter and objective dimension; and some performance indicators are also suitable for many objective problems. These advances have made it possible to compare performance of MOE algorithms when the number of objectives is scaled up beyond the typical two or three. As our studied MOP is to minimize the objective space, the best result will be the result of the one who minimized more than the other as an improvement of the objective space. We can explain it as the standard definition of Pareto dominance in the objective space is used. Assuming minimization, without loss of generality: *x* dominates *y*.
3. *Computational time*: one of the main reasons for using PEAsis to obtain efficient algorithms with an execution time much lower than that of their sequential counter parts in order to tackle more complex problems. This naturally leads to measuring the speedup of the PEA. PEAs have sometimes been reported to provide super-linear performances for different problems [12].

### 5.3. Results and Discussion

Indeed, several multi-objective optimization evolutionary algorithms (MOEA) are proposed to search for a uniformly distributed, near-optimal, and near-complete Pareto front for a given MOP. However, this common goal is far from being accomplished by all MOEAs. In this study, we have implemented two other MOEAs (NSGA-II and SPEA-II) which do not use dynamic population size. Using these implementations, we provide a comparative study of sequential DMOEA versus (NSGA-II and SPEA-II), then a comparative study between DMOEA and PDMOEA. Finally, we studied how the increase of processes number in PDMOEA influences its performance.

1. DMOEA Versus NSGA-II and SPEA-II

In this section, we compare DMOEA with two other multi-objective evolutionary algorithms namely NSGA-II and SPEA-II. Table 2 lists the specific parameter settings for the function of the studied MOP [20]. Table 3 illustrates the parameters setting for all the test functions as global parameters.

Table 2. Specific parameter setting for mop 1.

| | **Initial pop size** | **External pop size** | **ppv** | **Grid Scale** | **Age threshold** | **Tourn-ament size** |
|---|---|---|---|---|---|---|
| DMOEA | 2 | - | 3 | (30,30) | 10 | - |
| NSGA-II | 100 | 0 | - | - | - | 2 |
| SPEA-II | 80 | 20 | - | - | - | - |

Table 3. Specific parameter setting for all the test functions.

| | |
|---|---|
| Crossover rate | 0.7 |
| Maximum number of generations | 10000 |
| Number of runs | 50 |

The results of this comparison are represented in the following Figures 6, 7, 8. Figure 6 represents the final objective space and the Pareto front of NSGA-II, Figure 7 represents the objective space as a final result of SPEA-II, and Figure 8 shows the result of executing DMOEA based on previous setting parameters in Tables 2, 3.

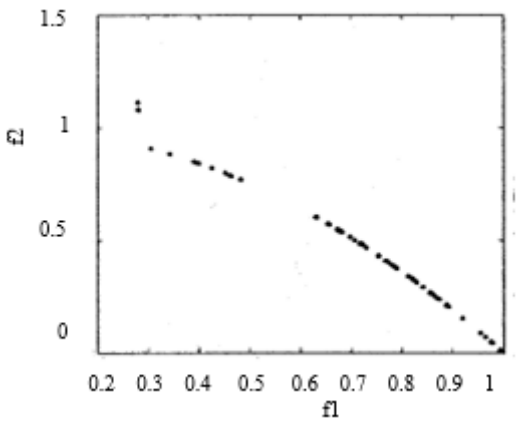



Figure 6. NSGA-II pareto front.

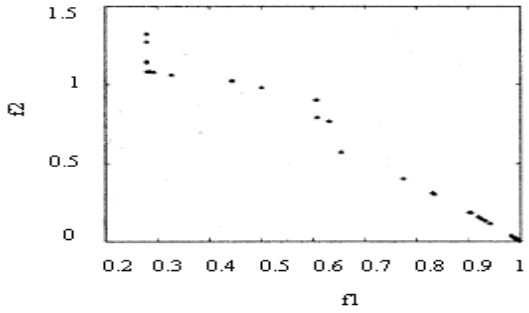
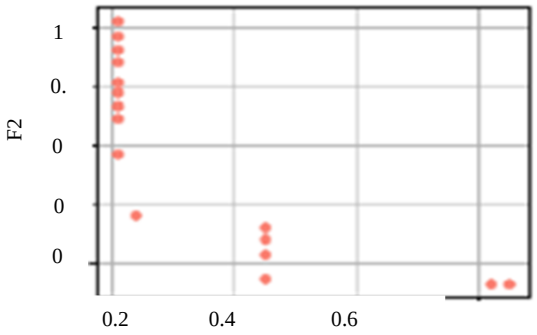



Figure 7. SPEA-II pareto front




Sequential Execution
Time=156.2923698009995

Figure 8. DMOEA pareto front.

Considering the three figures, it is clear by a visual observation that the objective space was improved in DMOEA better than the other MOEAs.

2. DMOEA Versus PDMOEA

In order to compare the efficiency and the performance of our parallel version PDMOEA to the sequential DMOEA, using the implementation of the studied MOP [20], we adopt the parameters in Table 4.

Table 4. Specific parameter setting for DMOEA and PDMOEA.

| | |
|---|---|
| Initial population size | 30 |
| Grid scale 1 | 50 |
| Grid scale 2 | 100 |
| Dimension | 2 |
| Generation number | 10000 |
| Age threshold | 10 |
| *ppv* | 5 |
| Runs number | 30 |

It is worth noting that the most important thing we have talked about is the generation number input because as we had seen before in PDMOEA the number of a generation will be divided into four processes, thus each process takes a quarter of this generations number. Hence, we start with a big generation's number of DMOEA and PDMOEA to avoid that division into four threads will not be adversely affected by the desired population that we will have.

We have compared the results of both versions and we concluded that in each execution of both versions using the same inputs, PDMOEA performs better than DMOEA at least in one of the three outputs that the user aims to improve. To prove the power of the evolutionary

algorithm using parallel computing, we have to use the formula in eq. 4 to calculate the trade-off hyper-areas $A_{to}$ of the last generation based on the desired population number *dps (n)*. Whenever the trade-off hyper-areas $A_{to}$ is bigger, the PDMOEA will be better in desired population size comparison factor. Also, time is considered better when it was moreminimized. We recall that we keep using the same parameters illustrated in Table 2 for both DMOEA and PDMOEA.

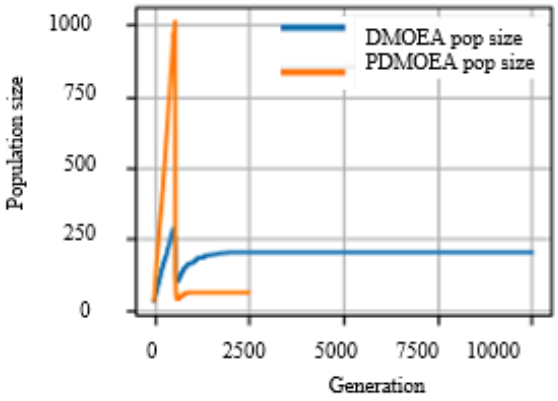



Figure 9. Population size comparison (DMOEA vs PDMOEA).

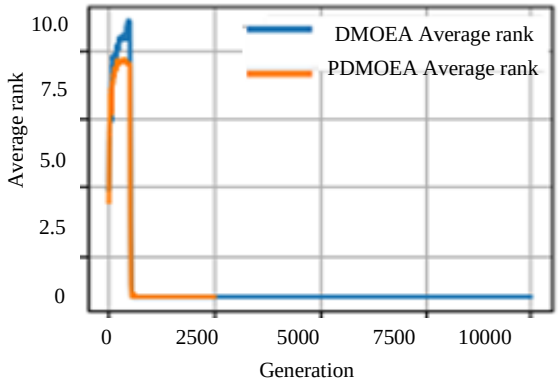



Figure 10. Average rank comparison (DMOEA vs PDMOEA).

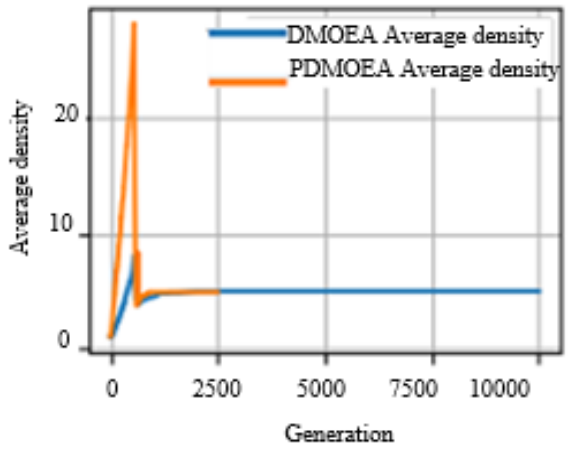



Figure 11. Average density comparison (DMOEA vs PDMOEA).

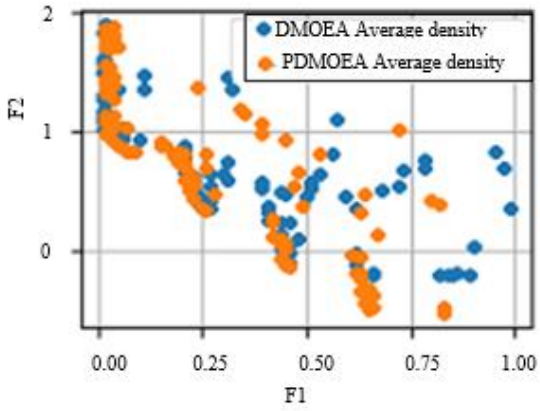



Figure 12. Objective space comparison (DMOEA vs PDMOEA).

The obtained results are represented in Figure 6 collecting DMOEA and PDMOEA curves. These results depict the comparison of performance metrics (i.e., population size, rank, density, and the objective space). In both DMOEA and PDMOEA, rank converges to be one, density converges to the desired population size per unit volume *ppv* and the desired population size is obtained finally. However, DMOEA executes all the generations (i.e., 10000 generations) but PDMOEA executes just 2500 generations which improves the CPU running time. The obtained population size from running PDMOEA achieves a smaller size than in DMOEA. The objective space obtained by running PDMOEA is improved better than DMOEA because PDMOEA guarantees the convergence to smaller values.

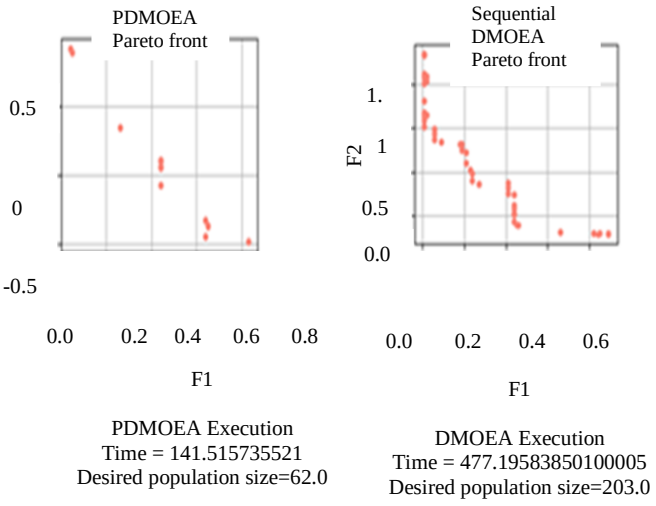



Figure 13. Objective space and running CPU time comparison.

Figure 13 represents the improvement of objective space and the running CPU time. PDMOEA outperforms DMOEA such that it improves the objective space in a short running CPU time. However, the number of solutions in the Pareto front obtained from running PDMOEA is smaller than those obtained from running DMOEA.

From Figure 13 in DMOEA the desired population size *dps (n)* is 62. Using eq.3, the trade-off hyper-areas $A_{to}$ is approximately equal to 12 based on the desired population per unit volume *ppv* supposed to be 5 (see Table 5. While the desired population size *dps (n)* obtained from PDMOEA is 203 and the trade-off hyper-areas $A_{to}$ is approximately equal to 40. Thus, the desired population size of PDMOEA is better than DMOEA. On another hand, the execution time in PDMOEA is better four times than DMOEA, and the objective space of PDMOEA is improved compared to DMOEA.

3. The effect of the processes number on the performance of PDMOEA

Programming distributed-memory multiprocessors and networks of workstations requires deciding what can be executed concurrently, how processes communicate, and where data is placed. These decisions can be made statically by a programmer or the compiler, or they can be made dynamically at run time [14].

In this paper, we have experimented PDMOEA on the Mutli-Objective Problem described in [20]. In the previous sections, we have provided experimentation results with a number of processes fixed to 4. This section aims to study the impact of increasing the number of processes on the performance of PDOMEA. Three configurations are compared: 2, 4, and 8 processes. Table 5. Shows settings used in this experimentation. The results of this experimentation are illustrated in Figures 8, 9, 10.

Table 5. Specific parameter setting for PDMOEA.

| | |
|---|---|
| Initial population size | 2 |
| Grid scale 1 | 50 |
| Grid scale 2 | 100 |
| Dimension | 2 |
| Generation number | 16000 |
| Age threshold | 10 |
| *ppv* | 5 |
| Run number | 1 |
| Processes number | 2, 4, 8 |

Figure 8. Shows the results of DMOEA versus PDMOEA when the number of processes in parallel version is 2.

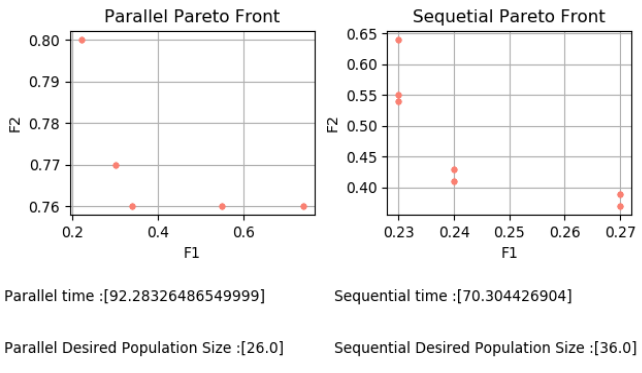



Figure 14. Results when processes number = 2.

Figure 15 shows the results of DMOEA versus PDMOEA when the number of processes in parallel version is 4.

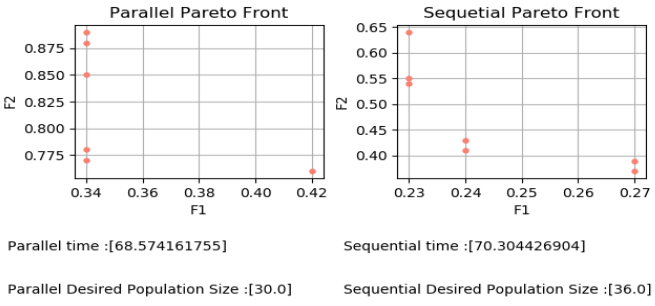



Figure 15. Results when processes number = 4.

Figure 15 shows the results of DMOEA versus PDMOEA when the number of processes in parallel version is 8.

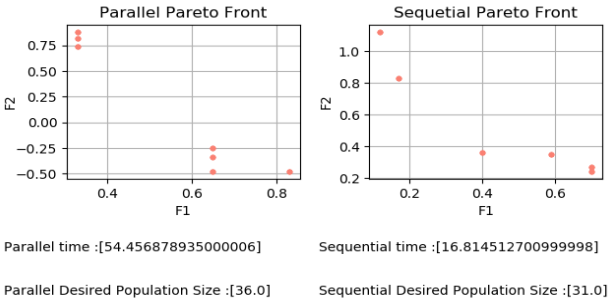



Figure 16. Results when processes number = 8.

$$\frac{Generations\ number}{Processes\ number} > 1000 \qquad (5)$$

We observe that each time we increase the number of processes, we get better results especially the computational time. However, this benefit is limited by a a threshold (a specific number of processes) which makes the generations number in each process converge to be less than 1000 because the decline population strategy starts when the number of generations is more than 500 and it takes around 500 generations to get the desired population size. We conclude that the increase of processes number is beneficial while the number of each process achieves formula in equation5.

## 6. Conclusions

Multi-Objective Optimization Evolutionary Algorithms (MOEAs) may be computationally quite demanding because instead of searching for a single optimum, one generally wishes to find the whole front of Pareto-optimal solutions.

We have implemented a DMOEA on the same test functions proposed in [20] to get the optimal set of results compared with other evolutionary algorithms. Without forgetting that the main objective of this study is not limited to implement this DMOEA with its sequential version and being satisfied with its results,

but the main aim is to search for improved results in term of minimizing computational time, improving objective space and converging to the desired population size. Thus, we have proposed and implemented a first parallel version of the DMOEA namely PDMOEA.

We consider that the parallel implementations of MOEAs is an appropriate way for improving results in terms of improving objective space and minimizing computational time. PDMOEA is the first parallel version of the dynamic DMOEA in literature. The proposed PDMOEA outperforms DMOEA in terms of three criteria: improving objective space, minimizing computational time and converging to the desired population size. The increase of processes number in PDMOEA is beneficial while the number of each process achieves a given formula (equation 5 in this paper).

As a future research direction, we intend to apply PDMOEA on a real-world problem and compare it with other state-of the- art parallel MOEAs.

## References


[1] Barker B., "Message Passing Interface (MPI)," *Workshop: High Performance Computing on Stampede*, Ithaca, vol. 262, 2015.

[2] Belaiche L. and Kahloul L., "The Optimal Process Planning for Reconfigurable Manufacturing Systems," *in Proceeding of the International Conference on Mathematics and Information Technology*, Adrar, pp. 309-316, 2017.

[3] Belaiche L., Kahloul L., Benharzallah S., and Hafidi Y., "Multi-objective Optimization-Based Approach for Throughput Maximization in Reconfigurable Manufacturing Systems," *in Proceeding of the 5th International Symposium on Innovation in Information and Communication Technology*, United States, pp. 1-8, 2018.

[4] Belaiche L., Kahloul L., Benharzallah S., and Hafidi Y., "Bi-Objective Framework For Planning A Supply Chain Process in Reconfigurable Manufacturing Systems," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 1675-1680, 2019.

[5] Czarnul P., *Parallel Programming for Modern High Performance Computing Systems*, Chapman and Hall/CRC, 2018.

[6] Dagostino D., Pasquale G., and Merelli I., "A Fine-Grained Cuda Implementation of the Multi-Objective Evolutionary Approach Nsga-Ii Potential Impact for Computational and Systems Biology Applications," *in Proceeding of the International Meeting on Computational Intelligence Methods for Bioinformatics and Biostatistics*, Cambridge, pp. 273-284, 2014.

[7] Deb K., *Multi-objective Optimization Using Evolutionary Algorithms*, John Wiley and Sons, 2001.

[8] Deb K., Pratap A., Agarwal S., and Meyarivan T., "A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-2," *IEEE Transactions on Evolutionary Computation*, vol. 6, n. 2, pp. 182-197, 2002.

[9] González-Álvarez D., Vega-Rodríguez M., and Rubio-Largo Á., "A Hybrid Mpi/Openmp Parallel Implementation of Nsga-Ii for Finding Patterns in Protein Sequences," *Journal of Supercomputing*, vol. 73, no. 6, Pp. 2285-2312, 2017.

[10] Grid M., Belaiche L., Kahloul L., and Benharzallah S., "Parallel Dynamic Multi-Objective Optimization Evolutionary Algorithm," *in Proceeding of the International Arab Conference on Information Technology*, Muscat, pp. 1-6, 2021.

[11] Gropp W., Lusk E., and Skjellum A., *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, 1999.

[12] Gropp W., *Tutorial on Mpi: The Message Passing Interface*, Argonne National Laboratory, 1995.

[13] Houimli M., Kahloul L., and Khalgui M., "Multi-Objective Optimization and Formal Specification Of Reconfigurable Manufacturing System Using Adaptive NSGA-II," *in Proceeding of the 1st International Conference on Embedded and Distributed Systems*, Oran, pp. 1-6, 2017.

[14] Lowenthal D., Freeh V., and Andrews G., "Using Fine-Grain Threads and Run-Time Decision Making in Parallel Computing," *Journal of Parallel and Distributed Computing*, vol. 37, no. 1, pp. 41-54, 1996.

[15] Lu H. and Yen G., "Dynamic Population Size in Multiobjective Evolutionary Algorithms," *in Proceeding of the Congress on Evolutionary Computation*, Honolulu, pp. 1648-1653, 2002.

[16] Mitchell M., *an Introduction to Genetic Algorithms*, MIT Press, 1998.

[17] Santander-Jiménez S. and Vega-Rodríguez M., "Applying Openmpbased Parallel Implementations of Nsga-Ii and Spea2 To Study Phylogenetic Relationships," *in Proceeding of the IEEE International Conference on Cluster Computing*, Madrid, pp. 305-313, 2014.

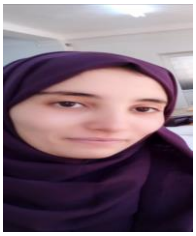
[18] Tan K., Lee T., and Khor E., "Evolutionary Algorithms with Dynamic Population Size and Local Exploration for Multiobjective Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 6, pp. 565-588, 2001.

[19] Torki F., Kahloul L., Hamani N., Belaiche L., and Benharzallah S., "Products Scheduling in Reconfigurable Manufacturing System Considering the Responsiveness Index," *in Proceeding of the 22nd International Arab Conference on Information Technology*, Muscat, pp. 1-6, 2021.

[20] Yen G. and Lu H., "Dynamic Multiobjective Evolutionary Algorithm: Adaptive Cell-Based Rank and Density Estimation," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 253-274, 2003.

[21] Zitzler E., Laumanns M., and Thiele L., "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," *TIK-report*, vol. 103, 2001.

**Maroua Grid** is Assistant Professor at "Si El House" university center in Barika (Batna city). Holder of a doctorate degree (3rd cycle, LMD), in computer science, "image technologies and artificial intelligence", at Mohamed Khider Biskra university in 2018. A member of LESIA Laboratory of Mohamed Khider Biskra University from 2012 to 2018 and currently affiliated with LINFI Laboratory, at Mohamed Khider Biskra University. Her research interests include bio-inspired optimization approaches, Parallel programming, and Deep Learning.



**Leyla Belaiche** is a PhD student in Biskra University. Holder of a master's degree, in computer science, "Software engineering and distributed systems," at Mohamed Khider Biskra University in 2018. She is a member of the LINFI Laboratory of Mohamed Khider Biskra University. Her research domains include bio-inspired optimization approaches, Parallel programming, and reconfigurable manufacturing systems. (https://orcid.org/0000-0003-1128-0725).

**Laid Kahloul** is a professor and researcher in the computer science department of Biskra University (Algeria). Received his Ph. D degree in 2012 from Biskra University (Algeria). Prof. KAHLOUL is currently attached to LINFI (Biskra University). His research interests include software engineering, formal methods, optimization, reconfigurable manufacturing systems, security, IoT, and deep learning applications. He published several papers in his research fields, in ranked conferences and journals. He participated as a reviewer and organizer in several international conferences and he is a reviewer in several indexed journals. (https://orcid.org/0000-0002-9739-7715).

**Saber Benharzallah** is a professor and researcher in the computer science department of Batna 2 University (Algeria). Received his Ph. D degree in 2010 from Biskra University (Algeria). Prof. Benharzallah is currently the director of laboratory LAMIE (Batna 2 University). His research interests include the Internet of things, service-oriented architecture, and context-aware systems. He published several papers in ranked conferences and journals. (https://orcid.org/0000-0002-3870-4383).