Democratic and Popular Republic of Algeria
Ministry of Higher Education and Scientific Research
University of Mohamed Khider - BISKRA
Faculty of Exact Sciences, Natural Sciences and Life
Computer Science Department

Order Number: GLSD4/M2/2019

Thesis

Presented to obtain the diploma of academic Master in

Computer Science

Option: **Software Engineering and Distributed Systems**

Parallelization of Dynamic Multi-objective Optimization Evolutionary Algorithm

By:
BESBAS Asma

Defended the 06/07/2019, in front of the jury composed of:

Djaber Khaled	MAA	President
Kahloul Laid	MCA	Supervisor
Chami Djazia	MAA	Examiner

Acknowledgements

All our thanks and gratitude for Allah the Almighty who gave us all the courage and the will to go till the end of this level.

I owe a major thank to my supervisor **Pr. KAHLOUL Laid** who has always been available and attentive to my questions, my deep respect and thanks for his commitment to giving me the opportunity to carry out this project. Your support and reflections on the work throughout the long-lasting process have been priceless.

Also, I appreciate the support of **Ph.D. student Leila BELAICHE** for her assistance and guidance throughout the planning and execution of this research.

I would also like to express my gratitude to the jury members: **Mr. Djaber Khaled** and **Mrs. Chami Djazia** for reading and evaluating my dissertation.

I never forget to thank all my teachers at Computer Science Department who taught me the basic principles of computer science. Major thanks are also directed to the Head of Department: **Pr. BABAHNINI Mohamed Chawki**.

At last, but not least I would like to thank my family and friends for all their support and understanding during this year of my studies.

Abstract

Multi-objective optimization evolutionary algorithms (MOEAs) are considered as the most suitable heuristic methods for solving multi-objective optimization problems (MOPs). These MOEAs aim to search for a uniformly distributed, near-optimal and near-complete Pareto front for a given MOP. However, MOEAs fail to achieve their aim completely because of their fixed population size. To overcome this limit, a new evolutionary approach to multi-objective optimization was proposed; the dynamic multi-objective evolutionary algorithms (DMOEAs). This work deals with improving the user requirements (i.e., getting a set of optimal solutions in minimum computational time). Although, DMOEA has the distinction of dynamic population size, being an evolutionary algorithm means that it will certainly be characterized by long execution time. One of the main reasons for adapting parallel evolutionary algorithms (PEAs) is to obtain efficient results with an execution time much lower than the one of their sequential counterparts in order to tackle more complex problems. Thus, we propose a parallel version of DMOEA (i.e., PDMOEA). As experimental results, the proposed PDMOEA enhances DMOEA in terms of three criteria: improving the objective space, minimization of computational time and converging to the desired population size.

Key words: *MOEA, DMOEA, PEA, PDMOEA, Objective space, Computational time*

Résumé

Les algorithmes évolutifs d'optimisation multi-objectifs (MOEA) sont les méthodes heuristiques les plus appropriées pour résoudre les problèmes d'optimisation multi-objectifs (MOP). Ces MOEA visent à rechercher un front de Pareto uniformément distribué, quasi optimal et presque complet pour une MOP donnée. Cependant, les MOEA souffrent de ne pas atteindre complètement leur objectif en raison de la taille fixe de leur population. Une nouvelle approche évolutive de l'optimisation multi-objectifs a été proposée ; l'algorithme évolutif multi-objectif dynamique (DMOEA). Ce travail concerne la satisfaction des besoins des utilisateurs (c'est-à-dire l'obtention d'un ensemble de solutions optimales en un temps de calcul minimal). Bien que DMOEA permet la taille dynamique de la population, mais il sera certainement caractérisé par un long temps d'exécution. L'une des principales raisons d'adapter les algorithmes évolutifs parallèles (PEA) est d'obtenir des résultats efficaces avec un temps d'exécution très inférieur à celui de leurs homologues séquentiels afin de traiter des problèmes plus complexes. Nous proposons donc une version parallèle de DMOEA (PDMOEA). En tant que résultats expérimentaux, le PDMOEA proposé améliore DMOEA en fonction de trois critères : améliorer l'espace objectif, minimiser le temps de calcul et converger vers la taille de population souhaitée.

Mots clés : *MOEA, DMOEA, PEA, PDMOEA, Espace objectif, Temps de calcul*

Contents

List of Tables

List of Figures

List of Algorithms

General Introduction

General 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 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 problems is called multi-objective optimization problems (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. However, this aim was difficult to reach by existing MOEAs. Their most important reason to not achieve their aim completely is the fixed population size which makes evolutionary algorithms suffering 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) was proposed in literature the with population size varying dynamically during the runtime of the algorithms using adaptive cell-based rank density estimation.

DMOEA is 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 this DMOEA works to give excellent results compared to other MOEAs, being an evolutionary algorithm means that it will certainly be characterized by long execution time. It can not be ignored that the user is always looking for the best results in less time.

One of the main reasons for using parallel evolutionary algorithms (PEAs) is to obtain efficient results with an execution time much lower than the 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 this dissertation, a parallel version of DMOEA (i.e., PDMOEA) is proposed for the first time in literature. 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 dissertation starts with an introduction which presents the problematic and the project aim. The dissertation is composed of two parts, the first part focuses on the theoretical aspect, the second part shows our contribution in this work. Part I is divided into two chapters. chapter 1 describes the multi-objective optimization evolutionary algorithm used in this work (DMOEA), and defines some related terms. chapter 2 gives the basic concepts of parallel computing organized in three main sections: concepts related to the parallelization, parallel computing, and parallel programming models.

Part II concentrates on the implementation and parallelization of DMOEA with the demonstration of its efficiency. It is composed of 2 chapters (i.e., chapter 3 and chapter 4). chapter 3 describes the analysis and design of our application in three main levels (Global design, DMOEA design, and PDMOEA dsign), and chapter 4 illustrates the implementation of the both algorithms (i.e., DMOEA, PDMOEA) and the implementation of the whole application with the demonstration of DMOEA efficiency using a comparison study between DMOEA and PDMOEA as well as other existing EAs in literature (i.e., NSGA-II and SPEA-II).

The dissertation ends with a conclusion which evaluates the results, and discusses some perspectives.

Part I

State of the art

Chapter 1

Dynamic Multi-objective Optimization Evolutionary Algorithm (DMOEA)

Chapter 1

Dynamic Multi-objective Optimization Evolutionary Algorithm

Introduction

Over time, problems have more than one objective because of their complexity. This kind of problems is called multi-objective optimization problems (MOPs). Multi-objective evolutionary algorithms (MOEAs) are one of the heuristic methods that are intended for solving problems of optimization where these algorithms share the common purpose which is searching for uniformly distributed, near optimal, and well-extended Pareto front for a given MOP. However, this goal is far from being accomplished by existing MOEAs. Most of the existing MOEAs adopt a fixed population size to initiate the evolutionary process, 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 waiting time for fitness improvement.

In this chapter, some terminologies will be presented related to evolutionary algorithms and a new approach to MOPs which is dynamic multi-objective optimization evolutionary algorithms (DMOEA) in particular cell-Based rank and density estimation strategy is used to compute efficiently dominance and diversity information when the population size varies dynamically.

1.1 Optimization problems

Definition [Gan+04]: Given an objective function, an optimization problem looks for an optimal solution (the minimum or maximum) from a large set of possible choices. Formally, in a domain P, and subject to an objective function *f*, there is an optimal solution ($x^* \in P$), where:

- **max *f(x)* :** $f(x) \leq f(x^*)$, $\forall\ x \in P$. Where *f(x*)* is the optimal solution and there is not any other bigger solution.

$\Longrightarrow$ Or

- **min *f(x)* :** $f(x) \geq f(x^*)$, $\forall\ x \in P$. Where *f(x*)* is the optimal solution and there is not any other smaller solution.

1.2 Multi-objective optimization problems

Solutions family of a multi-objective optimization problem is composed of all elements of the search space such that the components of the corresponding objective vectors cannot be all simultaneously improved [YL03].

Definition [Gan+04]: A problem may have more than one objective to be satisfied, and each objective is represented by a function which evaluates the quality of a specific solution. Thus several functions are defined. These functions are usually in conflict with each other.

Using a decision maker, the optimization process of MOPs searches for a solution which compromises the conflicting objectives, the solution is a vector $(x_1, x_2, \dots, x_k)$ that assigns to each objective function a value.

Formally, we can define it as: $\min_{x \in P^n} (f_1(x), ..., f_k(x))$, such that:

- x is a variable in the search space P,
- n is a size of the decision space,
- $f_{i, i \in 1..k}$ is an objective function,
- k is the number of objectives, if $k > 1$ then the problem will be a MOP.

1.3 Evolutionary algorithms

Instead of using classical methods which make the code very long with expensive and difficult maintenance and the search process is very costly and time-consuming, we should use evolutionary algorithms which are one of the heuristic methods that are intended for problems of optimization and problems that have an evolutionary nature.

Definition [Jon98]: EAs are stochastic techniques that simulate the process of natural evolution. EAs are based on biological principles and metaphors. This family of search methods is more appropriate to find approximated or optimal solution for a multi-objective optimization problem (MOP), in a rational time.

In an EA, there is a set of solutions (they are called individuals), this set is named a population. According to the objective functions, the process of EA aims to find the optimal solution or the near-optimal solution among the members of a population. An EA is based on four main steps (representation and fitness evaluation, selection, reproduction, and replacement) [Hus98].

Definition [Alb02]: Evolutionary algorithms (EAs) are techniques for optimization and learning. They are quite a largest of algorithmic families representing bio-inspired systems with a behavior drawn from Nature. Evolution is the basic force driving a population of tentative solutions towards problem regions where the optimal solutions are located.

1.4 Multi-objective optimization evolutionary algorithms

Multi-objective evolutionary algorithms (MOEAs) are used to solve problems requiring optimization of two or more potentially conflicting objectives, without resorting to the reduction of the objectives to a single objective.

During the past decade, several multi-objective evolutionary algorithms (MOEAs) have been proposed and applied in multi-objective optimization problems (MOPs) [Gan+04]. These algorithms share the common purpose which is searching for uniformly distributed, near-optimal, and well-extended Pareto front for a given MOP [YL03]. It needs to approximate a set of non-dominated solutions that produces a uniformly distributed Pareto front. In general, the size of the final Pareto set from most MOEAs is bounded by the size of the initial population. For each of the M conflicting objectives, there exists one different optimal solution.

1.5 Goals of an MOEA

MOEAs aim to finding multiple optimal solutions with a wide range of values for the objectives. In the absence of any such information, all Pareto optimal solutions are equally important. Therefore, there are 2 goals:

- To find a set of solutions as close as possible to the MOP (i.e., convergence)
- To find a set of solutions as diverse as possible (i.e., Diversity)

In addition to these two goals, MOEAs have common purpose which is searching for uniformly distributed, near-optimal, and well-extended Pareto front for a given MOP [YL03] as it is mentioned before section 1.4.

1.6 Dynamic multi-objective optimization evolutionary algorithms

Although, MOEAs common propose is to find a set of optimal solutions by using several strategies, to balance between the diversity of solutions, and the convergence to the Pareto-optimal solutions. this ultimate goal is far from being accomplished by the existing MOEAs. One of the negative aspects that made them move away from their goals, is the fixed population size to initiate the evolutionary process. It is very difficult to solve MOPs by MOEAs with a fixed population size. They have to homogeneously distribute the predetermined computation resource to all the possible directions in the objective space[Mit96]. MOEAs may suffer from premature convergence if the population size is too small or too large. However, undesired computational resources may be incurred and the waiting time for a fitness improvement may be too long in practice [TLK01].

Dynamic multi-objective evolutionary algorithms (DMOEA) is 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 [YL03].

1.7 Adaptive Cell-Based Rank and Density Estimation

During the past decade, fixed population size has great difficulty in obtaining Pareto front which is the desired population because the size of the Pareto front is unknown for most of the MOPs. Meanwhile choosing an excessive initial population size will not be desirable since it may require unnecessarily large computation resources and result in an extremely long-running time [YL03]. Therefore DMOEA is proposed.

1.7.1 Based Rank and Density Calculation Scheme

DMOEA converts the original MOP into a bi-objective optimization problem in algorithm domain [YL03], minimizing rank value and maintaining the density value of each individual as adding or removing individual will affect the rank and density of other individuals knowing that the rank and density values of each individual need to be recalculated after updating the population using grow and decline strategy which will be explained later. For this reason, Cell-Based Rank and Density calculation scheme were introduced.

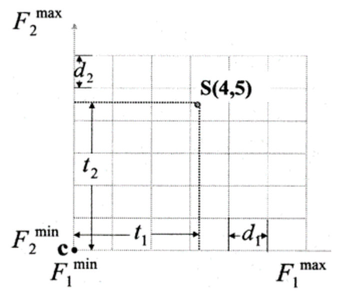


Figure 1.1 – Cell-based rank and density estimation scheme (From [YL03]).

This scheme shown in Figure 1.1 is uses to solve the dynamic movement of population, we can simply consider that the objective space is a grid of m dimensions divided into $k_1 \times k_2 \times \ldots \times k_m$ cells, where every cell is the home of zero, one or more individuals. The cell width of the ith objective dimension can be calculated using equation 1.1.

$$d_i = \frac{\max\limits_{x\epsilon X} f_i(x) - \min\limits_{x\epsilon X} f_i(x)}{k_i}, \qquad i = 1, ..., m \tag{1.1}$$

Where $k_i, i = 1...m$ is the grid scale and x is taken from the whole decision space X,we denote in equation 1.2

$$F_i^{min} = \min_{x\epsilon X} f_i(x), \qquad F_i^{max} = \max_{x\epsilon X} f_i(x) \tag{1.2}$$

As shown in Figure 1.1, point c is denoted as the origin point of the current objective space we can say that c is the cross point of all the lower boundaries of an m-dimensional objective space. The position of c is denoted as $F_1^{min}, F_2^{min}, ..., F_m^{min}$. For a newly generated individual s, whose position is $s_1, s_2, ..., s_m$ in the objective space, the distance between point s and point c will be measured in each dimension in the objective space as $t_1, t_2, ..., t_m$, where equation 1.3 denotes that

$$t_i = s_i - F_i^{min}, \qquad i = 1, ..., m \tag{1.3}$$

Therefore, the home address of individual in the i_{th} dimension is calculated as shown in equation 1.4

$$h_i = div(t_i, d_i) + 1, \qquad i = 1, ..., m \tag{1.4}$$

Therefore, finding the grid location (home address) of a single solution requires only m division operation [YL03].

1.7.2 Rank and Density calculation scheme

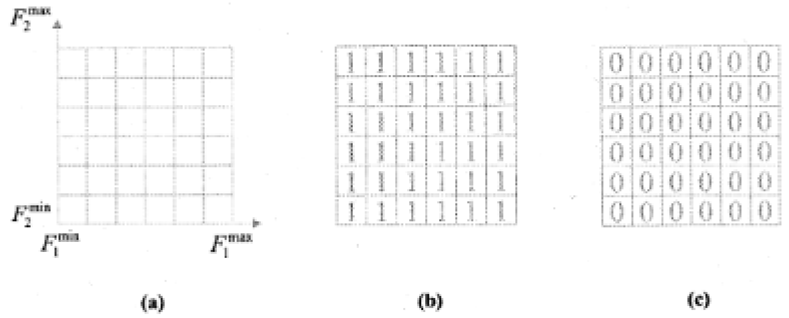


Figure 1.2 – Rank and density calculation scheme (From [YL03]).

1. (a) Estimated objective space and divided cells,
2. (b) Initial rank value matrix of the given objective space,
3. (c) Initial density value matrix of the given objective space.

As shown in Figure 1.2, the center position of each cell is obtained first, and two matrices are set up to store the rank and density values of each cell, which initially having all one and zero, respectively [shown in Figure 1.2[(b)–(c)]. For the purpose of visualization m is chosen to be two. Each individual of the initial population will search for its nearest cell center and identify this cell as its home address and consider the other individuals who share the same home address as its "family members" [YL03].

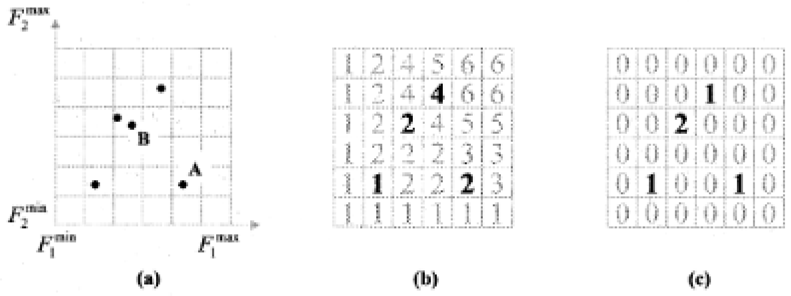


Figure 1.3 – Calculate rank and density (From [YL03]).

1. (a) Initial population and the location of each individual,
2. (b) Rank value matrix of initial population,
3. (c) Density value matrix of initial population.

As shown in Figure 1.3(a)–(c), for each of these "homes" the number of "family members" who dwell in it will be counted and saved as the density value of the "home" In addition, the rank values of the cells that are dominated by any of these "homes" will be increased by the density values of those "homes".

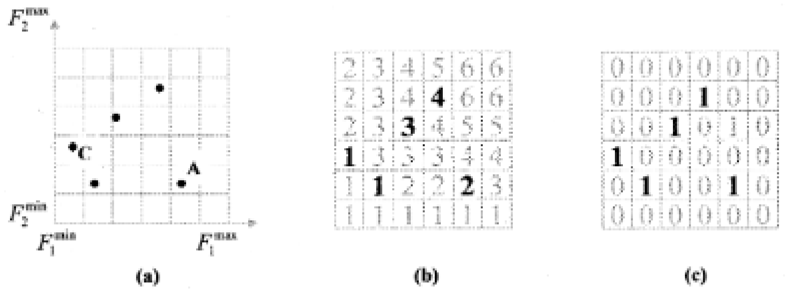


Figure 1.4 – Offspring rank and density (From [YL03]).

1. (a) New population and the location of each individual,
2. (b) Rank value matrix of new population,
3. (c) Density value matrix of new population.

When an offspring is generated and accepted [individual C in Figure 1.4(a)], its "home address" can be located easily by following home address calculation scheme, the density value of its "home" will be increased by one, and the rank values of the cells dominated by its "home" will be increased by one [YL03]. Same thing for an old individual [individual B Figure 1.4.(a)] is removed, its "home" will be notified, and the density value of its "home" will be decreased by one and the rank values of the cells dominated by its "home" will be decreased by one, correspondingly. For instance, as shown in Figure 1.4, the "home address", rank, and density values of individual A are (5,2), 2 and 1, respectively, same thing for individual c it's "home address", rank and density values are (1,3), 1 and 1 respectively. An individual's "home address" will never change if this individual is not removed [YL03].

The fitness evaluation of whether the resulting offspring is good or not is based on its location in the rank and density matrices. By this method, the procedure of updating rank and density matrices is irrelevant to the procedure of a fitness evaluation on an individual. As each crossover or mutation operation can only produce at most two new individuals, the computational load on updating the rank and density matrices will be trivial for each generation [YL03]. The pseudo codes of calculating the cell rank and density values for initial population, determining the home address for an individual, and updating rank and density matrices are illustrated in algorithms 1-3.

Algorithm 1 Pseudocode of calculating CBRDE initial scheme (From [YL03])

```
function $Initial\_cell\_based\_rank\_density\_estimation()$
    Generate initial population $P = [P(1), P(2), ..., P(P_0)]$ with size $P_0$,origin point
    $c = (F_1^{\min}, F_2^{\min}, ..., F_m^{\min})$of objective space($m - dimension$),cell width
    $d_i = \frac{\max_{x \epsilon X} f_i(x) - \min_{x \epsilon X} f_i(x)}{k_i}, (i = 1, ..., m)$ and rank matrix m_r= $1_{k_1 \times k_2 \times ... \times k_m}$ and
    density matrix m_d= $0_{k_1 \times k_2 \times ... \times k_m}$
    for $i = 1 : P_0$ do
        /*Calculate home address $[h_1(i), h_2(i), ..., h_m(i)]$of individual $i$
        $[h_1(i), h_2(i), ..., h_m(i)] = cal\_home\_add(P(i), e)$
        /*Update rank and density matrices
        $sign = 1;$
        $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), ..., h_m(i), sign, m\_d, m\_r)$
```

Algorithm 2 Pseudocode of calculating home address (From [YL03])

```
function $[h_1(i), h_2(i), ..., h_m(i)] = cal\_home\_add(P(i), e)$
    for $j = 1 : m$ do
        /*Calculate the distance between the individual and the original point in the
        $j$th dimension
        $t_j = p_j - F_j^{\min}$;
        /*locate the home address of $p$ on the $j$th dimension
        $h_j = \mod(t_j, d_j) + 1$;
```

Algorithm 3 Pseudocode of updating rank and density matrices (From [YL03])

```
function $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), .., h_m(i)], sign, m\_d, m\_r)$
    for $j = 1 : m$ do
        /*Update rank density matrices
        $m\_d(h_1, h_2, ..., h_m) = m\_d(h_1, h_2, ..., h_m) + sign \times 1$
        $m\_r([h_1..k_1], [h_2..k_2], ., [h_m..k_m]) = m\_r([h_1..k_1], [h_2..k_2], .., [h_m..k_m]) + sign \times 1$
```

1.7.3 Cell Rank and Health Indicator

The health indicator is associated with cell rank. In DMOEA, we assume the rank value of a cell to derive a health indicator in order to measure the dominance relationship of the concerned cell comparing to the others. Assume at generation n, a cell c has a rank value $rank(c, n)$ [YL03]. The health value of cell c at generation n is given as equation 1.5.

$$H(c, n) = \frac{1}{rank(c, n)} \tag{1.5}$$

1.7.4 Cell Density and Crowdedness Indicator

Using a given population size per unit volume, ppv, the optimal population size can be obtained if an MOEA can correctly discover all the trade-off hyper-areas for an MOP. In DMOEA,instead of estimating the trade-off hyper-areas $A_{to}(n)$ at each generation [YL03]. In DMOEA, the optimal population size and final Pareto front will be found simultaneously at the final generation [YL03].
The density value of a cell is defined as the number of the individuals located in it or the number of all "family members" situated on same cell. A crowdedness is associated with each cell to show current density information of the concerned cell [YL03]. Assume at generation n, the density value of cell c is $density(c, n)$, The crowdedness value of cell c is calculated as equation 1.6.

$$D(c, n) = \begin{cases} \frac{density(c,n)}{ppv}, & \text{if } density(c, n) > ppv \\ 1, & \text{otherwise} \end{cases} \tag{1.6}$$

1.7.5 Population Growing Strategy

Because exploring the cells with minimum rank values and maintaining these cells densities to a desired value are two converted objectives of DMOEA, crossover and mutation operations need to be devised to fit both of the purposes [YL03]. After selecting a fixed number of parents in a selection pool, DMOEA uses diffusion scheme in the objective space [YL03]. Where each individual shares its information with the leading individuals in order to locate its moving direction. If a resulting offspring is located in a cell with a better fitness (a lower or equal rank value, or a lower or equal density value) than its selected parents, it will be kept to the next generation. Otherwise, it will not survive. It is not clear to the authors yet if this "diffusion scheme" is more effective than the other selection operations

[YL03]. This strategy will guarantee that a newborn individual will have a better fitness value than at least one of its parents. DMOEA encourages an offspring to land in a sparse area. Some offspring will tend to move toward a direction opposite to the true Pareto front where the cells close to the Pareto front are crowded. Obviously, these movements hinder the population in converging to the Pareto front [YL03]. A forbidden region concept is proposed in the offspring-generating scheme for the density sub-population. Where forbidden region includes all cells strongly dominated by the selected parent.

Algorithm 4 Pseudocode of forbidden region (From [YL03])

```
function $F\_r = forbidden\_region([p\_h_1, p\_h_2, .., p\_h_m], [o\_h_1, o\_h_2, .., o\_h_m])$
    F_r=1;
    for $j = 1 : m$ do
        /*Calculate distance between an offspring and its parent in the $j$th dimension
        $z_j = o\_h_j - p\_h;$
        if $z_j \prec= 0$ then
            F_r=0
    end
```

As shown in algorithm 4, the offspring will produce $F_r = 1$ if it is located in the forbidden region, else $F_r = 0$. Therefore, an offspring located in the forbidden region will have higher value of home addresses in each dimension and will not survive in the next generation [YL03]. A resulting offspring of the selected parent p is located in the forbidden region will be eliminated even if it is located in a sparse area, because this kind of offspring has the tendency to push the entire population away from the desired evolutionary direction [YL03]. Algorithm 5 shows the pseudo code of population growing strategy.

Algorithm 5 Pseudocode of of population growing strategy (From [YL03])

```
function $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), .., h_m(i)], sign, m\_d, m\_r)$

    /*generate an offspring o via crossover or mutation from parent p, whose home address is $[p\_h_1, p\_h_2, .., p\_h_m]$.
    /*Calculate the home address $[o\_h_1, o\_h_2, .., o\_h_m]$ of an offspring $o$ $[o\_h_1, o\_h_2, .., o\_h_m] = cal\_home\_add(o, c)$
    if (o belong to rank sub-population) and (o has lower or rank equal value to any of its parent) then
        /*keep offspring o in the population and update m_r and m_d
        $sign = 1;$
        $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), ..., h_m(i)], sign, m\_d, m\_r)$
        $N = N + 1;$
    $F\_r = 0$
    if (o belong to rank sub-population) and (o has lower or equal density value transformed to any of its parent) and ($F\_r = 0$) then
        /*keep offspring o in the population and update m_r and m_d
        $sign = 1;$
        $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), ..., h_m(i), sign, m\_d, m\_r)$
        $N = N + 1;$
    for $i = 1 : P_0$ do
        /*Update the rank and density values rank(i) and density(i) of individual i
        $rank(i) = m\_r([o\_h_1, o\_h_2, .., o\_h_m])$
        $density(i) = m\_d([o\_h_1, o\_h_2, .., o\_h_m])$
```

1.7.6 Population Declining Strategy

A population declining strategy is necessary to prevent the population size from growing without limits. An individual will be removed or not depends on its *health*, *crowdedness* and *age* indicators [YL03]. An individual in the initial population, its initial *age* is one and it will be increased by one in each generation if it will be survive. the *age* of a newborn offspring is one and grows generation by generation as long as it is alive [YL03]. To calculate the *age* indicator. At generation n, an individual y has an $age(y, n)$. Its *age* indicator calculated as shown in 1.7.

$$A(y,n) = \begin{cases} \frac{age(c,n)-A_{th}}{n}, & \text{if } age(y,n) > A_{th} \\ 0, & \text{otherwise} \end{cases} \tag{1.7}$$

Where A_{th} is a specified *age* threshold. To ensure that an eliminated individual has a low fitness value, DMOEA periodically removes three types of individuals with different likelihoods [YL03].

1/ Likelihood of Removing the Most Unhealthy Individuals,

2/ Likelihood of Removing the Unhealthy Individuals in the Most Crowded Cells,

3/ Likelihood of Removing Non-dominated Individuals From the Most Crowded Cells After Convergence.

Likelihood of Removing the Most Unhealthy Individuals

The first likelihood is calculated as it is shown in equation 1.8.

$$l_1^i = (1 - H(c_i, n))^2 \times A(y_i, n) \tag{1.8}$$

Where $H(c, n)$ is the *health* indicator value of the cell c_i which contains individual y_i at generation n. It is calculated as equation 1.9.

$$H(c,n) = frac1rank_max \tag{1.9}$$

Likelihood of Removing the Unhealthy Individuals in the Most Crowded Cells

$$l_2^i = (1 - H(c_i, n))^2 \times \left(1 - \frac{1}{r_{di}}\right)^2 \times (D(c_i, n) - 1) \times A(y_i, n) \tag{1.10}$$

Where $H(c, n)$ is the *health* indicator value of the cell c_i which contains individual y_i at generation n. It is calculated as equation 1.11.

$$H(c,n) = \frac{1}{rank_{d\max}} \tag{1.11}$$

and $D(c_i, n)$ represent the *health* and *crowdedness* values of the cell c_i that contains individual y_i at generation n and $R_{dr} = r_{di}$ represents the local rank value of the individuals of set Y_{dr}.

Likelihood of Removing Non-dominated Individuals From the Most Crowded Cells After Convergence

$$l_3^i = (D(c_i, n) - 1) \times \left(1 - \frac{1}{r_{di}}\right)^2 \times A(y_i, n) \tag{1.12}$$

where $D(c_i, n)$ represents the *crowdedness* value of the cell that contains individual y_i in generation n.
To determine whether an individual y_i will be eliminated, three random numbers between [0, 1] are generated to compare with the concerned likelihoods l_1^i,l_2^i and l_3^i, according to the situation of the given individual. If the likelihood is greater than the corresponding random

number, the selected individual will be removed from the population. Otherwise, the selected individual will survive to the next generation [YL03]. Algorithm 6 shows the pseudo code of Population Declining Strategy.

Algorithm 6 Pseudocode of of population decline strategy (From [YL03])

```
function $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), .., h_m(i)], sign, m\_d, m\_r)$

    $R = \{\}$;
    $q = 0$;
    for i=1:N do
        /*generate three random numbers between 0 and 1
        $rand_1 = rand[0, 1]$
        $rand_2 = rand[0, 1]$
        $rand_3 = rand[0, 1]$
        /*Evaluate if individual $P(i)$ will be removed
        if $l_1^i > rand_1$ or $l_2^i > rand_2$ or $l_3^i > rand_3$ then
            $R = [R \quad P(i)]$;
            $q = q + 1$;
    Find the set of individuals $R = r_1, r_2, .., r_q$that are eligible to be removed from population $P$
    for $i = 1 : q$ do
        /*obtain home $[h_1(r_i), h_2(r_i), ..., h_m(r_i)]$ of individual
        $[h_1(r_i), h_2(r_i), ..., h_m(r_i)] = cal\_home\_add(r_i, c)$
        $N = N - 1$;
        $sign = -1$;
        /*Update the rank and density matrices
        $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), ..., h_m(i), sign, m\_d, m\_r)$
        $N = N + 1$;
    for $i = 1 : N$ do
        /*Update the rank and density values rank(i) and density(i) of individual i
        $rank(i) = m\_r([o\_h_1, o\_h_2, .., o\_h_m])$
        $density(i) = m\_d([o\_h_1, o\_h_2, .., o\_h_m])$
```

1.7.7 Objective Space Compression Strategy

In DMOEA the boundaries of the objective are usually selected to be very large, which may be far away from the true Pareto front, to ensure that the entire true Pareto front is covered by the estimated objective space. Figure 1.5, shows the rank values for both cells A and B is 1 since they contain the true Pareto front, all the resulting individuals located in cells A and B are non-dominated solutions according to the proposed cell-based rank calculation scheme. However, if we examine these individuals by using the pure Pareto ranking method, we will find that most of these individuals are dominated points [YL03]. An objective space compression strategy is designed to adjust the size of the objective space and to make it suitable to search for the true Pareto front with a high precision.

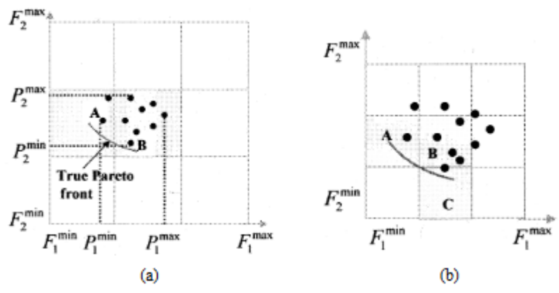


Figure 1.5 – Compression strategy (From [YL03])

1. (a) Objective space and resulting Pareto front before compression,
2. (b) Compressed objective space and newly explored non-dominated cell.

A refined upper boundary of the objective space is calculated by equation 1.13.

$$F_i^{\max} = (P_i^{\max} - F_i^{\max})/2 \tag{1.13}$$

Which means the distance between the updated upper boundary of the objective space and the upper boundary of the current population has decreased to half of its original value. Similarly, a refined lower boundary of the objective space is calculated by equation 1.14.

$$F_i^{\min} = (P_i^{\min} - F_i^{\min})/2 \tag{1.14}$$

Which means the distance between the updated lower boundary of the objective space and the lower boundary of the current population has decreased to half of its original value. Comparing Figure 1.5(a) with (b), we can see that the proposed objective space compression strategy results in three effects. Algorithm 7 shows the pseudo code of objective space compression strategy.

Algorithm 7 Pseudocode of objective space compression strategy (From [YL03]).

```
function $objective\_space\_compression()$

    /*Objective space compression strategy
    if $(max\_rank(P) = 1)$ and $(min\_age(P) > A_{th})$ then
        for $i = 1 : m$ do
            if $(F_i^{\max} - P_i^{\max}) > 0.1(F_i^{\max} - F_i^{\min})$ then
                /*Update the upper boundary of the objective space
                $F_i^{\max} = (P_i^{\max} - F_i^{\max})/2$
    Recalculate origin point of objective space $c$,cell width $d_i$;
    Initialize rank matrix m_r= $1_{k_1 \times k_2 \times \ldots \times k_m}$ and density matrix m_d= $0_{k_1 \times k_2 \times \ldots \times k_m}$
    for $i = 1 : N$ do
        /*Calculate home address $[h_1(i), h_2(i), ..., h_m(i)]$of individual $i$
        $[h_1(i), h_2(i), ..., h_m(i)] = cal\_home\_add(P(i), e)$
        /*Update rank and density matrices
        $sign = 1$;
        $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), ..., h_m(i), sign, m\_d, m\_r)$
```

1.7.8 Main loop

Taking into consideration all techniques introduced from Cell-Based rank density estimation algorithm, we can summaries DMOEA in steps and Figure 1.6 shows the relay of theses steps.

1/ Cell-Based Rank and Density Calculation Scheme,

2/ Population Growing Strategy,

3/ Population Declining Strategy,

4/ Objective Space Compression Strategy.

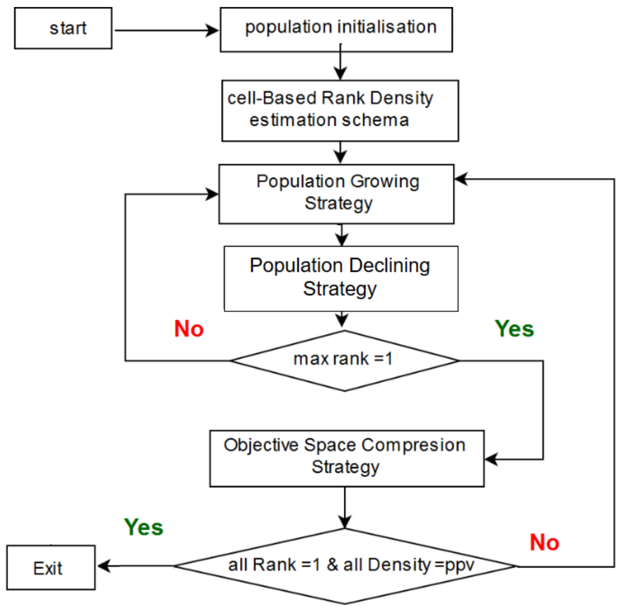

start
population initialisation
cell-Based Rank Density estimation schema
Population Growing Strategy
Population Declining Strategy
No
Yes
max rank =1
Objective Space Compresion Strategy
Yes
No
Exit
all Rank =1 & all Density =ppv

Figure 1.6 – DMOEA algorithm.

DMOEA should stop by examining the 3 stopping criteria [YL03]

1/ The rank values of all cells are ones,

2/ The objective space cannot be compressed anymore,

3/ Each resulting non-dominated cell contains *ppv* individuals .

The reason of this step is because another criterion "the Pareto ranks of all resulting individuals are equal to one should be satisfied as well to guarantee there is no dominance relationship among the resulting Pareto solutions at the final generation [YL03].

Conclusion

In this chapter, we have presented some definitions related to multi-objective optimization evolutionary algorithms (MOEA) and also its limit which is the fixed population size. This limit specified the appearance of dynamic multi-objective optimization evolutionary algorithms (DMOEA). DMOEA aims to search for a uniformly distributed, near-optimal, and well-extended Pareto front for a given MOP. DMOEA may suffer from a long computational time which makes our research tend to parallelize this algorithm, where the parallel computing is the essence of the next chapter.

Chapter 2

Parallel Computing

Introduction

Evolutionary algorithms (EAs) are stochastic search methods which have been applied successfully in many searches, optimization and machine learning problems [AT02]. Evolutionary Algorithms (EAs) have developed into a feasible optimization approach for many industrial applications. Especially, Multi-Objective EAs (MOEAs) receive a lot of attention, because many practical optimization problems are often multi-objective [SUZ05]. DMOEA was proposed to be better than MOEA such that the population size changes dynamically during the runtime of the algorithm, until it reaches the desired population size, a uniformly distributed Pareto front, and a near optimal and well extended Pareto front for the given problem. There are two kinds of limitations on fitness evaluations that make MOEAs and DMOEA infeasible for many real-world applications. First, a fitness evaluation requires real-world experiments, which can be both costly and time-consuming. Second, the fitness evaluation is computationally too expensive to allow optimization through EAs, MOEAs and also DMOEA in reasonable time [SUZ05]. The goal of this chapter is to bring uniformity and structure to the different research issues concerning parallel EA in general and DMOEA in especial.

2.1 Motivations for Parallel Computing

Development of parallel software requires intensive time and effort. Which causes inherent complexity of specifying and coordinating concurrent tasks [Kum02]. There are a lot of arguments in favor of parallel computing platforms. Before all, if it takes two years to develop a parallel application, during which time the underlying hardware and/or software platform has become obsolete, the development effort is clearly wasted. At the same time, standardized hardware interfaces have reduced the execution time from the development of a microprocessor to a parallel machine based on the microprocessor. Furthermore, considerable progress has been made in the standardization of programming environments to ensure a longer life-cycle for parallel applications [Kum02]. In addition to these arguments, computing power needs to remain an important driver in the evolution of computer technologies. In the scientific field, for example, applications exhaust computing power even on some of the most recent machines. Today, much of this power comes from the use of parallel machines. Nevertheless, getting the best performance from these machines remains difficult and usually depends on several factors. The advantage of parallel architectures lies in the performance of the applications they achieve.

2.2 Scope of Parallel Computing

Parallel computing has influenced a variety of areas ranging from computational simulations for scientific and engineering applications to commercial applications in data mining and transaction processing. The cost benefits of parallelism coupled with the performance requirements of applications present compelling arguments in favor of parallel computing [Kum02] these which we presents in section 2.1. We present one of several applications of parallel computing which is the most important for us it is applications in Engineering and Design.

Applications in Engineering and Design focus on optimization of a variety of processes. Parallel computers are used to solve a variety of discrete and continuous optimization problems and algorithms such as Simplex. Genetic programming for discrete optimization have been efficiently parallelized and are frequently used [Kum02].

2.3 Concepts and Terminology

This is the most important terminologies in parallel computing.

2.3.1 Parallel computing

The old vision of parallel computing is that parallel computing is more than just a strategy for achieving high performance, but it is a compelling vision for how computation can seamlessly scale from a single processor to virtually limitless computing power [Don+03]. Now, parallel computing deals with hardware and software for computation in which many calculations are carried out simultaneously. The main goal of parallel computing is the improvement in calculating capacity [KGG17]. In the simplest sense, parallel computing is the simultaneous use of multiple computes resources to solve the computational problems. To be run using multiple CPUs. A problem is broken into discrete parts that can be solved concurrently, where each part is further broken down to a series of instructions, each instructions from each part execute simultaneously on several CPUs as Figure 2.1 shows.

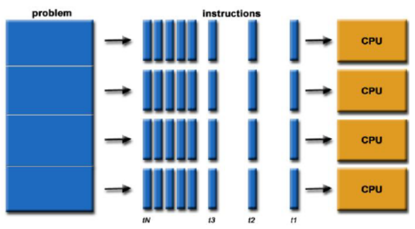


Figure 2.1 – Parallel Computing (From [KGG17]).

2.3.2 Parallel Computers

Parallel computers provide great amounts of computing power, but they do so at the cost of increased difficulty in programming and using them. Certainly, a uni-processor that was fast enough would be simpler to use [Don+03]. Virtually all stand-alone computers today are parallel from a hardware perspective:

- Multiple functional units (floating point, integer, GPU, etc.),
- Multiple execution units / cores,

- Multiple hardware threads.

Networks connect multiple stand-alone computers (nodes) to create larger parallel computer clusters, where each computer node is a multi-processor parallel computer in itself, Multiple compute nodes are networked together with an infinite-Band network and special purpose nodes, also multi-processor.

2.3.3 Von Neumann Computer Architecture

John Von Neumann (December 28, 1903 – February 8, 1957) was a Hungarian-American mathematician, physicist, and computer scientist. He made major contributions to a number of fields, including mathematics (foundations of mathematics, functional analysis, representation theory, operator algebras, geometry, topology, and numerical analysis...) [Von93]. John von Neumann who first authored the general requirements for an electronic computer in his 1945 papers, it is also known as "stored-program computer" - both program instructions and data are kept in electronic memory. Differs from earlier computers which were programmed through "hard-wiring" [18a]. Since the appearance of this architecture, virtually all computers have followed this basic design. Where its basic components comprised of four main components: Memory, Control Unit, Arithmetic Logic Unit, and Input/Output.

2.3.4 Instruction Stream and Data Stream

The term 'stream' refers to a sequence or flow of either instructions or data operated on by the computer. In the complete cycle of instruction execution, a flow of instructions from main memory to the CPU is established. This flow of instructions is called an instruction stream. In the same time, there is a flow of operands between processor and memory bidirectionally [18b]. This flow of operands is called a data stream.

2.3.5 Types of classification

Parallel computers are those that confirm the parallel processing between the operations in some way. After all the basic terms of parallel processing and computation have been defined. Parallel computers can be characterized based on the data and instruction streams forming various types of computer organizations. They can also be classified based on the computer structure, e.g. multiple processors having separate memory or one shared global memory. Parallel processing levels can also be defined based on the size of instructions in a program called grain size. Thus, parallel computers can be classified based on various criteria [18b].

The classification of parallel computers have been identified as Classification based on the instruction and data streams, classification based on the structure of computers, classification based on how the memory is accessed, and classification based on grain size.

2.3.6 Flynn's Classification

Michael Flynn proposed this classification as the first study in 1972. Flynn did not consider the machine architecture for classification of parallel computers; he introduced the concept of instruction and data streams for categorizing of computers [18b]. All the computers classified by Flynn are not parallel computers, but to understand the concept of parallel computers, it is necessary to understand all types of Flynn's classification. Since, this classification is based on instruction and data streams.
Flynn's classification is based on a multiplicity of instruction streams and data streams

observed by the CPU during program execution. Let Instruction Stream (Is) and Data Stream (Ds) are a minimum number of streams flowing at any point in the execution. The computer organization can be categorized as follows:

Single Instruction and Single Data stream (SISD)

In this organization, the sequential execution of instructions is performed by one CPU containing a single processing element (PE). Therefore, SISD machines are conventional serial computers that process only one stream of instructions and one stream of data.

Single Instruction and Multiple Data stream (SIMD)

In this organization, multiple processing elements work under the control of a single control unit. It has one instruction and multiple data stream. All the processing elements of this organization receive the same instruction broadcast from the CU. Main memory can also be divided into modules for generating multiple data streams acting as a distributed memory. Therefore, all the processing elements simultaneously execute the same instruction and are said to be 'lock-stepped' together. Each processor takes the data from its own memory and hence it has on distinct data streams. (Some systems also provide a shared global memory for communications.) Every processor must be allowed to complete its instruction before the next instruction is taken for execution. Thus, the execution of instructions is synchronous.

Multiple Instruction and Single Data stream (MISD)

In this organization, multiple processing elements are organized under the control of multiple control units. Where each control unit is handling one instruction stream and processed through its corresponding processing element. But each processing element is processing only a single data stream at a time. Therefore, for handling multiple instruction streams and single data stream, multiple control units and multiple processing elements are organized in this classification. All processing elements are interacting with the common shared memory for the organization of single data stream [18b].
This classification can be very helpful for specialized applications. But MISD organization is not popular in commercial machines as the concept of single data streams executing on multiple processors is rarely applied [18b]. For example, Real-time computers need to be faulty where several processors execute the same data for producing the redundant data. All these redundant data are compared as results which should be same; otherwise faulty unit is replaced. Thus MISD machines can be applied to fault tolerant real time computers.

Multiple Instruction and Multiple Data stream (MIMD)

In this organization, multiple processing elements and multiple control units are organized as in MISD. But the difference is that now in this organization multiple instruction streams operate on multiple data streams. Therefore, for handling multiple instruction streams, multiple control units and multiple processing elements are organized such that multiple processing elements are handling multiple data streams from the Main memory. The processors work on their own data with their own instructions. Tasks executed by different processors can start or finish at different times. They are not lock-stepped, as in SIMD computers, but run asynchronously. This classification actually recognizes the parallel computer. That means in the real sense MIMD organization is said to be a Parallel computer. All multiprocessor systems fall under this classification [18b].

2.3.7 Limits and Costs of Parallel Programming

Suppose we wish to compare a parallel and sequential computer built from the same units, to argue that a new parallel algorithm is many times faster than the best sequential algorithm (the same reasoning applies to logic gates on an integrated circuit). Given N parallel units and an algorithm that run times faster on sufficiently large inputs, one can simulate the parallel system on the sequential system by dividing its time between computational slices. Since this simulation is roughly N times slower, it runs M/N times faster than the original sequential algorithm. If this original sequential algorithm was the fastest possible, we have $M \leq N$. In other words, a fair comparison should not demonstrate a parallel speedup that exceeds the number of processors. a super linear speedup can indicate an inferior sequential algorithm or the availability of a larger amount of memory to N processors. The bound is reasonably tight in practice for small N and can be violated slightly because N CPUs include more CPU cache, but such violations alone do not justify parallel algorithms [Mar14].

2.4 Parallel Computer Memory Architectures

It has different types.

2.4.1 Shared Memory

General Characteristics:

Shared memory are parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space. Multiple processors can operate independently but share the same memory resources. Also Changes in a memory location effected by one processor are visible to all other processors. Historically, shared memory machines have been classified as UMA and NUMA, based on upon memory access times [18a].

Uniform Memory Access (UMA):

Most commonly represented today by Symmetric Multiprocessor (SMP) machines, they have Identical processors and equal access and access times to memory. Sometimes called CC-UMA - Cache Coherent UMA. Cache coherent means if one processor updates a location in shared memory, all the other processors know about the update. Cache coherency is accomplished at the hardware level [18a].

Non-Uniform Memory Access (NUMA):

Often made by physically linking two or more SMPs , where one SMP can directly access memory of another SMP. Not all processors have equal access time to all memories. Memory access across link is slower and cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA [18a].

Advantages:

There are two advantages. The first is that the global address space provides a user-friendly programming perspective to memory. Second, data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs.

Disadvantages:

Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management. Then, programmer responsibility for synchronization constructs that ensure ”correct” access of global memory.

2.4.2 Distributed Memory

General Characteristics:

Like shared memory systems, distributed memory systems vary widely but share a common characteristic. Distributed memory systems require a communication network to connect inter-processor memory.Where Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors. Because each processor has its own local memory, it operates independently. Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of cache coherency does not apply. When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer’s responsibility. The network ”fabric” used for data transfer varies widely, though it can be as simple as Ethernet [18a].

Advantages:

Memory is scalable with the number of processors. Increase the number of processors and the size of memory increases proportionately. Then, each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain global cache coherency. Finally, cost effectiveness: can use commodity, off-the-shelf processors and networking.

Disadvantages:

First, the programmer is responsible for many of the details associated with data communication between processors. Second, it may be difficult to map existing data structures, based on global memory, to this memory organization. Then non-uniform memory access times data residing on a remote node takes longer to access than node local data.

2.4.3 Hybrid Distributed-Shared Memory

The largest and fastest computers in the world today employ both shared and distributed memory architectures. Shared memory component can be a shared memory machine and/or GPU. Where Processors on a compute node share same memory space and Requires communication to exchange data between compute nodes.

Advantages and Disadvantages:

Whatever is common to both shared and distributed memory architectures. Then, Increased scalability is an important advantage. Finally, Increased programming complexity is a major disadvantage.

2.5 Parallel Programming Models

Parallel Programming Models exist as an abstraction above hardware and memory architectures.he standard parallel architectures support a variety of decomposition strategies, such as decomposition by task (task parallelism) and decomposition by data (data parallelism). Our introductory treatment will concentrate on data parallelism be-cause it represents the most common strategy for scientific programs on parallel machines. In data parallelism, the application is decomposed by subdividing the data space over which it operates and assigning different processors to the work as associated with different data subspace. Typically this strategy involves some data sharing at the boundaries, and the programmer is responsible for ensuring that this data sharing is handled correctly is, data computed by one processor and used by another are correctly synchronized.Once a specific decomposition strategy is chosen, it must be implemented. Here,the programmer must choose the programming model to use [Don+03]. The two most common models are The shared-memory model, in which it is assumed that all data structures areal located in a common space that is accessible from every processor, and the message-passing model, in which each processor (or process) is assumed to have its own private data space, and data must be explicitly moved between spaces as needed.
In the message-passing model, data structures are distributed across the processor memories; if a processor needs to use a data item that is not stored locally, the processor that owns that data item must explicitly "send" it to the requesting processor. The latter must execute an explicit "receive" operation, which is synchronized with the send, before it can use the communicated data item.

2.5.1 Shared Memory Model

In this programming model, processes/tasks share a common address space, which they read and write to asynchronously. Various mechanisms such as locks / semaphores are used to control access to the shared memory, resolve contentions and to prevent race conditions and deadlocks. This is perhaps the simplest parallel programming model. An advantage of this model from the programmer's point of view is that the notion of data "ownership" is lacking, so there is no need to specify explicitly the communication of data between tasks. All processes see and have equal access to shared memory. Program development can often be simplified with an important disadvantage in terms of performance is that it becomes more difficult to understand and manage data locality.

2.5.2 Threads Model

This programming model is a type of shared memory programming.In the threads model of parallel programming, a single "heavy weight" process can have multiple "light weight", concurrent execution paths. For applications where the workload depends on application input that can vary widely, delay the decision about the number of threads to employ until run-time when the input sizes can be examined. Examples of workload input parameters that affect the thread count include things like matrix size, database size, image/video size and resolution, depth/breadth/bushiness of tree based structures, and size of list based structures. Similarly, for applications designed to run on systems where the processor count can vary widely, defer the number of threads to employ decision till application run-time when the machine size can be examined.

For applications where the amount of work is unpredictable from the input data, consider using a calibration step to understand the workload and system characteristics to aid in choosing an appropriate number of threads. If the calibration step is expensive, the calibration results can be made persistent by storing the results in a permanent place like the file system. Avoid creating more threads than the number of processors on the system, when all the threads can be active simultaneously; this situation causes the operating system to multiplex the processors and typically yields sub-optimal performance.

When developing a library as opposed to an entire application, provide a mechanism whereby the user of the library can conveniently select the number of threads used by the library, because it is possible that the user has higher-level parallelism that renders the parallelism in the library unnecessary or even disruptive.

Finally, for OpenMP, use the *num_threads* clause on parallel regions to control the number of threads employed and use the if clause on parallel regions to decide whether to employ multiple threads at all. The *omp_set_num_threads* function can also be used but it is not recommended except in specialized well-understood situations because its affect is global and persists even after the current function ends, possibly affecting parents in the call tree. The *num_threads* clause is local in its effect and so does not impact the calling environment [Don+03].

2.5.3 Distributed Memory / Message Passing Model

The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum, which has over 40 participating organizations, including vendors, researchers, software library developers, and users. The goal of the Message Passing Interface is to establish a portable, efficient, and flexible standard for message passing that will be widely used for writing message passing programs. As such, MPI is the first standardized, vendor independent, message passing library. The advantages of developing message passing software using MPI closely match the design goals of portability, efficiency, and flexibility. MPI is not an IEEE or ISO standard, but has in fact, become the "industry standard" for writing message passing programs on HPC platforms [18a]. Message passing is an approach that makes the exchange of data cooperative. Data must both be explicitly sent and received. An advantage is that any change in the receiver's memory is made with the receiver's participation [Gro09]. One-sided operations between parallel processes include remote memory reads and write. An advantage is that data can be accessed without waiting for another process [Gro09]. MPI is a specification for the developers and users of message passing libraries. By itself, it is NOT a library but rather the specification of what such a library should be. MPI primarily addresses the message passing parallel programming model: data is moved from the address space of one process to that of another process through cooperative operations on each process. Simply stated, the goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs. The interface attempts to be Practical, Portable, Efficient and Flexible [18a]

2.5.4 Data Parallel Model

May also be referred to as the Partitioned Global Address Space (PGAS) model. The data parallel model demonstrates the characteristics which are :

- Address space is treated globally,
- Most of the parallel work focuses on performing operations on a data set. The data set is typically organized into a common structure, such as an array or cube,
- A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure,
- Tasks perform the same operation on their partition of work, for example, "add 4 to every array element".

On shared memory architectures, all tasks may have access to the data structure through global memory.On distributed memory architectures, the global data structure can be split up logically and/or physically across tasks [18a].

2.5.5 Hybrid Model

A hybrid model combines more than one of the previously described programming models. Currently, a common example of a hybrid model is the combination of the message passing model (MPI) with the threads model (OpenMP). Threads perform computationally intensive kernels using local, on-node data and Communications between processes on different nodes occurs over the network using MPI. This hybrid model lends itself well to the most popular hardware environment of clustered multi/many-core machines. Another similar and increasingly popular example of a hybrid model is using MPI with CPU-GPU (Graphics Processing Unit) programming.
MPI tasks run on CPUs using local memory and communicating with each other over a network, computationally intensive kernels are off-loaded to GPUs on-nod. A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure.Data exchange between node-local memory and GPUs uses CUDA (or something equivalent).

2.5.6 SPMD and MPMD

Single Program Multiple Data (SPMD):

SPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models. SINGLE PROGRAM: All tasks execute their copy of the same program simultaneously. This program can be threads, message passing, data parallel or hybrid. SPMD programs usually have the necessary logic programmed into them to allow different tasks to branch or conditionally execute only those parts of the program they are designed to execute. That is, tasks do not necessarily have to execute the entire program perhaps only a portion of it. The SPMD model, using message passing or hybrid programming, is probably the most commonly used parallel programming model for multi-node clusters [18a].

Multiple Program Multiple Data (MPMD):

Like SPMD, MPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models.MULTIPLE PROGRAM: Tasks may execute different programs simultaneously. The programs can be threads, message passing, data parallel or hybrid. MULTIPLE DATA: All tasks may use different data. MPMD applications are not as common as SPMD applications, but may be better suited for certain types of problems, particularly those that lend themselves better to functional decomposition than domain decomposition [18a].

2.6 Literature review

In paper [GVR17], they address the discovery of repeated common patterns as a multi-objective optimization problem by means of a hybrid MPI/OpenMP approach which parallelizes a well-known multi-objective meta heuristic, the fast non-dominated sorting genetic algorithm (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 OpenMP to parallelize a modified version of NSGA-II when finding common patterns on a set of amino acid sequences. This new modified algorithm, named H-NSGAII, combines the evolutionary properties of the well-known NSGA-II with a local search function specialized in improving the quality of the predictions made. The complexityof the addressed optimization problem (NP-hard) has motivated this research, mainly designed to take advantage of current hardware architectures (SMP clusters with many cores).

In paper [SV14], two parallel multi-objective meta-heuristics, NSGA-II and SPEA2, have been applied to tackle the inference of phylogenetic trees considering two criteria: maximum parsimony and maximum likelihood. These algorithms were implemented with OpenMP 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 sand phylogenetic results by means of well-known parallel and multi-objective metrics.

In paper [DPM14], they are presented a fine-grained parallelization of the Fast Non-dominating Sorting Genetic Algorithm (NSGA-II) for the CUDA architecture. In particular, they discussed how this solution can be exploited to solve multi-objective optimization task in the field of computational and systems biology.

Most of well-known multi-objective optimization algorithms (MOEAs) have been parallelized using different parallel technologies for improving their performance. Comparing to state-of-the-art MOEAs, DMOEA is found to be competitive in terms of maintaining diversity and improving the objective space [YL03]. However, DMOEA suffers from long computational time. In this work, we propose a parallel version of DMOEA (i.e., PDMOEA) to enhance the sequential DMOEA in terms of three criteria (i.e., improving the objective space, minimization of computational time and converging to the desired population size). For implementing PDMOEA, we adopted threads model of parallel computing.

Conclusion

To conclude this chapter, we must know that it is not intended to cover Parallel Programming in depth, as this would require significantly more time. This chapter began with a discussion on parallel computing - what it is and how it's used, followed by a discussion on concepts and terminology associated with parallel computing. The topics of parallel memory architectures and programming models are then explored. In the next chapter, we will use one of these techniques to parallelize DMOEA.

Part II

Implementation & parallelization of DMOEA with the demonstration of its efficiency

Chapter 3

Analysis & Design

Chapter 3

Analysis & Design

Introduction

After taking knowledge in the first chapter about evolutionary algorithms and their limits which caused the appearance of a dynamic multi-objective evolutionary algorithm to get the desired population with the advantage of changing population size dynamically until getting the near optimal, well extended Pareto front. This chapter is divided into five sections. The first section is dedicated to describe the problem of this project. The second section is used for explaining the project cycle. The third section is used for create the global design of the application. Next, the two last sections are dedicated to the detailed design of the both versions, where each section contains analysis and design of each version.

3.1 Problem description

This dissertation is not limited to implement an evolutionary or genetic algorithm only and to be satisfied with its results but the main objective is to propose an adapted parallel version (i.e., PDMOEA) and compare between the results of this two implementations regardless of the techniques used to make this algorithm in parallel. This makes us thinking that the main problem addressed by this dissertation is how can we parallelize DMOEA. The problem is rather than a set of several sub-problems, such as how to achieve both versions of the algorithm and how to form an application with the possibility of comparison between their results and also how to choose the multi-objective optimization problem that will be solved by this algorithm. Figure 3.1

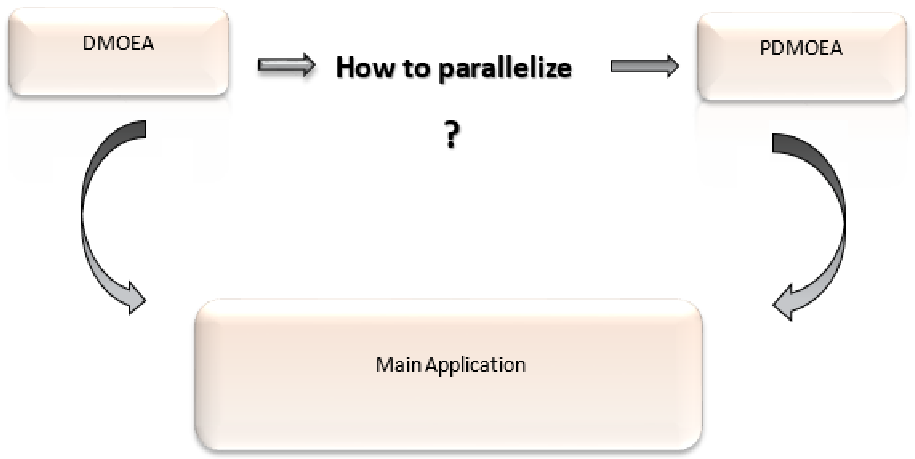


Figure 3.1 – Project Problem.

3.2 project cycle

After describing the problem of this study in the last section, it is clear now that our project will be divided into three parts which are:

- Implement the sequential version of DMOEA,
- Parallelize DMOEA and obtain a parallel version of DMOEA (i.e., PDMOEA),
- Compare results of both versions,
- Compare DMOEA results with other evolutionary algorithms results and Parallel DMOEA (i.e., PDMOEA) with other parallelized EAs (i.e., NSGA-II, SPEA-II) .

All these steps are described in Figure 3.2.

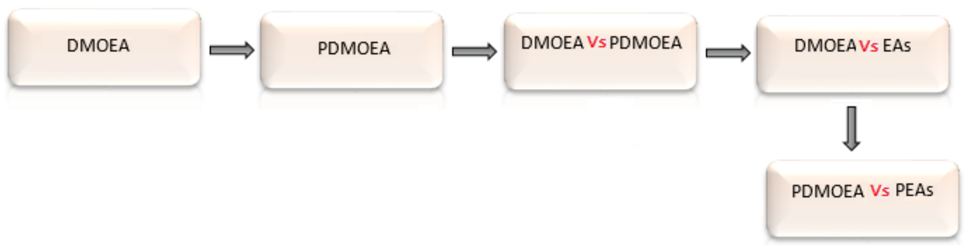


Figure 3.2 – Project Cycle.

3.3 Global Design of the whole Application

Figure 3.3 shows the structure of the whole application with its inputs and outputs.

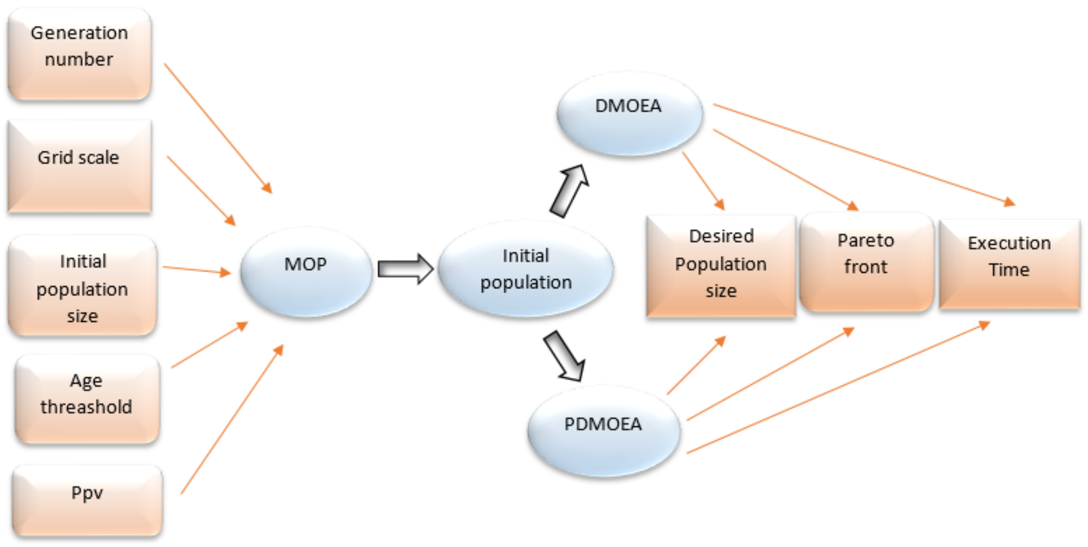


Figure 3.3 – Global Design of the Application.

In our application, we allow the user to identify the following data as inputs:

- Initial population size,
- The desired population size per unit volume,
- A_{th} is a specified age threshold,

- Generation number giving by multi-objective optimization problem,
- Grid scale for each objective function depending on dimension of objective space.

3.4 Sequential version of DMOEA

The meaning of sequential version of DMOEA is that we implement DMOEA as it is described in paper [YL03]. DMOEA is divided into four main functions invoked in a sequential manner until we get the desired population with the desired size. In this section, we will detail DMOEA and how to implement its functions and what we will get as results.

3.4.1 Analysis

DMOEA is an evolutionary algorithm proposed in 2002 described in [LY02]. As in the explanation we conducted in the first chapter, MOEAs common propose is to find a set of optimal solutions by using several strategies, to balance between the diversity, and the convergence to the Pareto-optimal solutions. This ultimate goal is far from being accomplished by the existing MOEAs. One of the negative aspects that made them move away from their goals, is the fixed population size to initiate the evolutionary process. It is very difficult to solve MOPs by MOEAs with a fixed population size, since they have to homogeneously distribute the predetermined computation resource to all the possible directions in the objective space [Mit96]. MOEAs may suffer from premature convergence if the population size is too small. If the population is too large, however, undesired computational resources may be incurred and the computational time for a fitness improvement may be too long in practice [TLK01].

Dynamic multi-objective evolutionary algorithms (DMOEA) is proposed to obtain a Pareto front with a desired resolution because the shape and size of the true Pareto front is unknown a priory for most of the MOPs [YL03]. As a description of this algorithm, we can say that DMOEA consists of four main functions and we will detail each function alone, with mentioning that this DMOEA is based on adaptive cell-based rank density estimation. This last means that we consider the objective space as a grid, the number of its cells principle on dimension of MOP and the grid scale in each dimension which we will explain later. Knowing that, we have explained it before in the first chapter by giving mathematical formula which is mentioned in paper [YL03]. The first function is the initial cell-based rank density calculation scheme which initializes principal variable as initial population size to create a population of initial individuals, calculate cells width, initialize objective function and create a grid which contains a number of cells defined in each dimension depending on the grid scale which gives by the multi-objective problem. The second function is a population growth strategy which aims to make population size bigger than the initial one to increase the probability of obtaining better individuals. like all evolutionary algorithms, DMOEA uses crossover and mutation functions to create new offspring. Regardless that these functions are good for creating more individuals which mean an increase in population size. But, this increase will never 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 population growth strategy by decreasing population size using the killing technique to kill each individual that has an age bigger than the age threshold definite 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.

Initial cell-based rank and density calculation scheme:

DMOEA converts the original MOP into a bi-objective optimization problem in algorithmic domain, minimizing rank value and maintaining the density value of each individual as adding or removing individual will affect the rank and density of other individuals. Knowing that, the rank and density values of each individual need to be recalculated after updating the population using grow and decline strategy which will be explained later. For this reason, Cell-Based Rank and Density calculation schema were designed. This scheme is used to solve the dynamic movement of population, we can simply imagine that the objective space is a grid of m dimensions divided into $k_1 \times k_2 \times \ldots \times k_m$ cells, where every cell is the home of zero, one or more individuals.

Population grow function:

Because exploring the cells with minimum rank values and maintaining these cells densities to a desired value are two converted objectives of DMOEA, crossover and mutation operations need to be devised to fit both of the purposes [YL03]. After selecting a fixed number of parents in a selection pool, DMOEA uses diffusion scheme in the objective space [YL03]. Where each individual shares its information with the leading individuals in order to locate its moving direction. If a resulting offspring is located in a cell with a better fitness (a lower or equal rank value, or a lower or equal density value) than its selected parents, it will be kept to the next generation; otherwise, it will not survive. This strategy will guarantee that a newborn individual will have a better fitness value than at least one of its parents. DMOEA encourages an offspring to land in a sparse area. Some offspring will tend to move toward a direction opposite to the true Pareto front where the cells close to the Pareto front are crowded Obviously, these movements hinder the population in converging to the Pareto front [YL03]. A forbidden region concept is proposed in the offspring-generating scheme for the density sub-population. Forbidden region includes all cells strongly dominated by the selected parent.

Population decline function:

A population declining strategy is indispensable to prevent the population size from growing without limits. An individual will be removed or not depends on its *health*, *crowdedness* and *age* indicators defined [YL03]. For an individual in the initial population, initial *age* is one and it will be increased by one in each generation if it will survive. The *age* of a newborn offspring is one and grows generation by generation as long as it is kept [YL03]. To calculate the *age* indicator at generation n, an individual y has an $age(y, n)$. To ensure that an eliminated individual has a low fitness value, DMOEA periodically removes three types of individuals with different likelihoods [YL03]:

1/ Likelihood of Removing the Most Unhealthy Individuals,

2/ Likelihood of Removing the Unhealthy Individuals in the Most Crowded Cells,

3/ Likelihood of Removing Non-dominated Individuals From the Most Crowded Cells After Convergence.

Objective space compression function:

In DMOEA, the bound-aries of the objective are usually selected to be very large, which may be far away from the true Pareto front, to ensure that the entire true Pareto front is covered by the estimated objective space. An objective space compression strategy is designed to adjust the size of the objective space and to make it suitable to search for the true Pareto front with a high precision.

3.4.2 Global Design

After studying the project aim and specifying project problematic, we have to discuss the sub-problems which are carried out in the first DMOEA implementation. After explaining the DMOEA in the last section, the global architecture of DMOEA application must be identified with the set of components, and relations between them in a detailed way. Figure 3.4 shows the global architecture of DMOEA

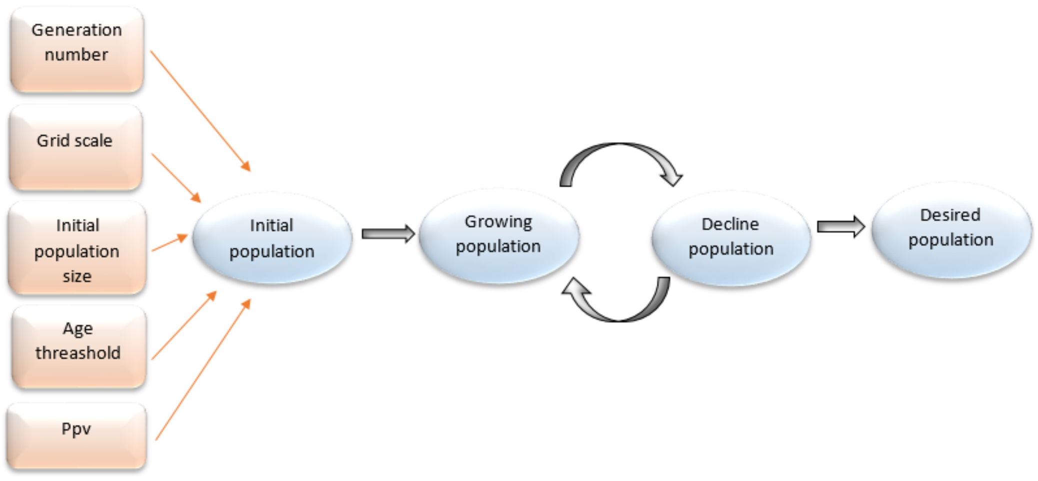


Figure 3.4 – Sequential version design.

In this part of Design we use several UML diagrams to explain classes created in this sequential version and also to represent relationship between their continents using two different diagrams.

Figure 3.5 shows the Classes diagram of the sequential version.

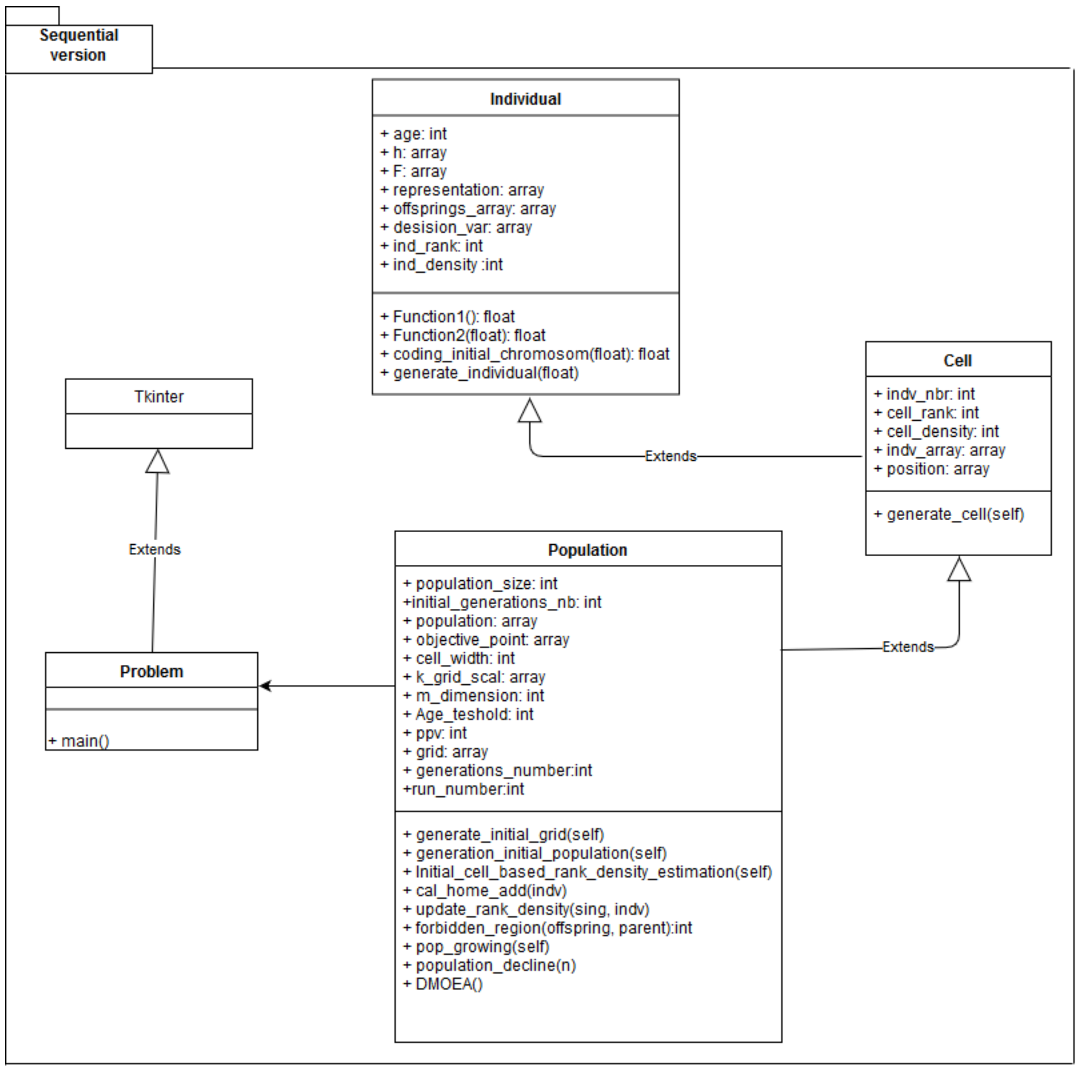


Figure 3.5 – Sequential version classes diagram.

Figure 3.6 shows the sequences diagram of the sequential version.

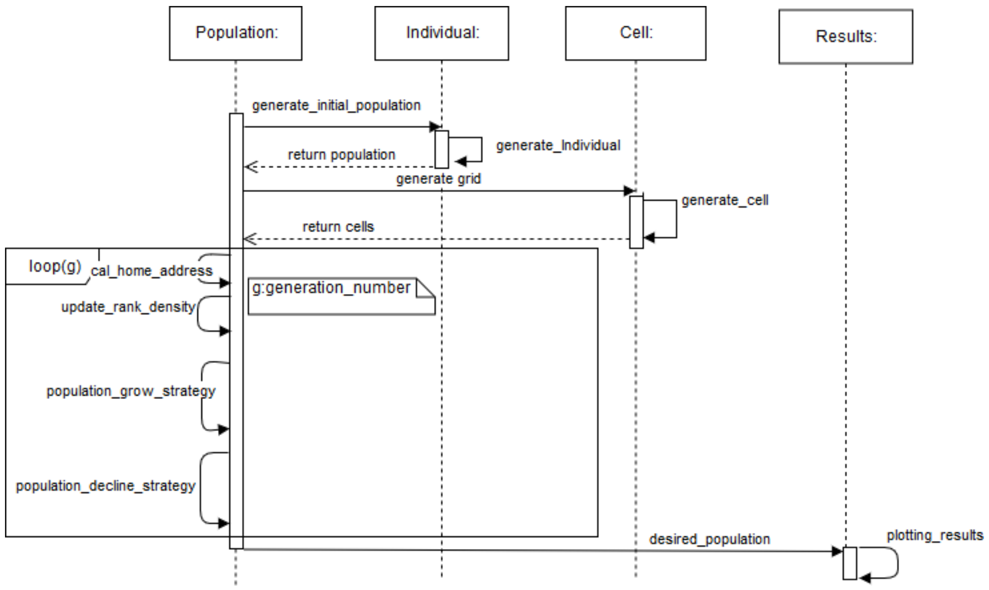


Figure 3.6 – Sequential sequences diagram.

3.4.3 Detailed Design

In this detailed design, detailed description of the coding functions will be discussed starting with exposing classes and detailing each attribute and method in each class.

Individual class :
This class is dedicated to creating an Individual object which is the smallest unit in this design. Individual's principal attribute is its fitness which is divided into objective functions where the number of objective functions determines the dimension of the MOP. Then, individual also has the age attribute which starts by one when a new offspring is accepted and it will increase by one after each generation. An individual can be eliminated if its age is greater or equal the age threshold which is determined in the MOP. The home address should be affected in each individual when offspring is created, it is calculated by a mathematical formula using the objective functions. Also rank and density are attributes in an individual class and they are initialized first by one and zero respectively. Then they will be increased by mathematical formula as we said before in the analysis section. Individual representation is an attribute which contain the decision variables.

create initial chromosome
This is the first method in individual class to generate the random decision variables.

Generate individual
Is the generator method of individual with initialization of all attributes of individual class.

Objective functions
This method creates the fitness of individual object. In our case,the fitness of individual is a mathematical formula defined in [YL03] and [LY02].

Cell class :
This class is dedicated to create the object cell which is used in DMOEA because it used a calculation scheme. This scheme divides the objective space into equal cells in a grid. Each cell has attributes as rank, density (they have the same meaning of rank and density of each individual), position which determines where this cell is located in the grid. The individuals' array attribute which contains individual located in the same home address of this cell and also the attribute of individuals number situated in this array.

Population class :
As we said before, DMOEA is an evolutionary algorithm specified by its dynamic population size until getting the desired population. Population class has the attributes that we had presented before. In this section, we will discuss each attribute of them. Actually, these attributes will be the inputs of the main application later as we mentioned in Figure 3.3 in the global design. These attributes are First m which is the dimension of MOP, initial population size which will be changed dynamically to get the desired one, grid-scale which defines the number of cells in each dimension of objective space usually it is determined by MOP, generation number means the number of iterations of the algorithm. After setting DMOEA class attributes we will pass to the methods. As we said before, this algorithm has four main functions which are the calculation of initial schema, population growth strategy, population decline strategy and the main loop. In Addition to these main functions, there are several other functions. In this section, we will discuss all of these functions using pseudo codes to explain how can we implement these functions.

Generate initial population

As most evolutionary algorithms, the initial population is created first randomly by the population size given in the MOP. Generate initial population is the method which will create this population by a loop function. This creation will invoke the class Individual to create objects type individual surly by using the method ”generate an individual”. The algorithm 8 shows the detail of this function

Algorithm 8 Generate initial population

```
for $i = 1 : initial\_population\_size$ do//i is used as an index of the initial population size
    /* create object individual
    individual.generate_Individual
    /* add individual to the population
```

Generate initial grid

As DMOEA used a cell base-rank and density estimation, it is clear that we will use cells to estimate MOP. To implement this schema we have to divide the objective space of the MOP into a number of cells, determined in each dimension of the MOP to make this objective space look like a grid. The method that creates this grid is shown in algorithm 9, it is dedicated to invoke the object cell.

Algorithm 9 Generate initial grid

```
for $i = 1 : grid\_scale[1]$ do //i is used as an index of the first grid scale
    for $j = 1 : grid\_scale[m]$ do //j is used as an index of the $m$ grid scale
        /* create object cell
        /* initialize cell position [1...m]
        initialize cell.rank = 1
        initialize cell.density = 0
        /* add cell to the cells_array of the initial grid
```

Initial cell based rank density calculation scheme

This scheme uses to solve the dynamic movement of population. We can simply imagine that the objective space is a grid of m dimensions divided into $k_1 \times k_2 \times \ldots \times k_m$ cells, where every cell is the home of zero, one or more individuals. Algorithm 10 shows the initialization of the initial cell based rank density calculation scheme.

Algorithm 10 Initial cell based rank density calculation scheme

```
Generate initial population $P = [P(1), P(2), ..., P(P_0)]$
for $i = 1 : m$ do
    //i is an index to iterate the dimension
    /* get min($F1..Fm$)
    /* get max($F1..Fm$)
    $c = (F_1^{\min}, F_2^{\min}, ..., F_m^{\min})$
for $i = 1 : m$ do
    $d_i = \frac{\max_{x \epsilon X} f_i(x) - \min_{x \epsilon X} f_i(x)}{k_i}, (i = 1, ..., m)$
/* generate initial grid
for $i = 1 : P_0$ do
    /*Calculate home address $[h_1(i), h_2(i), ..., h_m(i)]$of individual $i$
    $[h_1(i), h_2(i), ..., h_m(i)] = cal\_home\_add(P(i), e)$
    /*Update rank and density matrices
    $sign = 1;$
    $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), ..., h_m(i), sign, m\_d, m\_r)$
```

Calculate home address

Algorithm 11 describe the method which calculates home address of each individual in the population.

Algorithm 11 Home address

```
function $[h_1(i), h_2(i), ..., h_m(i)] = cal\_home\_add(P(i), e)$
    for $j = 1 : m$ do
        /*Calculate the distance between the individual and the original point in the
        $j$th dimension
        $t_j = p_j - F_j^{\min}$;
        /*locate the home address of $p$ on the $j$th dimension
        $h_j = \mod (t_j, d_j) + 1$;
```

Update rank and density

Algorithm 12 the method to update rank and density of each individual in each population.

Algorithm 12 Updating rank and density matrices

```
function $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), .., h_m(i)], sign, m\_d, m\_r)$
    for $j = 1 : m$ do
        /*Update rank density matrices
        $m\_d(h_1, h_2, ..., h_m) = m\_d(h_1, h_2, ..., h_m) + sign \times 1$
        $m\_r([h_1..k_1], [h_2..k_2], ., [h_m..k_m]) = m\_r([h_1..k_1], [h_2..k_2], .., [h_m..k_m]) + sign \times 1$
```

Forbidden region

DMOEA encourages an offspring to land in a sparse area. Some offspring will tend to move towards a direction opposite to the true Pareto front where the cells close to the Pareto front are crowded. Obviously, these movements hinder the population in converging to the Pareto front [YL03]. A forbidden region concept is proposed in the offspring-generating scheme for the density sub-population. A forbidden region includes all cells strongly dominated by the selected parent. This forbidden region is shown in Algorithm 13.

Algorithm 13 Forbidden region

```
function $F\_r = forbidden\_region([p\_h_1, p\_h_2, .., p\_h_m], [o\_h_1, o\_h_2, .., o\_h_m])$
    F_r=1;
    for $j = 1 : m$ do
        /*Calculate distance between an offspring and its parent in the $j$th dimension
        $z_j = o\_h_j - p\_h$;
        if $z_j \prec= 0$ then
            F_r = 0
    end
```

parent selection

Algorithm 14 shows the method to select parents from population to create a new offspring.

Algorithm 14 Select parent

```
pool = [ ]
minRank = $cal\_min\_rank$
for individual in $population\_size$ do
    if individual.rank == minRank then
        /* add individual to pool
```

Population growing strategy

After selecting a fixed number of parents in a selection pool, DMOEA uses diffusion scheme in the objective space. Each individual shares its information with the leading individuals in order to locate its moving direction. If a resulting offspring is located in a cell with a better fitness (a lower or equal rank value, or a lower or equal density value) than its selected parents, it will be kept to the next generation; otherwise, it will not survive. This strategy will guarantee that a newborn individual will have a better fitness value than at least one of its parents. This population growing strategy implementation is shown in Algorithm 15.

Algorithm 15 Population growing strategy

```
if (population size =2) then
    offspring = crossover ($p1, p2$)
    offspring.H = $cal\_home\_add(o, c)$
    /* Initialize offspring.rank and offspring.density
    if o has lower or rank equal value to any of its parent) then
        /*keep offspring o in the population and update m_r and m_d
        $sign = 1;$
        $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), ..., h_m(i)], sign, m\_d, m\_r)$
        $F\_r = 0$
    if (o has<= density value transformed to any of its parent) and ($F\_r = 0$) then
        /*keep offspring o in the population and update m_r and m_d
        $sign = 1;$
        $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), ..., h_m(i), sign, m\_d, m\_r)$
        $N = N + 1;$
/* $calculate\_max\_rank$
if $max\_Rank <> 1$ then
    /* Select parent
    offspring = crossover ($p1, p2$)
    offspring.H = $cal\_home\_add(o, c)$
    /* Initialize offspring.rank and offspring.density
    if (o has lower or rank equal value to any of its parent) then
        /*keep offspring o in the population and update m_r and m_d
        $sign = 1;$
        $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), ..., h_m(i)], sign, m\_d, m\_r)$
        $F\_r = 0$
    if (o has<=density value transformed to any of its parent) and ($F\_r = 0$) then
        /*keep offspring o in the population and update m_r and m_d
        $sign = 1;$
        $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), ..., h_m(i), sign, m\_d, m\_r)$
        $N = N + 1;$
/* Select2 parent
offspring = crossover ($p1, p2$)
offspring.H = $cal\_home\_add(o, c)$
/* Initialize offspring.rank and offspring.density
if (o has lower or rank equal value to any of its parent) then
    /*keep offspring o in the population and update m_r and m_d
    $sign = 1;$
    $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), ..., h_m(i)], sign, m\_d, m\_r)$
    $F\_r = 0$
if (o has<=density value transformed to any of its parent) and ($F\_r = 0$) then
    /*keep offspring o in the population and update m_r and m_d
    $sign = 1;$
    $[m\_d, m\_r] = update\_rank\_density([h_1(i), h_2(i), ..., h_m(i), sign, m\_d, m\_r)$
    $N = N + 1;$
for $i = 1 : P_0$ do
    /*Update the rank and density values rank(i) and density(i) of individual i
    $rank(i) = m\_r([o\_h_1, o\_h_2, .., o\_h_m])$
    $density(i) = m\_d([o\_h_1, o\_h_2, .., o\_h_m])$
```

Population decline strategy

A population declining strategy is necessary to prevent the population size from growing without limits. An individual will be removed or not depends on its *health*, *crowdedness* and *age* indicators defined [YL03]. An individual in the initial population has an initial *age* equal to one and it will be increased by one in each generation if it will survive. This population decline strategy implementation is described in Algorithm 16.

Algorithm 16 Population decline strategy

```
$R = \{\}$ ;
$q = 0$;
for i=1:$population\_size$ do
    /*generate three random numbers between 0 and 1
    if i in $max\_rank$ then or i in $max\_rank\_density$
        $rand_1 = rand[0,1]$
        $rand_2 = rand[0,1]$
        if $l_1^i > rand_1$ and $l_2^i > rand_2$ then
            /* individual $P(i)$ will be removed
    if i.density ¿ ppv then
        $rand_1 = rand[0,1]$
        if $l_3^i > rand_3$ then
            /* individual $P(i)$ will be removed
for $i = 1 : N$ do
    $sign = -1$;
    /*Update the rank and density values rank(i) and density(i) of individual i
    $rank(i) = m\_r([o\_h_1, o\_h_2, .., o\_h_m])$
    $density(i) = m\_d([o\_h_1, o\_h_2, .., o\_h_m])$
```

DMOEA

This last method can resume the steps of the whole algorithm. Algorithm 17 describes the main loop implementation of DMOEA.

Algorithm 17 DMOEA

```
for i=1:$run\_number$ do
    $Initial\_cell\_based\_rank\_density\_calculation\_schema$
    for i=1:$generation\_number$ do
        $Population\_growing\_strategy$
        $Population\_decline\_strategy$
    /* getting desired population
```

3.5 Parallel version of DMOEA

3.5.1 Analysis

PDMOEA is an evolutionary algorithm proposed in this thesis as a result of our own contribution to get better results with less executing time, improved objective space and a maximum desired population size. This algorithm has the same main functions of DMOEA but the only difference is that PDMOEA divides generation number into n threads. After dividing the generation number to the n threads each process will create an initial population and makes it growing and declining until it gets the desired population size with the best solutions.

PDMOEA use the "master/slave"[1] principle where the main is the master and the n-processes are the slaves.

3.5.2 Global Design

The global architecture of PDMOEA application must be identified with the set of components, and discuss the relation between them in a detailed way. Figure 3.7 shows the global architecture of PDMOEA

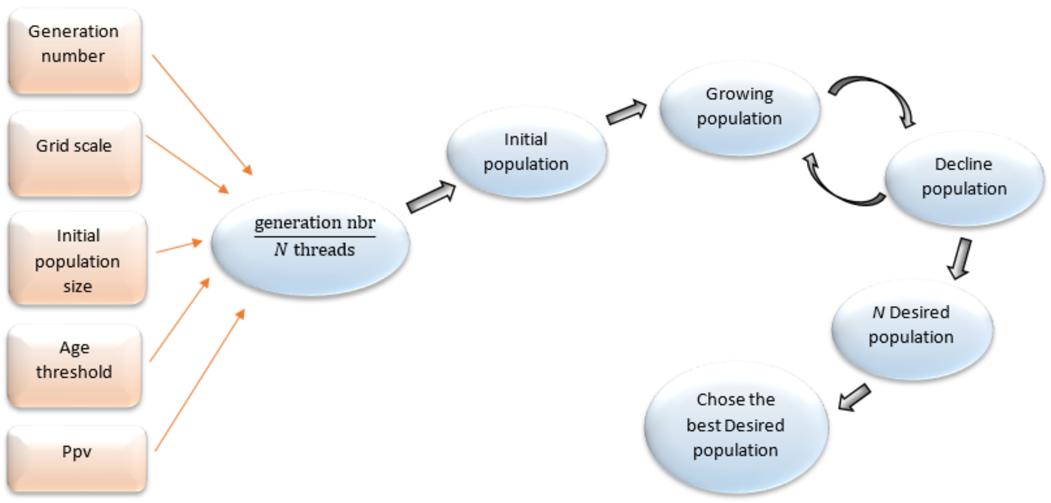


Figure 3.7 – Parallel version design.

In this part of design we use several UML diagrams to explain classes created in this parallel version and also to represent relationship between their components.

Figure 3.8 shows the classes diagram of this parallel version.

[1]Master/slave is a model of communication where one device or process has unidirectional control over one or more other devices. In some systems a master is selected from a group of eligible devices, with the other devices acting in the role of slaves [19a].

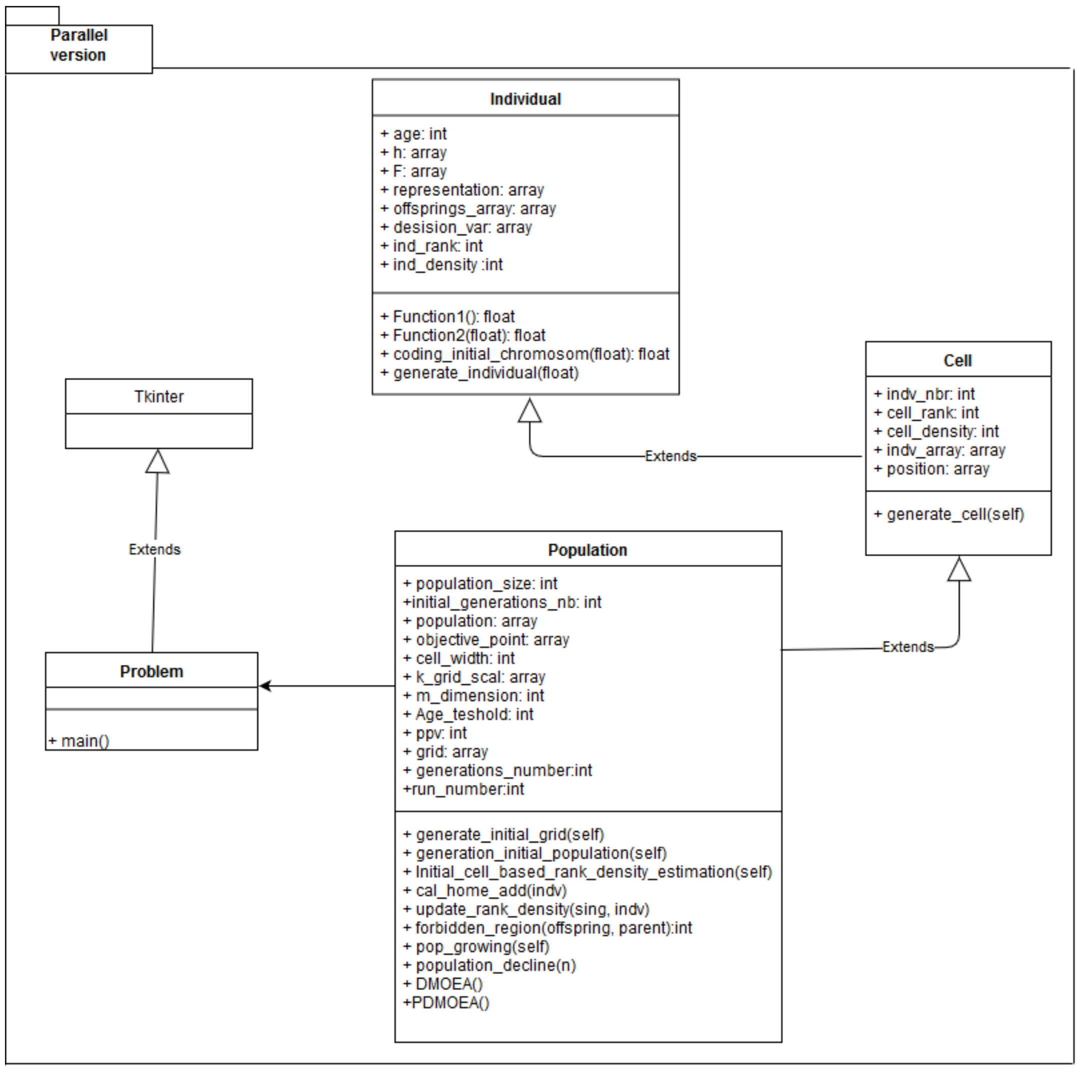


Figure 3.8 – Parallel classes diagram.

Figure 3.9 shows the sequences diagram this parallel version.

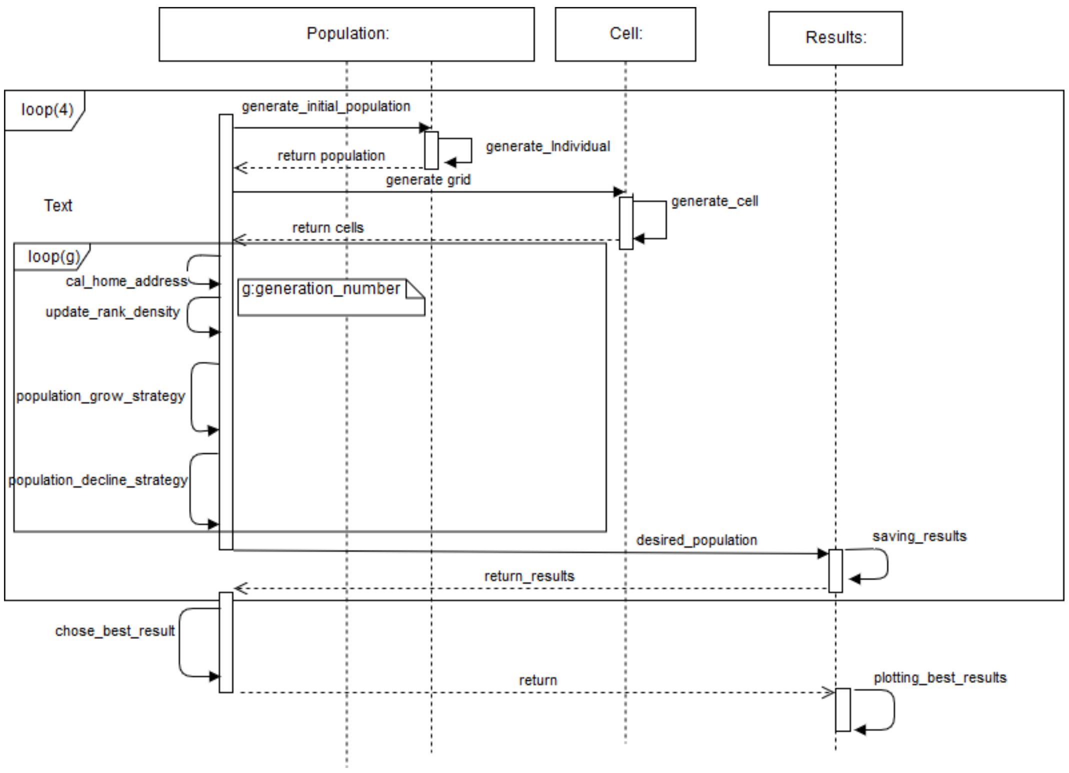


Figure 3.9 – Parallel sequence diagram.

3.5.3 Detailed Design

In the parallel version, a new method is added which is PDMOEA. The rests are similar. The difference starts when we use the package multiprocessing to create n threads. PDMOEA divides the generations number into n threads and chooses the result of the best one. The pseudo-algorithm of parallel main loop method is described using algorithm 18 without re-explaining the other methods because they will not change.

Algorithm 18 PDMOEA

```
q = mp.Queue()
pool_process = [ ]
for i=0:4 do
    p = mp.Process(target=self.main_loop)
    p.start()
    pool_process.append(q.get())
    p.join()
```

Where the Queue class is a near clone of Queue.Queue.

Conclusion

In this chapter, we have presented our contributions. This chapter is divided into five sections. The last two sections are considering as the main sections, where we present in each section of them the design and analysis concerning one of the two versions (i.e., sequential and parallel version) using UML diagrams in each global design and detailed description of the creating classes in each detailed design. The next chapter is dedicated to implement the whole application which gather the both versions (i.e., DMOEA and PDMOEA).

Chapter 4

Implementation & Experimental study

Introduction

After the analysis and design of each version (i.e., sequential and parallel) with a detailed description for all the project cycle. The next step is the implementation of this DMOEA and PDMOEA. This chapter aims to code the both versions (i.e., DMOEA and PDMOEA), implement a global application gathers both versions, test this application using several MOPs, and compare the results of these two algorithms to conclude the best version.

This chapter is divided into four sections. The first section introduces briefly the development tools and languages that we have exploited in the realization of our project. The second section describes the main implementation results of our final application. The third section is dedicated to test the both versions (i.e., DMOEA and PDMOEA) using two MOPs from two different papers (i.e., [YL03] and [LY02]). The last section presents a comparative study to prove the efficiency of DMOEA against EAs (i.e., NSGA-II, SPEA-II,...) and the efficiency of PDMOEA against DMOEA.

The comparison we are talking about requires specific factors and indicators to compare them. These factors are mentioned each time when both versions are compared. These factors are the improvement of objective space, the minimization of computational time and the maximization of the desired population size. Each of these factors will be explained in a detailed and precise manner later.

4.1 Development Tools and Languages

In this section, we present different tools and languages, that help us during the realization of our project in both levels (programming level, and theoretical level).

4.1.1 Python programming language

Python is an intelligent programming language that we have used it in the implementation of our application. It is easy to learn, because it is flexible, and its syntax doesn't hard to learn. A Python program is short than other languages' programs, because of the availability of many implemented functions. Python is an open source and untyped programming language. It is available for all these operating system (Windows, LINUX, Mac OS).

4.1.2 PyCharm Programming Editor

PyCharm is an open source Integrated Development Environment (IDE), used for python programming. It is a powerful coding assistant, it can highlight errors and introduces quick fixes based on an integrated Python debugger. It is a suitable editor for writing and testing many lines of code and classes, since it offers a structural project view, and a quick files navigation.

4.1.3 Tool Kit Interface “Tkinter” Package

Tool Kit Interface in short “*Tkinter*” [05], it is an open source *Graphical User Interface* (*GUI*) package. It is intended for Python programming language. We have preferred the *Tkinter* toolkit for developing GUIs of our application, because it is simple to learn it , and it is a powerful toolkit. It is available on both operating systems (Windows, Linux, and Mac OS).

4.1.4 Plotting Library “matplotlib”

matplotlib [17b] is an open source Python library. It is used for 2D plotting. With a short code, one can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc, and can produce quality figures for the generated plots in a variety of hardcopy formats. Line styles, font properties, and axes properties are controlled by simple lines of code. Since in our project we have many results that must be plotted, thus we have chosen *matplotlib* to plot them.

4.1.5 Process-based ”threading” “interface”

`multiprocessing`

multiprocessing [19c] is a package that supports spawning processes using an API similar to the threading module. The multiprocessing package offers both local and remote concurrency, effectively side-stepping the Global Interpreter Lock by using sub-processes instead of threads. Due to this, the multiprocessing module allows the programmer to fully leverage multiple processors on a given machine. It runs on both Unix and Windows. The multiprocessing module also introduces APIs which do not have analogs in the threading module. A prime example of this is the Pool object which offers a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism)[19c].

4.1.6 Document Preparation System LaTeX

LaTeX [Lam17] is a powerful and flexible typesetting system for producing high quality technical and scientific papers. It based on the tags language. It follows the design philosophy of separating presentation from content, thus authors focus on what they are writing, not on what is displayed, because the appearance is handled by LaTeX. The appearance includes many aspects, document structure (part, chapter, section, ..etc), figures, cross-references and bibliographies. It is more familiar to a computer programmer, because it follows the code-compile-execute cycle.

4.1.7 Typesetting Editor (TEX MAKER)

TEX MAKER [17a] is a free and open source editor for drafting papers, based on LATEX system. It supports a powerful spell-checker, code auto-completion, and a *pdf* displayer. We have used TEX MAKER to draft our report and make our presentation, because it produces high quality papers and talks.

4.1.8 Desktop Diagram Application "Draw.io"

draw.io [19b] Desktop is a completely free, stand-alone desktop diagramming application by the technology leaders in web diagramming. No registration, no limitations, no catches.

4.2 Implementation

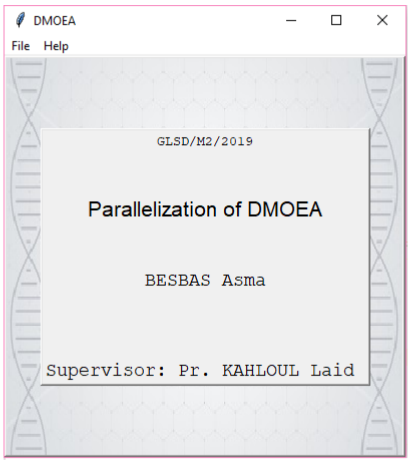
The studied algorithms are implemented using oriented object programming (OOP) paradigm, and a set of software and hardware which are summarized in Table 4.1.

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

Table 4.1 – Software/Hardware versions

Main Application:

Figure 4.1 shows the main GUI of the whole application.


DMOEA

File Help

GLSD/M2/2019

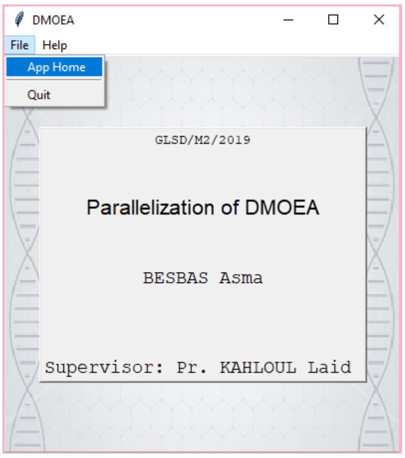
Parallelization of DMOEA

BESBAS Asma

Supervisor: Pr. KAHLOUL Laid

Figure 4.1 – Main GUI.

In our implementation, we developed many graphical user interfaces (GUIs) to facilitate the use of the application. The application contains a menu bar with two menus (File and Help), Figures 4.2-4.4 illustrate each menu and its commands.



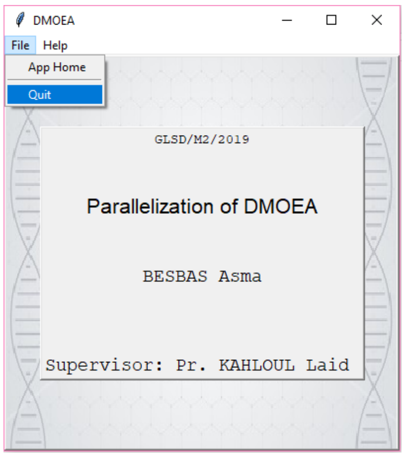
GLSD/M2/2019

Parallelization of DMOEA

BESBAS Asma

Supervisor: Pr. KAHLOUL Laid

Figure 4.2 – App home menu.



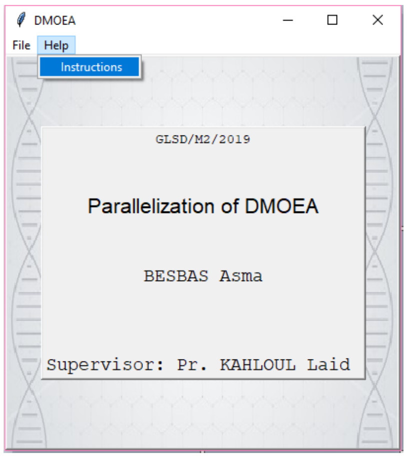
GLSD/M2/2019

Parallelization of DMOEA

BESBAS Asma

Supervisor: Pr. KAHLOUL Laid

Figure 4.3 – Quit menu.



GLSD/M2/2019

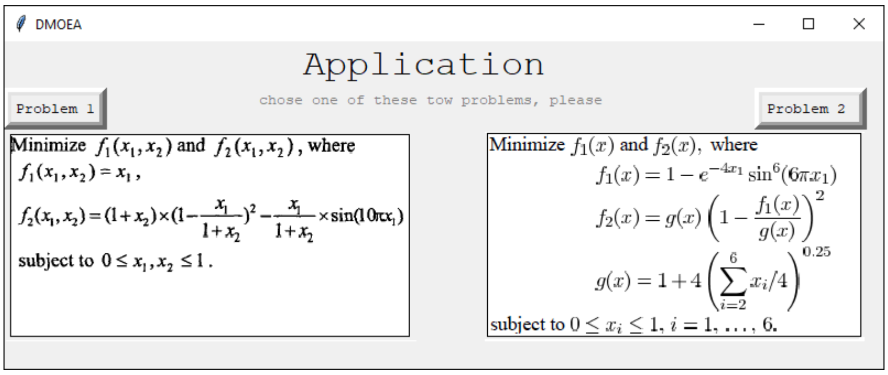
Parallelization of DMOEA

BESBAS Asma

Supervisor: Pr. KAHLOUL Laid

Figure 4.4 – Help menu.

File menu offers two functionalities for the user to create new frame, choosing between MOPs or quite. The main objective of this application is not to implement just these two problems and being satisfied with results but the main objective is to resolve every MOP giving to this version. Figure 4.5 shows our application main GUI.



Application

chose one of these tow problems, please

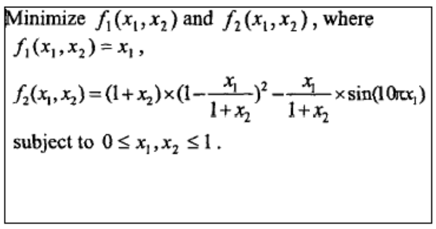
Minimize $f_1(x_1,x_2)$ and $f_2(x_1,x_2)$, where
$f_1(x_1,x_2)=x_1$,
$f_2(x_1,x_2)=(1+x_2)\times(1-\frac{x_1}{1+x_2})^2-\frac{x_1}{1+x_2}\times\sin(10\pi x_1)$
subject to $0\le x_1,x_2\le 1$.

Minimize $f_1(x)$ and $f_2(x)$, where
$$f_1(x)=1-e^{-4x_1}\sin^6(6\pi x_1)$$
$$f_2(x)=g(x)\left(1-\frac{f_1(x)}{g(x)}\right)^2$$
$$g(x)=1+4\left(\sum_{i=2}^{6}x_i/4\right)^{0.25}$$
subject to $0\le x_i\le 1$, $i=1,\ldots,6$.

Figure 4.5 – Application home.

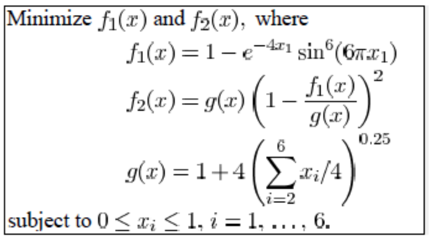
Application Home:

To implement this algorithm (DMOEA) we use two mathematical problems. The first MOP in [YL03] designed by Deb as the preliminary test function that has a discontinuous Pareto front. The second MOP is a 6-D Decision Space proposed in [LY02]. This two problems are formulated in Figures 4.6 and 4.7



Minimize $f_1(x_1,x_2)$ and $f_2(x_1,x_2)$, where
$f_1(x_1,x_2)=x_1$,
$f_2(x_1,x_2)=(1+x_2)\times(1-\frac{x_1}{1+x_2})^2-\frac{x_1}{1+x_2}\times\sin(10\pi x_1)$
subject to $0\le x_1,x_2\le 1$.

Figure 4.6 – MOP 1 (From [LY02]).



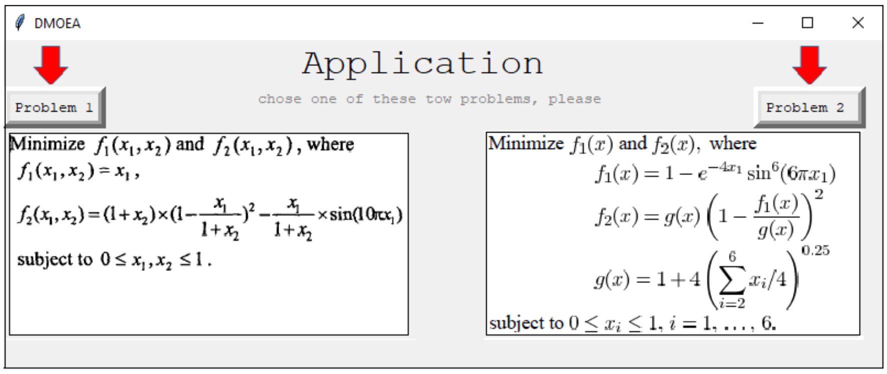
Minimize $f_1(x)$ and $f_2(x)$, where
$$f_1(x)=1-e^{-4x_1}\sin^6(6\pi x_1)$$
$$f_2(x)=g(x)\left(1-\frac{f_1(x)}{g(x)}\right)^2$$
$$g(x)=1+4\left(\sum_{i=2}^{6}x_i/4\right)^{0.25}$$
subject to $0\le x_i\le 1$, $i=1,\ldots,6$.

Figure 4.7 – MOP 2 (From [YL03]).

In order to help the user, we have used many GUIs (Graphical User Interface). Each button opens a new GUI either to enter the data or to show the results related to the both versions.

Problems GUIs:

The first button is dedicated to the first problem shown in Figure 4.6 and the second button is for the problem in Figure 4.7. Both buttons take as to the same second GUI because DMOEA always take the same parameters with different values. Figure 4.8 shows the adopted MOPs in this application.

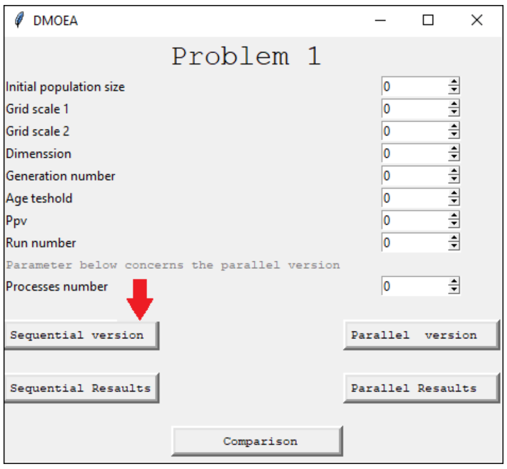


Minimize $f_1(x_1,x_2)$ and $f_2(x_1,x_2)$, where
$f_1(x_1,x_2)=x_1$,
$f_2(x_1,x_2)=(1+x_2)\times(1-\frac{x_1}{1+x_2})^2-\frac{x_1}{1+x_2}\times\sin(10\pi x_1)$
subject to $0 \le x_1, x_2 \le 1$.

Minimize $f_1(x)$ and $f_2(x)$, where
$$f_1(x)=1-e^{-4x_1}\sin^6(6\pi x_1)$$
$$f_2(x)=g(x)\left(1-\frac{f_1(x)}{g(x)}\right)^2$$
$$g(x)=1+4\left(\sum_{i=2}^{6}x_i/4\right)^{0.25}$$
subject to $0 \le x_i \le 1$, $i=1,\ldots,6$.

Figure 4.8 – MOP buttons.

Figures 4.9 and 4.10 show the sequential and parallel buttons. Such as the sequential button implements DMOEA and the parallel button implement PDMOEA.

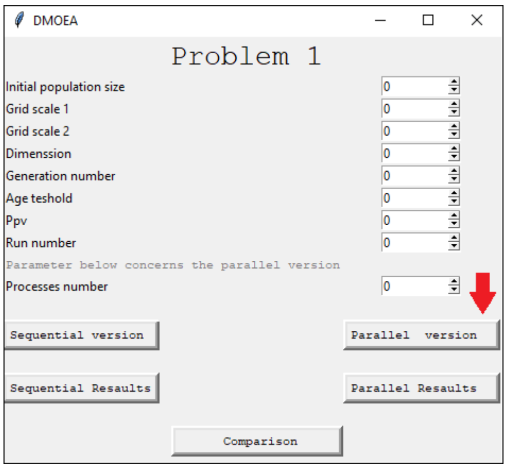


Initial population size	0
Grid scale 1	0
Grid scale 2	0
Dimenssion	0
Generation number	0
Age teshold	0
Ppv	0
Run number	0

Parameter below concerns the parallel version

Processes number	0

Figure 4.9 – Sequential button



Initial population size	0
Grid scale 1	0
Grid scale 2	0
Dimenssion	0
Generation number	0
Age teshold	0
Ppv	0
Run number	0

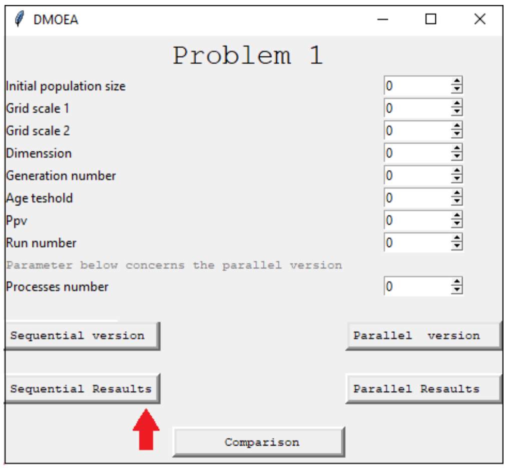
Parameter below concerns the parallel version

Processes number	0

Figure 4.10 – Parallel button

"Sequential Button" and "Parallel Button" are dedicated to execute the Sequential and Parallel version of DMOEA. Figures 4.11 and 4.12 show these two buttons.

Results GUIs:



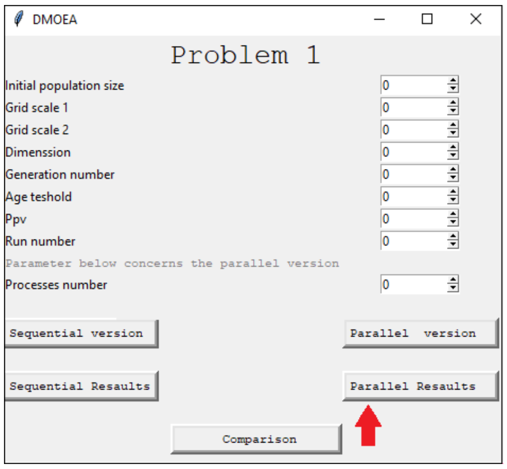
Problem 1

Initial population size	0
Grid scale 1	0
Grid scale 2	0
Dimenssion	0
Generation number	0
Age teshold	0
Ppv	0
Run number	0

Parameter below concerns the parallel version

Processes number	0

Figure 4.11 – Sequential results Button



Problem 1

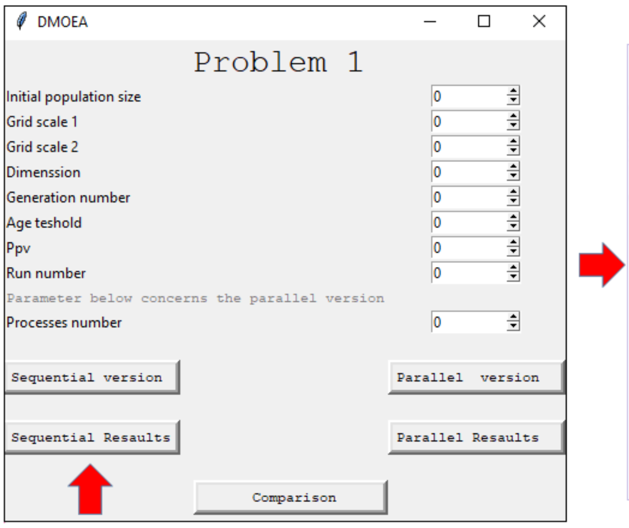
Initial population size	0
Grid scale 1	0
Grid scale 2	0
Dimenssion	0
Generation number	0
Age teshold	0
Ppv	0
Run number	0

Parameter below concerns the parallel version

Processes number	0

Figure 4.12 – Parallel results Button

These two buttons' work is to display plot result of DMOEA and PDMOEA respectively using matplotlib as we said before. These results and plots are divided into 4 plots as Figure 4.13 and Figure 4.14 show where each plot is dedicated to show a part of DMOEA results. The first plot is the evolution of population size generation by generation, the second and the third show the evolution of rank and density also generation by generation and the last describes the objective space where all individuals are located and converging generation by generation until the Pareto front which is described with pink color without forgetting that these individuals with pink color in the Pareto front represent the last population of the last generation.



Problem 1

Initial population size	0
Grid scale 1	0
Grid scale 2	0
Dimenssion	0
Generation number	0
Age teshold	0
Ppv	0
Run number	0

Parameter below concerns the parallel version

Processes number	0

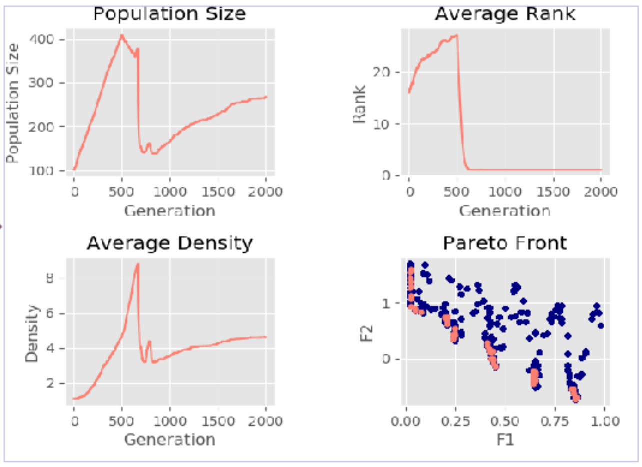
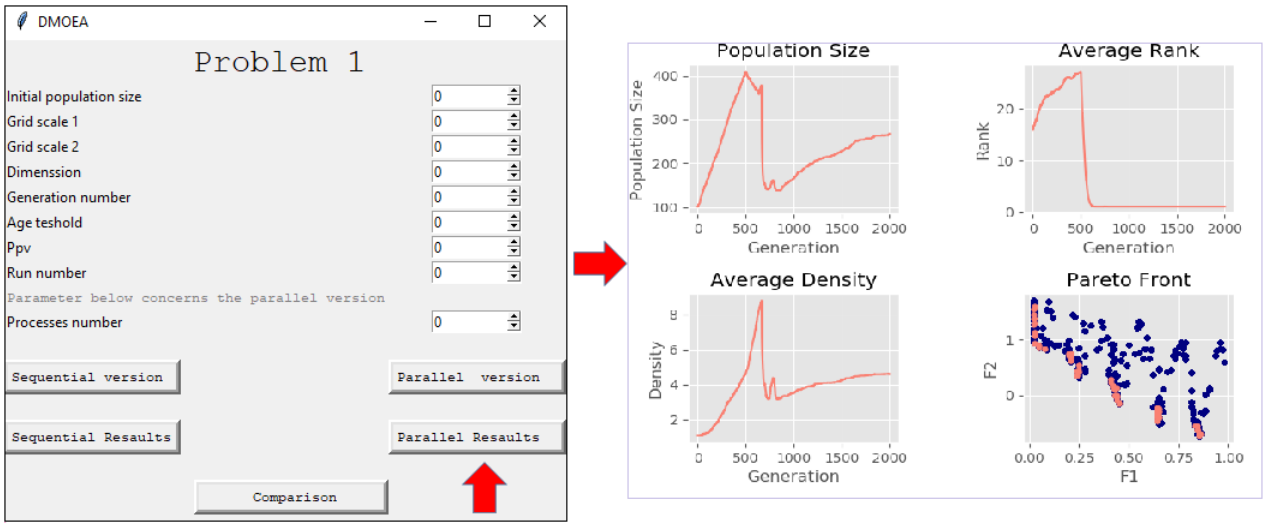


Figure 4.13 – Sequential results plots.



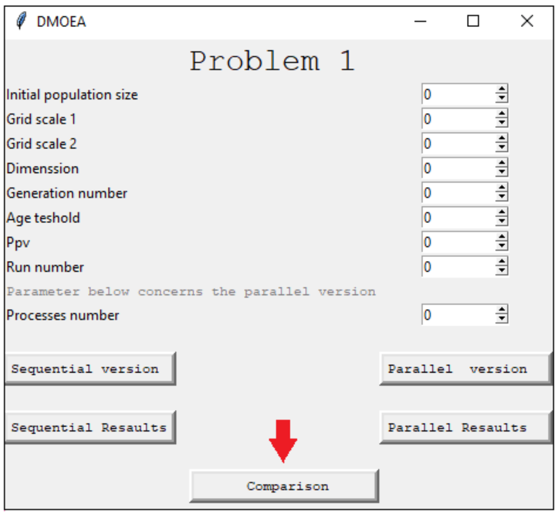
DMOEA

Problem 1

Initial population size	0
Grid scale 1	0
Grid scale 2	0
Dimenssion	0
Generation number	0
Age teshold	0
Ppv	0
Run number	0
Parameter below concerns the parallel version	
Processes number	0

Figure 4.14 – Parallel results plots.

Also, problems GUIs contain comparison button which dedicated to display the objective space of the last population of the last generation which means the pareto front of both versions DMOEA and PDMOEA. Figure 4.15 shows the comparison button and the Figure 4.16 shows the result when we click on it.

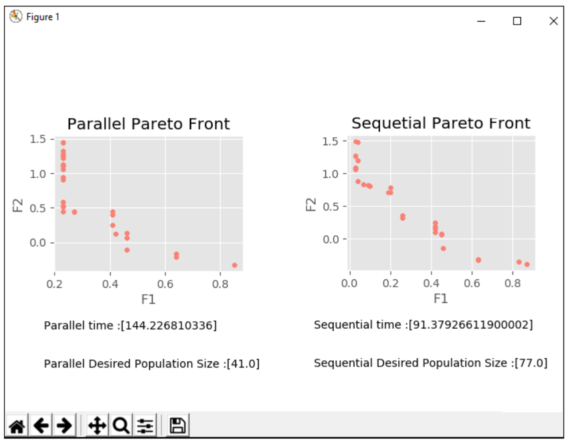


DMOEA

Problem 1

Initial population size	0
Grid scale 1	0
Grid scale 2	0
Dimenssion	0
Generation number	0
Age teshold	0
Ppv	0
Run number	0
Parameter below concerns the parallel version	
Processes number	0

Figure 4.15 – Comparison Button



Parallel time :[144.226810336]

Sequential time :[91.37926611900002]

Parallel Desired Population Size :[41.0]

Sequential Desired Population Size :[77.0]

Figure 4.16 – Comparison plot

Figure 4.17 shows the comparative factors to chose the best version.

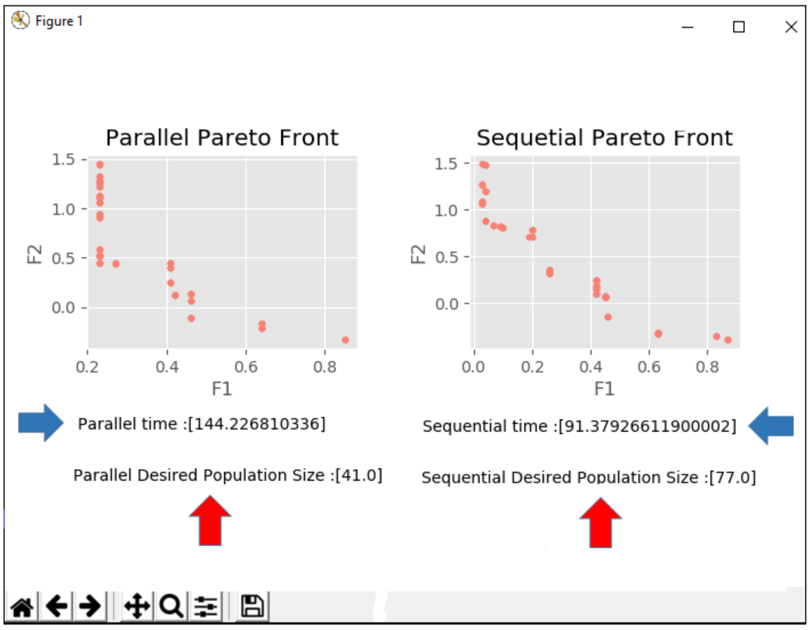


Figure 1
Parallel Pareto Front
Sequetial Pareto Front
F1
F2
Parallel time :[144.226810336]
Sequential time :[91.37926611900002]
Parallel Desired Population Size :[41.0]
Sequential Desired Population Size :[77.0]

Figure 4.17 – Comparison factors.

The blue arrow refers to the time related to both versions and the red arrow refers to the desired population size in each version.

This application is created to facilitate the comparison between the results of the sequential version and the parallel version. Now, we will test this application and the credibility of our application results using inputs mentioned in [YL03] and [LY02] and compare the results.

4.3 Test & Experimental Study

We divide this test and experimental study into two parts. The first part is to discuss the results of sequential version using two MOPs provided in different papers [YL03] and [LY02] and to test the credibility of the application results. The second part is to prove the efficiency of PDMOEA".

4.3.1 Experimental results of DMOEA:

First chosen MOP:
This test function is proposed in [LY02]. For the given test function, they select the boundary of the feasible objective space to be [0, 1] and [-1, 2.5] and the number of cells of each dimension proposed to be $K_1 = 50$ and $K_2 = 100$. The desired population size per cell is predefined as ppv = 5 . Three specified initial population sizes: 2, 30 and 100 are chosen to test the robustness of DMOEA. The age threshold, the stopping generation, the chromosome length of each decision variable, the crossover rate and the mutation rate are chosen to be 10, 2000, 15, 0.7, and 0.1, respectively in the simulation. Figure 4.18 shows the results when the initial population size is 2.

- Initial population size= 2,
- Grid scale= 50,
- Grid scale= 100,
- Dimension= 2,
- Generation number= 2000,
- Age threshold= 10,
- ppv= 5,
- Run number= 30.

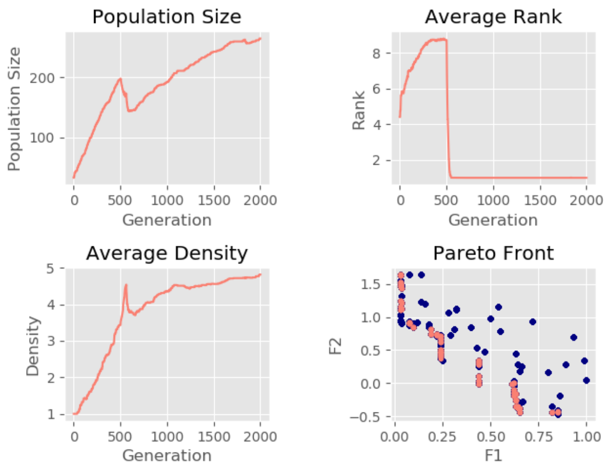


Figure 4.18 – Results with initial pop size=2

Figure 4.19 shows the results when the initial population size is 30

- Initial population size= 30,
- Grid scale= 50,
- Grid scale= 100,
- Dimension= 2,
- Generation number= 2000,
- Age threshold= 10,
- ppv= 5,
- Run number= 30.

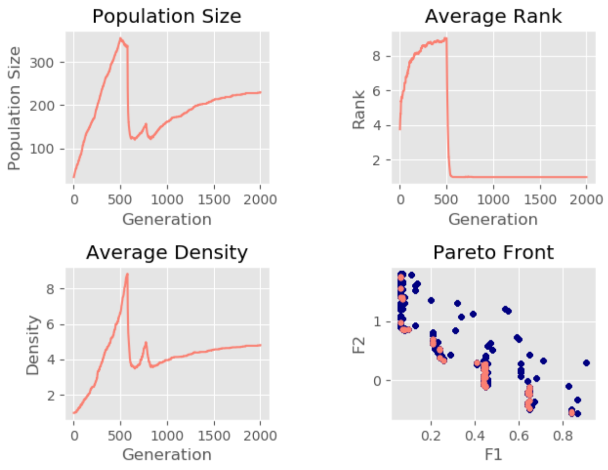


Figure 4.19 – Results with initial pop size=30

Figure 4.20 shows the results when the initial population size is 100

- Initial population size=100,
- Grid scale= 50,
- Grid scale= 100,
- Dimension= 2,
- Generation number= 2000,
- Age threshold= 10,
- ppv= 5,
- Run number= 30.

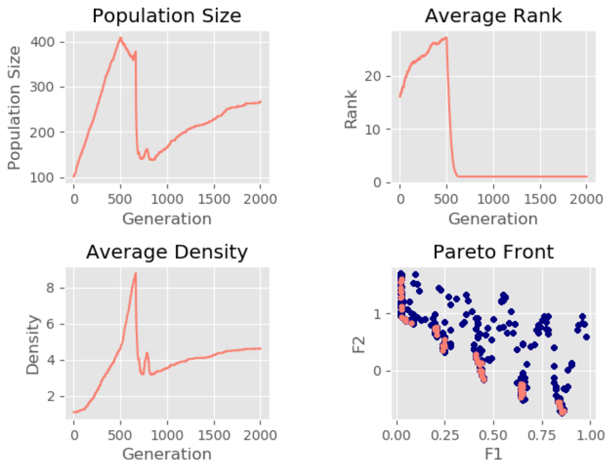


Figure 4.20 – Results with initial pop size=100

These were the results which we had obtained in our test of DMOEA, which represent the evaluation of population size, rank and density and also location in the objective space until we get the desired population size with fix average rank which equal to one and a fix average density which equal to *ppv*

Now, we will incorporate results related to paper [LY02] to compare the results of each plot without forgetting that the desired population size is an approximate number because the first creation of the initial population is always random.

The plots are divided into 4 plots as what we had created in our application. Where the plot box contains the evolution of population size, the average rank plot and the average density plot. But the difference between this plots and our application results is that each plot contains the result of the 3 experiences with three different initial population size where the first is 2, the second is 30 ,and the last is 100.

Figure 4.21 represents the evolution of population size generation by generation, Figure 4.22 represents the average rank, Figure 4.23 represents the average density and Figure 4.24 represents the objective space with the hole individuals and it also indicates the Pareto front by dark color.

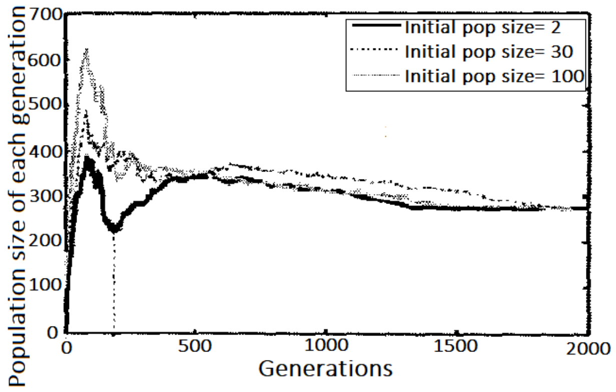


Figure 4.21 – Population size (From [LY02]).

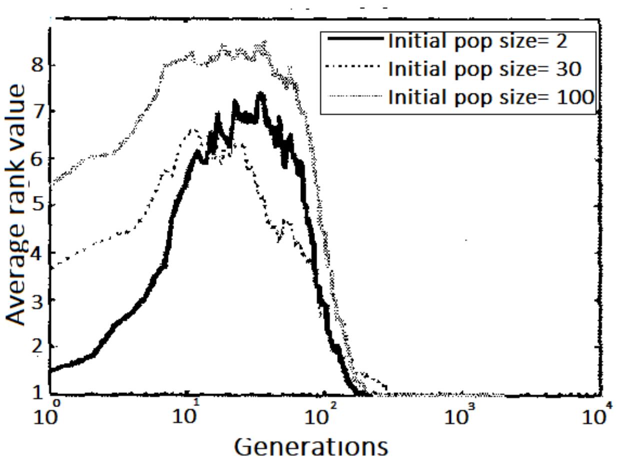


Figure 4.22 – Average rank (From [LY02]).

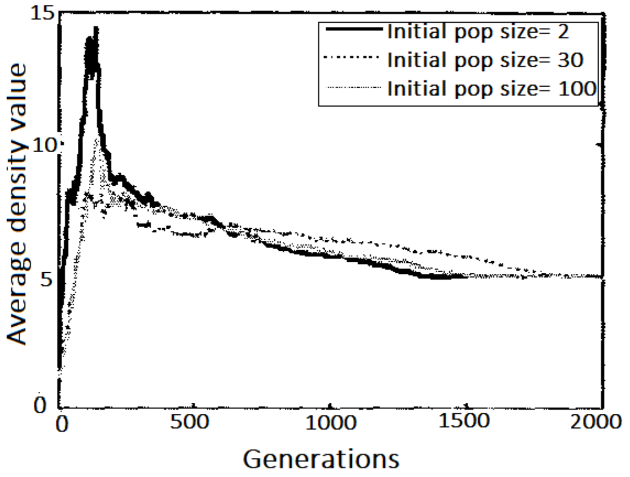


Figure 4.23 – Average density (From [LY02]).

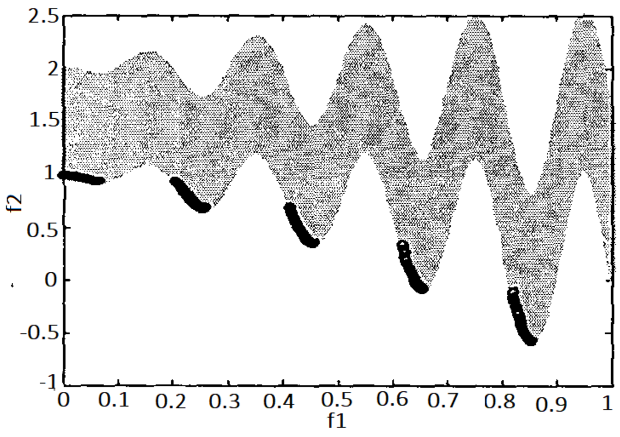


Figure 4.24 – Objective space (From [LY02]).

Figure 4.23 – Average density (From [LY02]).

Second chosen MOP:
The second MOP is proposed in paper [YL03]. This test function is as an MOP with a 6-D decision space and local Pareto fronts in objective space as shown in Figure 4.7. The specific parameter settings for function using the snapshots of the objective space and individuals resulted from DMOEA at generations 10000. The number of cells of each dimension proposed to be $K_1 = 30$ and $K_2 = 30$. The desired population size per cell cell is predefined as ppv = 3 . one specified initial population sizes 2 is chosen to test the robustness of DMOEA. The age threshold, the stopping generation, the chromosome length of each decision variable, the crossover rate and the mutation rate are chosen to be 10, 10000, 15, 0.7, and 0.1, respectively in the simulation. Figure 4.25 shows the results when the initial population size is 2.

- Initial population size=2,
- Grid scale= 30,
- Grid scale= 30,
- Dimension= 2,
- Generation number= 10000,
- Age threshold= 10,
- ppv= 3,
- Run number= 50.

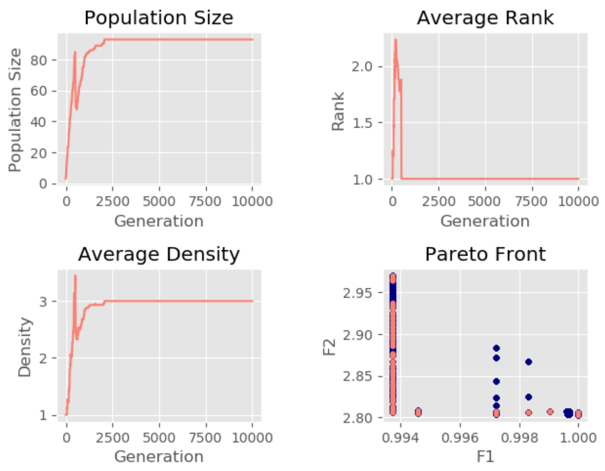
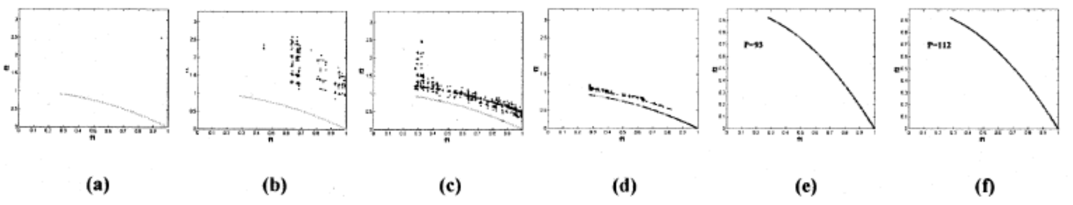


Figure 4.25 – MOP 2 Results of initial pop size=2

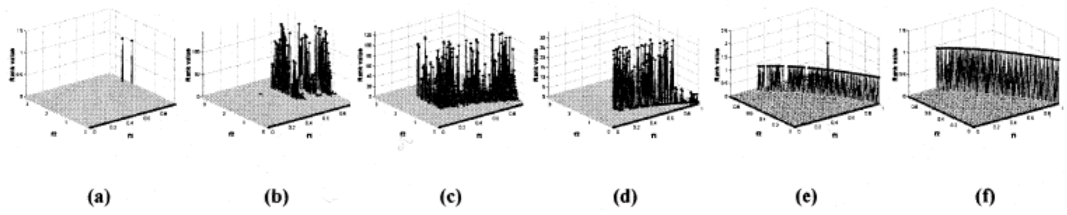
These were the results which we had obtained in our test of DMOEA, which represent the evaluation of population size, rank and density and also location in the objective space until we get the desired population size with fix average rank which equal one and a fix average density which equal *ppv* which was defined by 3 in this MOP.

Now, we will incorporate results related to paper [YL03] to compare the results of each plot without forgetting that the desired population size is an approximate number because the first creation of the initial population is always random. Figures below shows the paper [YL03] results about the First MOP.

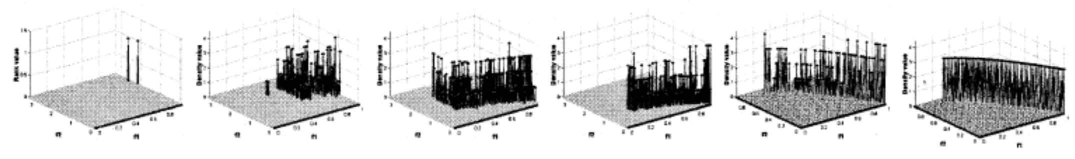
Figure 4.26 shows paper [YL03] results with 3D plots. No matter how different the results are represented, what matters to us is that all the results are similar in the endings as the rank always converges to one, density converges to the *ppv* value which was defined as 3 in the last generation and the Pareto front is showing clearly.



Snapshots of objective spaces and populations resulted from DMOEA on $F1$ at generation (a) 1, (b) 100, (c) 250, (d) 750, (e) 1300, and (f) 10 000.



Snapshots of objective spaces and rank values resulted from DMOEA on $F1$ at generation (a) 1, (b) 100, (c) 250, (d) 750, (e) 1300, and (f) 10 000.



Snapshots of objective spaces and density values resulted from DMOEA on $F1$ at generation (a) 1, (b) 100, (c) 250, (d) 750, (e) 1300, and (f) 10 000.

Figure 4.26 – MOP 2 results (From [YL03]).

Figure 4.27 represents the evolution of population size generation by generation, Figure 4.28 represents the average density.

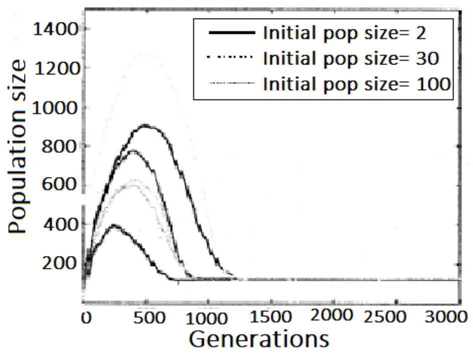


Figure 4.27 – MOP 2 population size (From [YL03]).

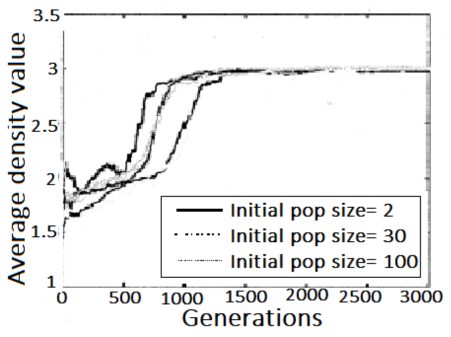


Figure 4.28 – MOP 2 density (From [YL03]).

This was the results which we had obtained and this was the results of the papers and we can observe that regardless the difference between the desired population size but the principle of the algorithm is aching to get the desired population size and Pareto front in the both problems.

4.3.2 Experimental results of PDMOEA:

Now, we will test the parallel version of DMOEA which is PDMOEA by using the same two previous problems and we will compare the results. As we did before in the last section which is the algorithm of DMOEA we will divide this section into two parts, the first part concerns the first problem related to paper [LY02] and the second concerns the second problem related to the paper [YL03].

First chosen MOP: The number of cells of each dimension proposed to be $K_1 = 50$ and $K_2 = 100$. The desired population size per cell is predefined as ppv = 5. Three specified initial population sizes 2, 30, and 100 are chosen to test the robustness of DMOEA. The age threshold, the stopping generation, the chromosome length of each decision variable, the crossover rate and the mutation rate are chosen to be 10, 8000, 15, 0.7, and 0.1, respectively in the simulation. Here we chose a different generation number which is 8000 because and as we said before that this number of generation will be divided into 4 threads which means each one of this four process will iterate into 2000 generation regardless the number of the run because this division on processes will be repeated in each run. Figure 4.29 shows the results when the initial population size is 2.

- Initial population size=2,
- Grid scale= 50,
- Grid scale= 100,
- Dimension= 2,
- Generation number= 8000,
- Age threshold= 10,
- ppv= 5,
- Run number= 30,
- Processes number= 4.

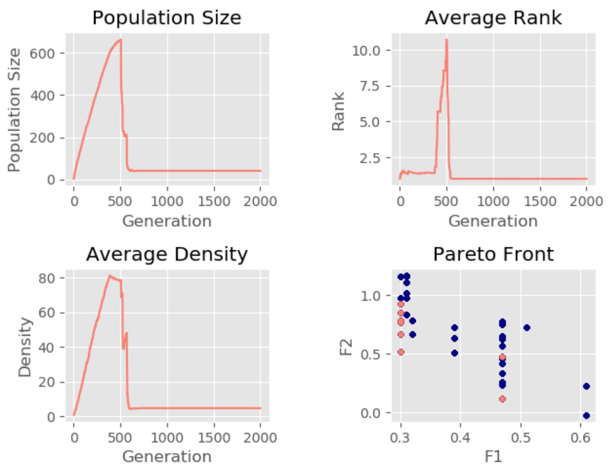


Figure 4.29 – MOP 1 Parallel Results of initial pop size=2

Figure 4.30 shows the results when the initial population size is 30.

- Initial population size=30,
- Grid scale= 50,
- Grid scale= 100,
- Dimension= 2,
- Generation number= 8000,
- Age threshold= 10,
- ppv= 5,
- Run number= 30,
- Processes number= 4.

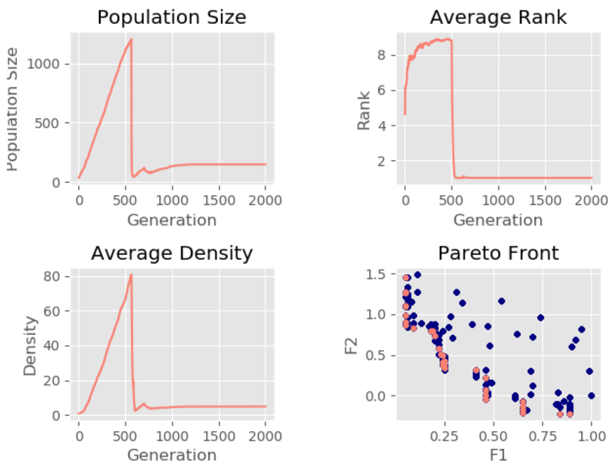


Figure 4.30 – MOP 1 Parallel Results of initial popsize=30

Figure 4.31 shows the results when the initial population size is 100.

- Initial population size=100,
- Grid scale= 50,
- Grid scale= 100,
- Dimension= 2,
- Generation number= 8000,
- Age threshold= 10,
- ppv= 5,
- Run number= 30,
- Processes number= 4.

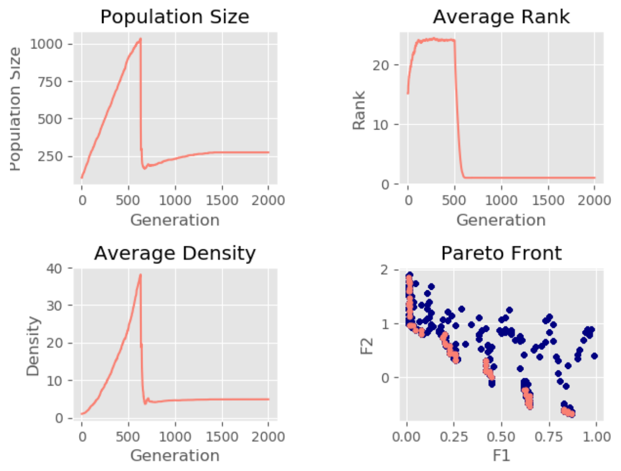


Figure 4.31 – MOP 1 Parallel Results of initial pop size=100

Second chosen MOP: The second problem is the MOP that we had presented before in the second problem of algorithm DMOEA with same inputs. PDMOEA divides the generations number of this algorithm into four threads as we said before to execute this algorithm.

The number of cells of each dimension proposed to be $K_1 = 30$ and $K_2 = 30$. The desired population size per cell is predefined as ppv = 3 . Three specified initial population sizes 2 is chosen to test the robustness of DMOEA. The age threshold, the stopping generation, the chromosome length of each decision variable, the crossover rate and the mutation rate are chosen to be 10, 10000, 15, 0.7, and 0.1, respectively in the simulation. Figure 4.32 shows the results when the initial population size is 100.

- Initial population size=2,
- Grid scale= 30,
- Grid scale= 30,
- Dimension= 2,
- Generation number= 10000,
- Age threshold= 10,
- ppv= 3,
- Run number= 50,
- Processes number= 4.

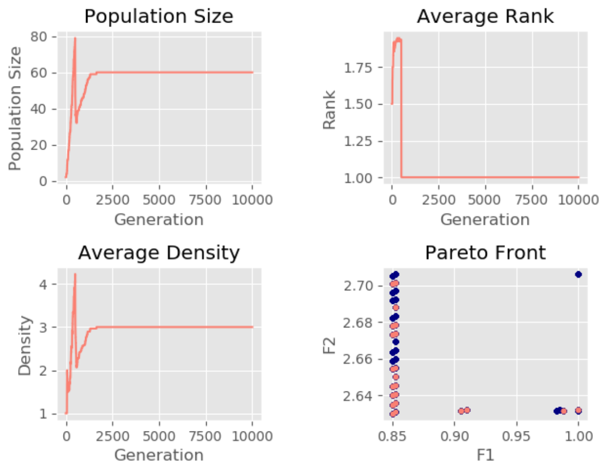


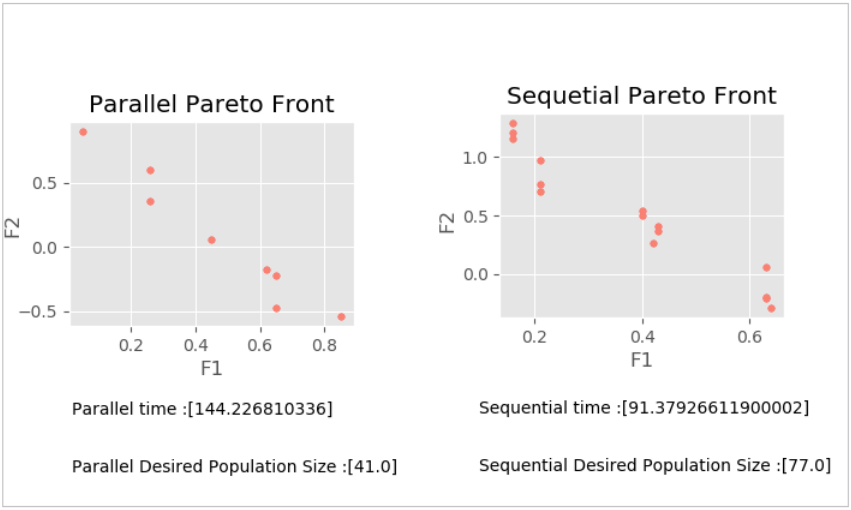
Figure 4.32 – MOP 2 Parallel Results of initial pop size=2

When we compare the results of DMOEA and PDMOEA we can observe that objective space is improved in PDMOEA better than DMOEA also time and desired population.

Comparison between DMOEA and PDMOEA:

When we come to compare DMOEA and PDMOEA using the implementation of both MOPs which we had seen before, we must think about how to compare these two algorithms by using same inputs of course. 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 and each process will take the quarter of this generation number. So what we have to do is giving a big generation number to DMOEA and PDMOEA where this division into four threads will not be adversely affected by the desired population which we will have. This number will be 8000 in the first MOP and 10000 in the second MOP.

First MOP:
Figure 4.33 shows when the initial population size is 2.



Parallel time :[144.226810336]

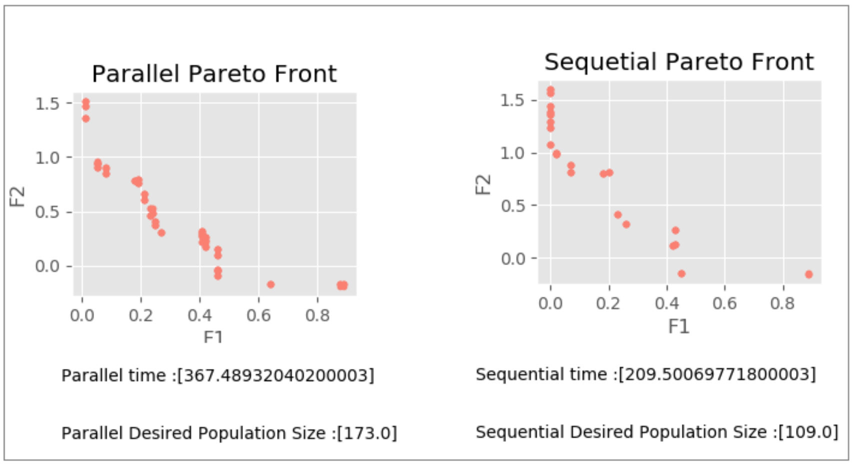
Sequential time :[91.37926611900002]

Parallel Desired Population Size :[41.0]

Sequential Desired Population Size :[77.0]

Figure 4.33 – MOP 1 comparison result pop size = 2

Figure 4.34 shows when the initial population size is 30.



Parallel time :[367.48932040200003]

Sequential time :[209.50069771800003]

Parallel Desired Population Size :[173.0]

Sequential Desired Population Size :[109.0]

Figure 4.34 – MOP 1 comparison result pop size = 30

Figure 4.35 shows when the initial population size is 100.

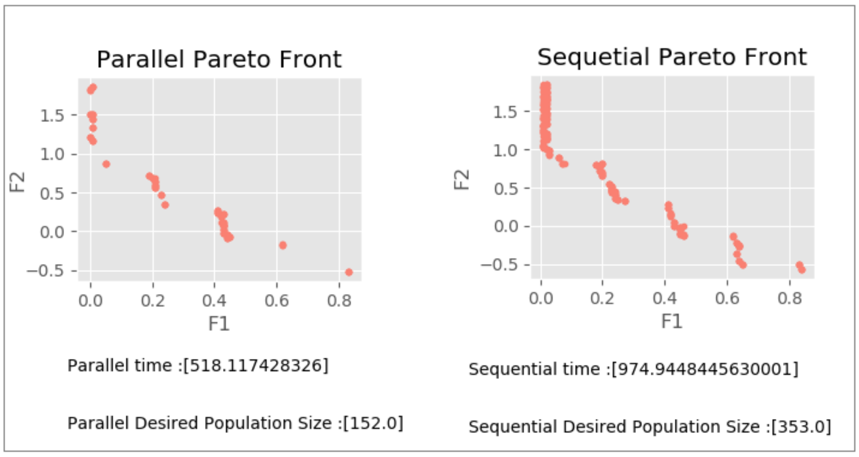


Figure 4.35 – MOP 1 comparison result pop size = 100

Second MOP:
Figure 4.36 shows when the initial population size is 2.

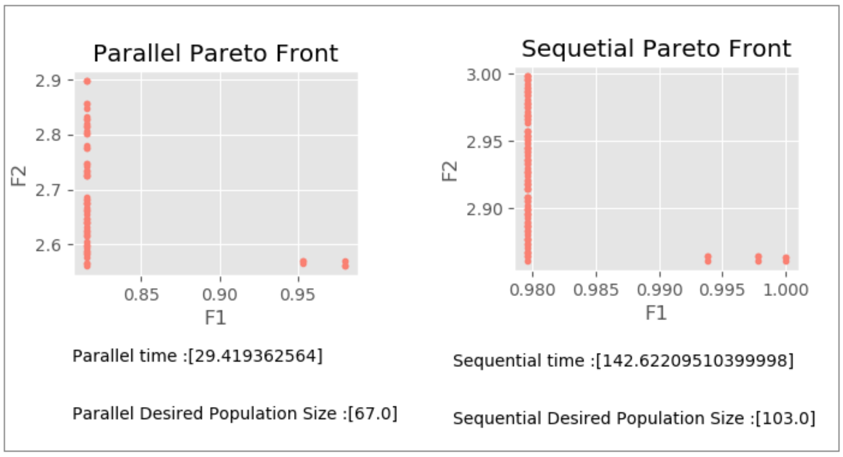


Figure 4.36 – MOP 2 comparison result.

In DMOEA, user looks for improving these main three outputs which are objective space related to the MOP even it is a maximization or minimization, desired population size and time of execution. In our implementation as we said before we now that both MOPs are a minimization problems, we can observe that in each result we can find that parallel version PDMOEA is better than DMOEA in at least one of this comparison main three outputs and this is the main reason which made us think to parallelize DMOEA.

4.4 Comparative study

4.4.1 Comparative factors

Comparative factors means indexes which we observe each time as outputs of each execution. These factors, which are in fact the results obtained every time we execute both versions we had mentioned in several sections. In this chapter, we will try to explain these factors based on the references from which we have taken MOPs before and other several references.

Desired population size

As mentioned in paper [YL03], the desired population size which called $dps(s)$, with the desired population size per unit volume ppv (a user-specified parameter), and an approximated number of trade-off hyper-areas A_{to}, discovered by the population at generation is defined as equation 4.1.

$$lowbps < dps(n) = ppv \times A_{to}(n) < upbps \tag{4.1}$$

where $lowbps$ and $upbps$ are the lower and upper bounds for the desired population size $dps(n)$, respectively. In paper [TLK99] they applied a method used to estimate the approximated number of hyper-areas as equation 4.2.

$$A_to \approx \frac{\pi^{(m-1)/2}}{(\frac{m-1}{2})!} \times (\frac{d(n)}{2})^{(m-1)} \tag{4.2}$$

where m is the number of objectives and $d(n)$ is the diameter of the hypersphere at generation n. Therefore, based on the difference between the resulting population size and estimated desired population size $dps(n)$. 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 nondominated 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 approximate and when we approach it will be better. In order for the comparison to be fair, this value is calculated in the same way in both versions in DMOEA and PDMOEA .If one of the two versions came close to this value, it would be better compared to the other version.

Improving the objective space

The past two decades have seen the development of more and more effective and efficient multi-objective optimization evolutionary algorithms (MOEAs). These methods 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 [KC07]. DMOEA main aim is to improve this objective space and make results be better than the other MOEAs. In this chapter, we will compare objective space results of DMOEA with other choosing MOEAs, they will be NSGA-II and SPEA-II using two carefully chosen test functions which we had discussed in the previous chapter with the primary results of our application.

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. In our case, we need to compare these MOEAs with two objective space defined in our choosing MOPs. As both two MOPs are 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.

Computational time

One of the main reasons for using parallel evolutionary algorithms (PEAs) is 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 [Alb02].

The effect of the Processes number

Programming distributed-memory multiprocessors and networks of workstations require deciding what can execute concurrently, how processes communicate, and where data is placed. These decisions can be made statically by a programmer or compiler, or they can be made dynamically at run time [LFA96].

In this dissertation, we adopt the previous MOPs and we propose PDMOEA to solve them with a fixed number of processes as we use in experimental study section 4.3. Now, the question that comes to our minds is how the results will be every time we increase the number of processes and which gain will be better. In this section, we work on changing the number of processes and finally compare the results. Table 4.2 shows the change in the number of operations using the same PDMOEA inputs. The results of this experiment are

	Initial population size	Grid scale 1	Grid scale 2	Dimenssion	Generation number	Age treshold	ppv	Run number	processes number
DMOEA	2	50	100	2	16000	10	5	1	-
PDMOEA	2	50	100	2	16000	10	5	1	2
PDMOEA	2	50	100	2	16000	10	5	1	4
PDMOEA	2	50	100	2	16000	10	5	1	8

Table 4.2 – Changing processes number for PDMOEA

illustrated in the Figures 4.37-4.39. Where Figure 4.37 shows the results of DMOEA vs PDMOEA when the number of processes in parallel version is 2.

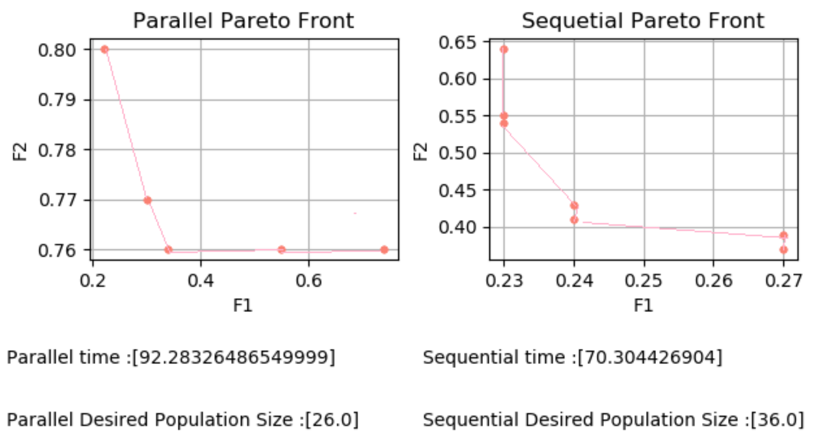
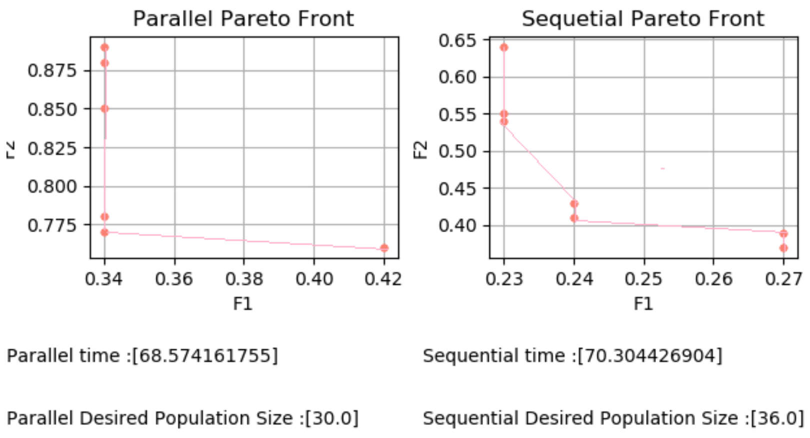


Figure 4.37 – Processes number = 2.

Figure 4.38 shows the results of DMOEA vs PDMOEA when the number of processes in parallel version is 4.



Parallel time :[68.574161755]

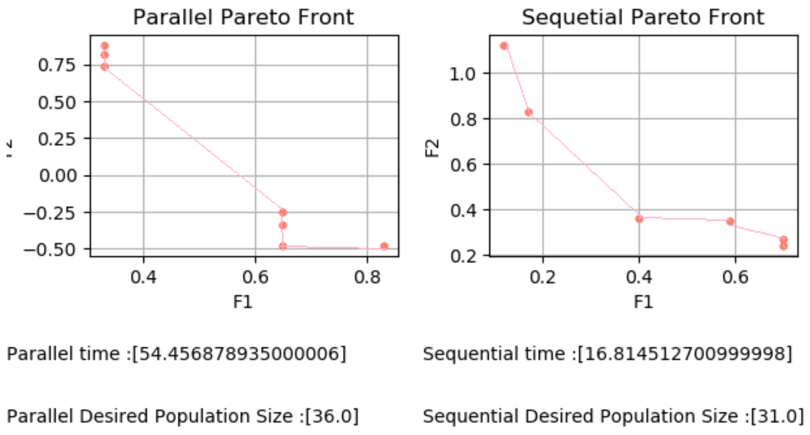
Sequential time :[70.304426904]

Parallel Desired Population Size :[30.0]

Sequential Desired Population Size :[36.0]

Figure 4.38 – Processes number = 4.

Figure 4.39 shows the results of DMOEA vs PDMOEA when the number of processes in parallel version is 8.



Parallel time :[54.456878935000006]

Sequential time :[16.814512700999998]

Parallel Desired Population Size :[36.0]

Sequential Desired Population Size :[31.0]

Figure 4.39 – Processes number = 8.

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 number of processes which makes the generation number of 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 benefit while the number of each process achieves formula 4.3

$$1000 < \frac{generation\_number}{processes\_number} \tag{4.3}$$

4.4.2 DMOEA Vs other EAs

During the time, several multi-objective optimization evolutionary algorithms (MOEA) are proposed searching for a uniformly distributed, near-optimal and near-complete Pareto front for a given MOP. Whoever this common goal is far from being accomplished by the existing MOEAs [YL03]. In this dissertation we have implemented another MOEA with the distinction of using dynamic population size. This implementation makes as think about how to prove the power of this DMOEA by several test functions.

At this section, we will compare DMOEA with other multi-objective evolutionary algorithms and we choose two of them which are NSGA-II and SPEA-II. Where Table 4.3 lists the specific parameter settings for the function of the second MOP.

	Initial pop size	External pop size	ppv	Gride Scale	Age treshold	Tournament size
DMOEA	2	-	3	(30,30)	10	-
NSGA-II	100	0*	-	-	-	2
SPEA-II	80	20	-	-	-	-

Table 4.3 – Specific parameter setting for MOP 1

Table 4.4 is dedicated to the parameters setting for all the test functions as a global parameters.

Common parameters	chromosome length	Crossover rate	maximum generation	Number of runs
DMOEA	15 $\times dec\_num$*	0.7	10000	50
NSGA-II	15 $\times dec\_num$*	0.7	10000	50
SPEA-II	15 $\times dec\_num$*	0.7	10000	50

Table 4.4 – Specific parameter setting for all the test functions

The results of this comparison are represented in Figure 4.40 represents the final objective space which had minimized and the Pareto front of NSGA-II, Figure 4.41 represents the objective space as a final result of SPEA-II and Figure 4.42 is the result of executing DMOEA based on previous setting parameters in Table 4.3 and Table 4.4.

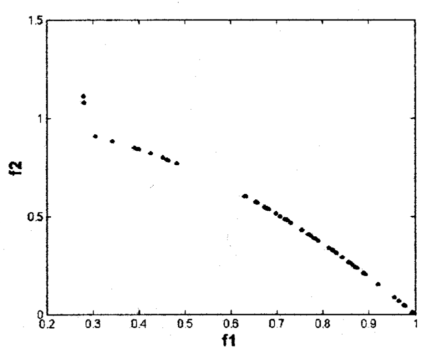


Figure 4.40 – NSGA-II plot.

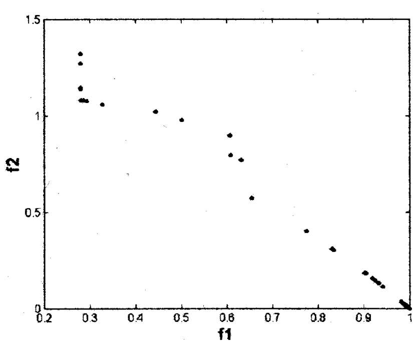
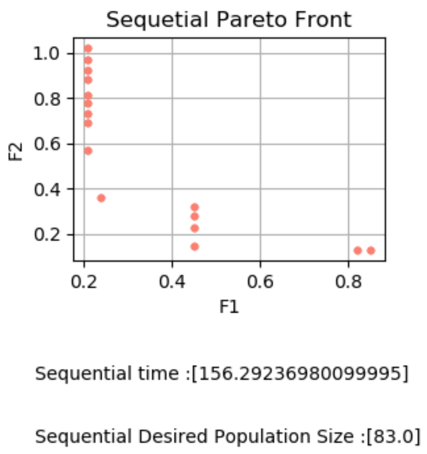


Figure 4.41 – SPEA-II plot.



```
Sequential time :[156.29236980099995]

Sequential Desired Population Size :[83.0]
```

Figure 4.42 – DMOEA plot.

These results are the Pareto fronts by two chosen MOEAs and the DMOEA Pareto front and it is clear by a visual observation that the objective space was improving in DMOEA better than the other MOEAs.

4.4.3 DMOEA Vs PDMOEA

We had compared the results of both versions and we had concluded that in each execution of this both versions using the same inputs, at least one of the three outputs which user searching to improve with PDMOEA is better than DMOEA.

To prove the power of dynamic multi-objective optimization evolutionary algorithm using parallel computing, we have to use the previous formula 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} was bigger, the PDMOEA will be better in desired population size comparison factor. Also time is considering better when it was more minimized. Table 4.5 shows the parameters which we use in both problems to make a fair comparison without forgetting that this parameters are same in both DMOEA and PDMOEA.

	Initial population size	Grid scale 1	Grid scale 2	Dimenssion	Generation number	Age treshold	ppv	Run number
Problem 1	30	50	100	2	10000	10	5	30
Problem 2	30	30	30	2	10000	10	3	50

Table 4.5 – Specific parameter setting for DMOEA and PDMOEA

First multi-objective optimization problem :

This is the results obtained after we setting this parameters of the first MOPs. The results are represented in Figure 4.43 as sub-plots collecting DMOEA and PDMOEA curves. Figure 4.44 represents the improvement of objective space results based on Table 4.5 parameters setting.

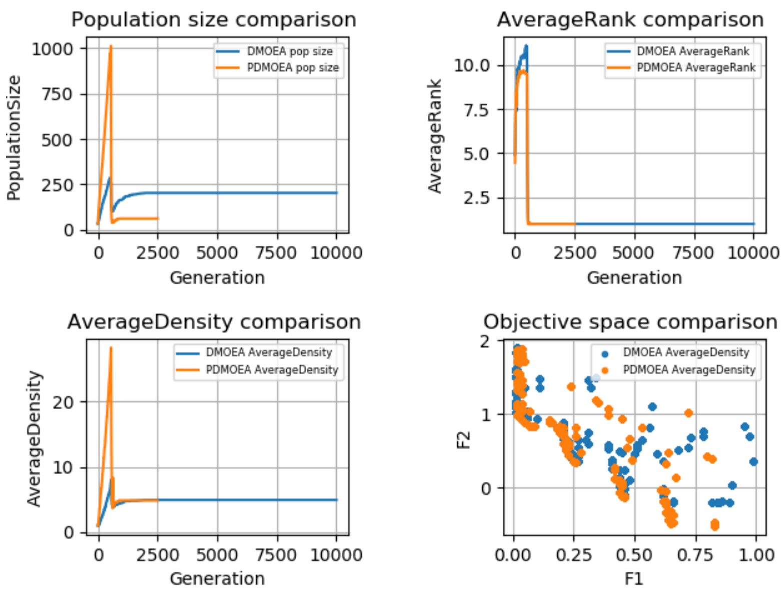


Figure 4.43 – MOP 1 comparison subplots.

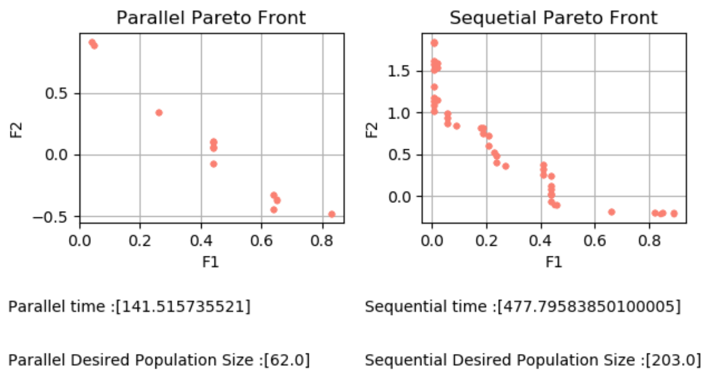


Figure 4.44 – MOP 1 objective Space comparison.

Figure 4.45 shows the Pareto front of both versions.

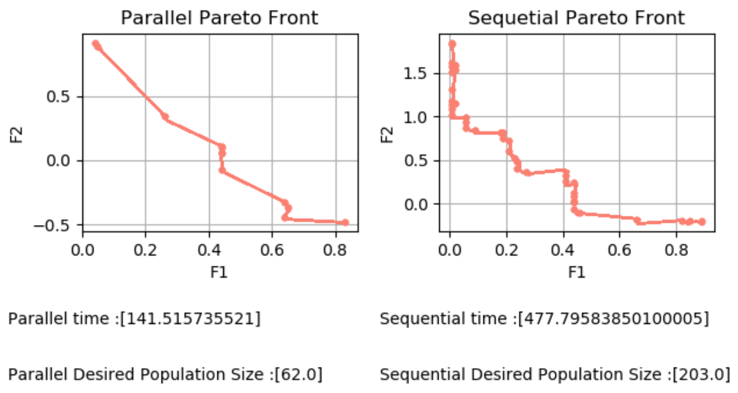


Figure 4.45 – MOP 1 Pareto front comparison.

To analyze this results we will specific two part first part is a visual observation part and the second part is calculating results.

Visual observation: As writing in the legend of each plot the blue color is for DMOEA and the orange color is for PDMOEA. The first observation we drop it on the sub-plots that is the generation number is different for the two colors and this is because PDMOEA divided the global generation number into four processes and each process execute its own parameters and finally we get the results of the best process comparing with the other rest processes. This what we had talked about in the previous chapter and that justified the variation of the generation number. While the DMOEA execute the generation number as it. Rank converges to be one, density converges to the desired population size per unit volume *ppv* and the desired population size is obtained finally.

Calculating results: In the DMOEA the desired population size $dps(n)$ is obtained as 62 that means and using the mathematical formula up, the trade-off hyper-areas A_{to} is approximately is 12 based on the desired population per unit volume *ppv* is 5, the while the desired population size $dps(n)$ obtained from PDMOEA is 203 and the trade-off hyper-areas A_{to} is approximately is 40 it means that in the desired population size of PDMOEA is better than DMOEA. however the nicest thing is that the time in the PDMOEA is better four times than DMOEA also the objective space of PDMOEA is improving more better than DMOEA.

Second multi-objective problem

This is the results obtained after we setting this parameters of the first MOPs. The results are represented in Figure 4.46 as sub-plots collecting DMOEA and PDMOEA curves. Figure 4.48 represents the improvement of objective space results based on Table 4.5 parameters setting.

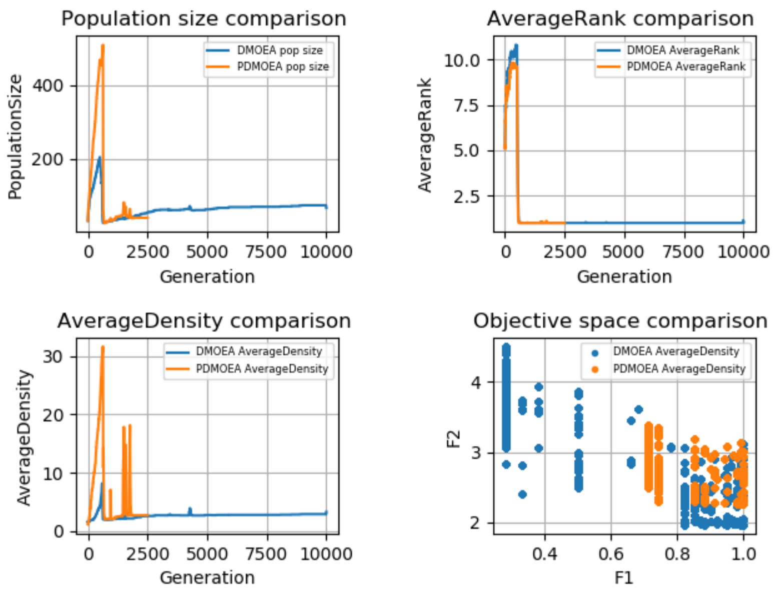
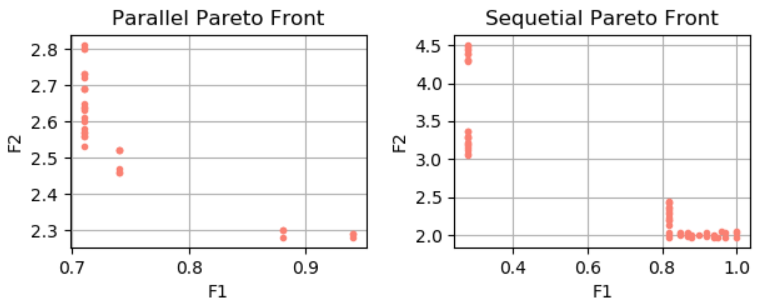


Figure 4.46 – MOP 2 comparison subplots.

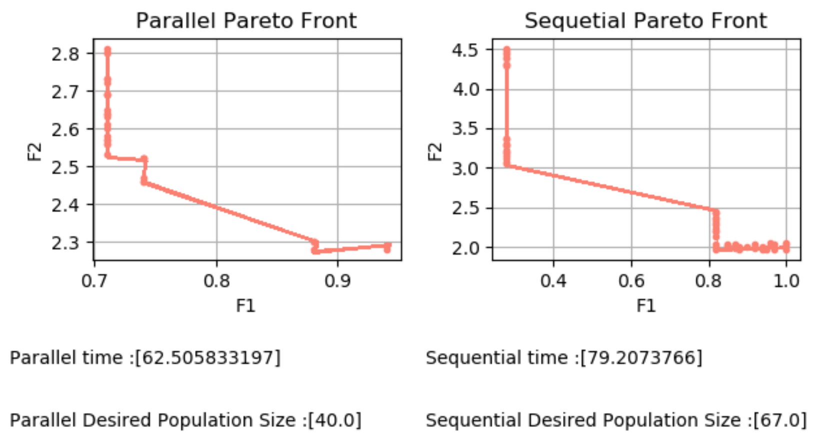


```
Parallel time :[62.505833197]                Sequential time :[79.2073766]

Parallel Desired Population Size :[40.0]     Sequential Desired Population Size :[67.0]
```

Figure 4.47 – MOP 2 objective Space comparison.

Figure **??** shows the Pareto front of both versions.



Parallel time :[62.505833197]

Sequential time :[79.2073766]

Parallel Desired Population Size :[40.0]

Sequential Desired Population Size :[67.0]

Figure 4.48 – MOP 2 Pareto front comparison.

As we said before to analyze this results we will specific two part first part is a visual observation and the second part is calculating results.

Visual observation: As writing in the legend of each plot the blue color is for DMOEA and the orange color is for 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.

Calculating results: In the DMOEA the desired population size $dps(n)$ is obtained as 40 that means and using the mathematical formula up, the trade-off hyper-areas A_{to} is approximately is 13 based on the desired population per unit volume *ppv* is 5, the while the desired population size $dps(n)$ obtained from PDMOEA is 67 and the trade-off hyper-areas A_{to} is approximately is 22 it means that in the desired population size there is no big difference between PDMOEA and DMOEA. But the nicest thing is that the time in the PDMOEA is better four times than DMOEA also the objective space of PDMOEA is improving more better than DMOEA.

Conclusion

In this chapter, we have presented the tools that we have used to release our application, then the results of our implementation as a set of graphical user interfaces (GUIs) and plots. The last section discussed the results of running the implemented algorithm with various parameters and prove the efficiency of our parallelized algorithm (i.e.,PDMOEA).

General Conclusion

General Conclusion

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. For that reason, parallelizing MOEAs is an important issue. Since we are looking for a Pareto-optimal solutions with different trade-offs between the objectives, it seems natural to assign different parts of the search space to different processors. Also minimizing the computational time is a main reason to parallelize MOEAs. Users are always looking for the best results in less time, these reasons made us think about the creation of PDMOEA.

During the project realization, we have learned knowledge about:

- Multi-objective optimization problems,
- Multi-objective optimization evolutionary algorithms (MOEAs),
- Dynamic multi-objective evolutionary algorithm (DMOEA),
- Parallel computing, including parallel terminologies, parallel architectures and parallel programming models,
- Parallelization of DMOEA (i.e., PDMOEA).

We have implemented a dynamic multi-objective optimization evolutionary algorithm (DMOEA) on the same test functions proposed in [YL03] and [LY02] 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 enhances results more and more in term of minimizing computational time, improving objective space and converging to the desired population size. This reason made as parallelize the DMOEA and implement the PDMOEA. We have implemented these algorithms using Python programming language. Our application is useful for any test function. Worth mentioning in this dissertation is that the proposed PDMOEA represents the first parallel version of DMOEA in literature. PDMOEA is more efficient than DMOEA in terms of three factors: improving objective space, minimizing computational time and converging to the desired population size. These three criteria are required by the user.

In our future research we intend to concentrate on addressing other questions which remain to resolve, some of which are:

- Use a case study selected from the real world,
- Turn the problem in many-objective optimization problem by achieving other objectives (more than two objectives as using in test functions),
- Implement other MOEAs (their sequential as well as their parallel versions) and compare them with DMOEA and PDMOEA to prove the efficiency of the proposed PDMOEA.

Bibliography

[19a] *Wikipedia*. June 21, 2019. URL: `https://bit.ly/2X5ocHH`.

[Von93] John Von Neumann. “First Draft of a Report on the EDVAC”. In: *IEEE Annals of the History of Computing* 15.4 (1993), pp. 27–75.

[LFA96] David K Lowenthal, Vincent W Freeh and Gregory R Andrews. “Using fine-grain threads and run-time decision making in parallel computing”. In: *Journal of Parallel and Distributed Computing* 37.1 (1996), pp. 41–54.

[Mit96] Melanie Mitchell. “An introduction to genetic algorithms mit press”. In: *Cambridge, Massachusetts. London, England* (1996).

[Hus98] Talib S Hussain. “An introduction to evolutionary computation”. In: *1998 CITO Researcher Retreat* (1998), pp. 12–14.

[Jon98] Gareth Jones. “Genetic and evolutionary algorithms”. In: *Encyclopedia of Computational Chemistry* 2 (1998), pp. 1127–1136.

[TLK99] KC Tan, Tong H Lee and EF Khor. “Evolutionary algorithms with goal and priority information for multi-objective optimization”. In: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406).* Vol. 1. IEEE. 1999, pp. 106–113.

[TLK01] Kay Chen Tan, Tong Heng Lee and Eik Fun Khor. “Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization”. In: *IEEE Transactions on Evolutionary Computation* 5.6 (2001), pp. 565–588.

[Alb02] Enrique Alba. “Parallel evolutionary algorithms can achieve super-linear performance”. In: *Information Processing Letters* 82.1 (2002), pp. 7–13.

[AT02] Enrique Alba and Marco Tomassini. “Parallelism and evolutionary algorithms”. In: *IEEE transactions on evolutionary computation* 6.5 (2002), pp. 443–462.

[Kum02] Vipin Kumar. *Introduction to parallel computing.* Addison-Wesley Longman Publishing Co., Inc., 2002.

[LY02] Haiming Lu and Gary G Yen. “Dynamic population size in multiobjective evolutionary algorithms”. In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC’02 (Cat. No. 02TH8600).* Vol. 2. IEEE. 2002, pp. 1648–1653.

[Don+03] Jack Dongarra et al. *Sourcebook of parallel computing.* Vol. 3003. Morgan Kaufmann Publishers San Francisco, 2003.

[YL03] Gary G Yen and Haiming Lu. “Dynamic multiobjective evolutionary algorithm: adaptive cell-based rank and density estimation”. In: *IEEE Transactions on Evolutionary Computation* 7.3 (2003), pp. 253–274.

[Gan+04] Xavier Gandibleux et al. *Meta-heuristics for multi-objective optimisation.* 2004.

[05] *An Introduction to Tkinter.* `http://effbot.org/tkinterbook/`. accessed on May 18, 2017. 2005.

[SUZ05] Felix Streichert, Holger Ulmer and Andreas Zell. “Parallelization of multi-objective evolutionary algorithms using clustering algorithms”. In: *International Conference on Evolutionary Multi-Criterion Optimization*. Springer. 2005, pp. 92–107.

[KC07] Joshua Knowles and David Corne. “Quantifying the effects of objective space dimension in evolutionary multiobjective optimization”. In: *International Conference on Evolutionary Multi-Criterion Optimization*. Springer. 2007, pp. 757–771.

[Gro09] William Gropp. “Tutorial on mpi: The message-passing interface”. In: *Mathematics and Computer Science Division Argonne National Laboratory Argonne, IL* 60439 (2009).

[DPM14] Daniele DAgostino, Giulia Pasquale and Ivan Merelli. “A Fine-Grained CUDA Implementation of the Multi-objective Evolutionary Approach NSGA-II Potential Impact for Computational and Systems Biology Applications”. In: *International Meeting on Computational Intelligence Methods for Bioinformatics and Biostatistics*. Springer. 2014, pp. 273–284.

[Mar14] Igor L Markov. “Limits on fundamental limits to computation”. In: *Nature* 512.7513 (2014), p. 147.

[SV14] Sergio Santander-Jiménez and Miguel A Vega-Rodrıéguez. “Applying OpenMP-based parallel implementations of NSGA-II and SPEA2 to study phylogenetic relationships”. In: *2014 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2014, pp. 305–313.

[GVR17] David L González-Álvarez, Miguel A Vega-Rodrıéguez and Álvaro Rubio-Largo. “A hybrid MPI/OpenMP parallel implementation of NSGA-II for finding patterns in protein sequences”. In: *The Journal of Supercomputing* 73.6 (2017), pp. 2285–2312.

[KGG17] Yoram Koren, Xi Gu and Weihong Guo. “Reconfigurable manufacturing systems: Principles, design, and future trends”. In: *Frontiers of Mechanical Engineering* (Nov. 2017). DOI: `10.1007/s11465-018-0483-0`.

[17a] *L’éditeur LaTeX universel*. `http://www.xm1math.net/texmaker/index_fr.html`. accessed on May 18, 2017. 2017.

[Lam17] Leslie Lamport. *LaTeX*. `https://en.wikipedia.org/wiki/LaTeX`. accessed on May 18, 2017. 2017.

[17b] *matplotlib*. `http://matplotlib.org/`. accessed on May 18, 2017. 2017.

[18a] *Introduction to Parallel Computing*. Mar. 2018. URL: `https://docs.huihoo.com/hpc-cluster/parallel-computing/index.html`.

[18b] *Introduction to Parallel Computing*. Mar. 2018. URL: `https://computing.llnl.gov/tutorials/parallel_comp/`.

[19b] *Flowchart Maker & Online Diagram Software*. June 2019. URL: `https://www.draw.io/`.

[19c] *multiprocessing Process based threading interface Python documentation*. May 2019. URL: `https://docs.python.org/2/library/multiprocessing.html`.