Journal of Engineering Managment and Soft Computing (JEMSC)

Solving Multi-objective Optimization Problems Using the Society Deciling Process Algorithm

Aylar Poorahad[1], Einollah Pira[2]✉, Mohammad Khodizadeh-Nahari[3] and Sajad Esfandyari[4]

1. Faculty of Information Technology and Computer Engineering, Azarbaijan Shahid Madani University, Tabriz, Iran. Email: aylar.p.2000@gmail.com
2. Corresponding author, Faculty of Information Technology and Computer Engineering, Azarbaijan Shahid Madani University, Tabriz, Iran. Email: pira@azaruniv.ac.ir
3. Faculty of Information Technology and Computer Engineering, Azarbaijan Shahid Madani University, Tabriz, Iran. Email: m.khodizadeh@azaruniv.ac.ir
4. Department of Computer Engineering, Malayer University, Malayer, Iran. Email: sajad.a1367@gmail.com

Article Info

Article type:
Research Article

Article history:
Received 22 August 2025
Received in revised form 4 February 2026
Accepted 13 March 2026
Published online 1 April 2026

Keywords:
metaheuristic algorithms, optimization, multi-objective, society deciling process, Pareto front.

ABSTRACT

Advancements in technology and the emergence of multi-objective optimization problems across various scientific domains have spurred research and development of novel metaheuristic algorithms to address these challenges. Although these methods have largely succeeded in approaching the Pareto-optimal front, the optimization process has not been fully realized. This paper introduces a multi-objective version of the Social Division Process (SDP) algorithm, termed MOSDP, aimed at improving the quality of Pareto front solutions. The MOSDP algorithm employs a memory structure as an archive to store non-dominated solutions. Additionally, it utilizes a non-dominated sorting mechanism based on crowding distance to establish a hierarchical social division structure and guide the evolutionary process in the multi-objective problem space. The performance of MOSDP is evaluated using 18 well-known multi-objective test functions, UF, and IMOP, and compared with the Multi-Objective City Councils Evolution (MOCCE), Multi-Objective Ant Lion Optimization (MOALO), Multi-Objective Slime Mould Algorithm (MOSMA), and Multi-Objective Artificial Hummingbird Algorithm (MOAHA). The results of the Friedman average rank test demonstrate the superiority of MOSDP over the aforementioned algorithms in terms of Inverted Generational Distance (IGD), Generational Distance (GD), and Maximum Spread (MS) metrics.

Cite this article: Pira, P. & et al, (2026)., Solving Multi-objective Optimization Problems Using the Society Deciling Process Algorithm. *Journal of Engineering Management and Soft Computing*, 12 (2). 75-94.

DOI: https://doi.org/10.22091/jemsc.2026.13697.1297

Publisher: University of Qom
DOI: https://doi.org/10.22091/jemsc.2026.13697.1297

1) Introduction

Optimization is the act of selecting the best solution from a set of possible options for a specific problem (Khishe et al., 2023). In the world of engineering, finding the best solution for design problems has always been a fundamental challenge. Metaheuristic approaches have been effective for various optimization objectives due to their high accuracy, optimization speed, and low computational complexity (Połap & Woźniak, 2021). Traditional methods, which relied on repetitive testing and the construction of numerous prototypes, were not only costly and time-consuming but also lacked the necessary efficiency due to human errors. In the real world, many optimization problems involve multiple conflicting objectives, such as cost versus quality, speed versus accuracy. In situations where several competing objectives exist, the problem is referred to as a multi-objective optimization problem. Multi-objective optimization is distinguished from single-objective optimization by the challenge of dealing with conflicting objectives and finding Pareto optimal solutions (the Pareto front set). In some cases, an ideal and perfectly optimal solution may not be achievable at all; in such situations, the need for compromise and the use of practical estimates becomes apparent.

Nowadays, metaheuristic algorithms have emerged as intelligent and automated solutions to these challenges. By providing automated computational solutions, metaheuristic algorithms have brought about a fundamental transformation in engineering design processes. These algorithms are employed in situations where the problem is computationally hard (NP-hard) and solving it, using methods such as linear and dynamic programming, is time-consuming; where the solution space is large and classical methods become ineffective; or when multiple conflicting objectives exist. There are two ultimate objectives in applying metaheuristic algorithms to obtain Pareto-optimal solutions: (1) Achieving Pareto-optimal solutions with a uniform and realistic distribution (convergence), and (2) Ensuring complete coverage of the search space and obtaining a homogeneous set of solutions (coverage) (Połap & Woźniak, 2021). Although multi-objective metaheuristic algorithms have so far been able to approximate the Pareto-optimal front to an acceptable extent, the optimization process has not yet been fully realized.

In this paper, a multi-objective version of the Society Deciling Process (SDP) algorithm, called MOSDP, is proposed. The main innovation of this method lies in integrating the hierarchical decile-based structure with a non-dominated sorting mechanism and the crowding-distance criterion. This integration enables the algorithm to simultaneously maintain convergence toward the Pareto front and diversity in the solution space. Unlike similar algorithms, such as MOCCE and MOALO, which require the tuning of multiple parameters, the MOSDP algorithm is designed with a minimal number of parameters and offers high stability and simplicity in implementation.

2) Literature Review

Optimization algorithms are generally inspired by the natural behavior of an agent, which may be human, animal, plant, or a physical or chemical phenomenon (Naruei & Keynia, 2021). Metaheuristic algorithms are intelligent methods designed to solve complex, nonlinear, multimodal, and NP-hard optimization problems. These algorithms are typically inspired by nature, physics, mathematics, or collective behaviors and, compared with exact methods, provide approximate yet high-quality solutions within a reasonable computational time. Metaheuristic algorithms can be categorized into several classes with different operational mechanisms. In this section, multi-objective optimization–related algorithms are reviewed in three main groups—evolutionary algorithms, physics- and mathematics-based algorithms, and population-based algorithms—in order to clarify the position of the proposed MOSDP method among them.

Evolutionary algorithms are a class of metaheuristic algorithms inspired by Darwin's principles of natural evolution. These algorithms solve complex optimization problems by simulating natural selection, genetic crossover, and mutation processes. One of the most well-known evolutionary algorithms is the Multi-Objective Genetic Algorithm (MOGA) (Murata & Ishibuchi, 1995). This algorithm operates based on the framework of the Genetic Algorithm (GA) (Holland, 1992) but incorporates specific mechanisms for handling multiple objectives simultaneously. Another prominent

algorithm in this category is NSGA-II (Deb et al., 2000b). This algorithm ranks solutions using non-dominated sorting, divides them into different fronts, and assigns ranks based on fitness values. It also employs the crowding-distance criterion to preserve diversity within each front. Another evolutionary algorithm is SPEA2 (Zitzler et al., 2001), in which non-dominated solutions are first stored in a separate archive. Then, density estimation is performed using the k-nearest neighbor method, and finally, a combination of the main population and the archive is selected. Other evolutionary algorithms include MOEA/D (Zhang & Li, 2007), NPGA (Horn et al., 1994), PDE (Abbass et al., 2001), MOMBI (Gómez & Coello, 2013), DOMOEA (Zeng et al., 2006), and IBEA (Zitzler & Künzli, 2004).

The next category includes physics- and mathematics-based algorithms. These algorithms utilize physical laws or mathematical concepts to guide the optimization search process. One well-known physics-based algorithm is MOEA (Premkumar et al., 2022), which is an improved multi-objective version of the EA algorithm (Faramarzi et al., 2020). Another algorithm in this category is MOWCA (Sadollah et al., 2015), which is inspired by the natural water cycle process. This algorithm mimics the flow of rivers and streams toward the sea, derived from observations of the hydrological cycle in nature. To select the most efficient (best) solutions in the population as seas and rivers, the crowding-distance mechanism is employed. This parameter represents the distribution of non-dominated solutions around a particular non-dominated solution. A smaller crowding-distance value indicates a higher concentration of solutions in a specific region. In multi-objective problems, this parameter is computed in the objective space. Therefore, to calculate this parameter for each non-dominated solution, all non-dominated solutions must first be sorted according to one of the objective functions. The resulting non-dominated solutions are then designated as seas and rivers, and the flow intensity for rivers and seas is calculated based on crowding-distance values. Under these conditions, non-dominated solutions are more likely to be generated around seas and rivers in the next iteration, and their crowding-distance values are adjusted and reduced. Another well-known physics-based algorithm is MOGBO (Premkumar et al., 2021). This algorithm employs two operators—local escaping and gradient search laws—along with multiple vector sets during the search phase. An elitist non-dominated sorting mechanism is used to rank agents and identify Pareto-optimal solutions. This algorithm is an a posteriori approach and utilizes the traditional crowding-distance mechanism to ensure adequate coverage of the best solutions for the given objectives (Premkumar et al., 2021).

Population-based algorithms form another class of optimization methods that operate on a set of potential solutions (a population) rather than a single solution. Due to their strong capability in solving complex real-world problems, population-based algorithms are widely used across various scientific and industrial fields. The advantages of these algorithms include better exploration of the search space through simultaneous consideration of multiple solutions, reduced likelihood of being trapped in local optima, parallelizability due to independent solution evaluations, and high flexibility in adapting to different problem types. The optimization process typically begins with generating a set of random solutions. These initial solutions are then combined, moved, or evolved over a predefined number of steps, referred to as iterations or generations. This framework forms the basis of nearly all population-based algorithms (Mirjalili, Mirjalili, et al., 2016). One of the most well-known algorithms in this group is the Multi-Objective Particle Swarm Optimization (MOPSO) algorithm (Coello & Lechuga, 2002), which is an extended version of the classical PSO (Kennedy & Eberhart, 1995) for multi-objective problems.

The Multi-Objective Ant Colony Optimization (MOACO) algorithm (Alaya et al., 2007) is inspired by the intelligent behavior of ants in finding the shortest paths to food sources. This algorithm is an extended version of the ACO algorithm (Dorigo et al., 2006). Initially, the pheromone matrix is generated and parameters are initialized. Each ant constructs a solution; all solutions are evaluated using the objective functions, and non-dominated solutions are identified. Step selection is performed based on probabilistic rules. In the next phase, pheromones are updated such that good solutions (paths) are reinforced, while pheromones on poor paths evaporate and diminish. Additionally, archive management is required, where newly found non-dominated solutions are added and dominated ones are removed. This process continues until the stopping criterion is satisfied.

The Multi-Objective City Council Evolution (MOCCE) algorithm (Ghaffar Alishahi et al., 2023) is designed based on the indirect election system used in democratic societies. In this method, citizens elect local council members, and the heads of these councils form a higher-level council. This hierarchical process continues from the smallest neighborhoods to the city level. The criteria for selecting council leaders include popularity, education level, executive experience, and other performance indicators. All council members strive to improve these criteria to increase their chances of being selected in subsequent rounds. The single-objective version of this algorithm is the City Council Evolution (CCE) algorithm (Pira, 2022), which is a metaheuristic inspired by the process of forming a city's supreme council. In the CCE algorithm, solutions of the current population are maintained in a council tree structure organized as a max-heap.

Other algorithms in this group include the Multi-Objective Artificial Bee Colony (MOABC) (Akbari et al., 2012), Multi-Objective Grey Wolf Optimizer (MOGWO) (Mirjalili, Saremi, et al., 2016), Multi-Objective Ant Lion Optimizer (MOALO) (Mirjalili et al., 2017), Multi-Objective Slime Mould Algorithm (MOSMA) (Premkumar et al., 2020), Multi-Objective Artificial Hummingbird Algorithm (MOAHA) (Khodadadi et al., 2022), and Multi-Objective Emperor Penguin Optimizer (MOEPO) (Kaur et al., 2020). The conducted review indicates that, despite the significant progress achieved in multi-objective algorithms, none of them exploit a hierarchical structure similar to societal decile-based stratification. This research gap constitutes the primary motivation for developing the MOSDP algorithm in this study.

3) Methodology

3-1) Basic Definitions

In this section, fundamental concepts, such as multi-objective optimization, as well as the definitions relevant to this research, are presented.

3-1-1) Multi-Objective Optimization

Multi-objective optimization is one of the important branches of optimization that seeks to find solutions to problems in which multiple conflicting objectives must be optimized simultaneously. Unlike single-objective optimization, which has a single optimal solution, multi-objective optimization typically results in a set of optimal solutions (commonly referred to as Pareto solutions), none of which is strictly superior to the others.

A multi-objective optimization problem is defined in Equation (1):

$$\begin{cases} Minimize \quad F(x) = [f_1(x), f_2(x), \dots, f_k(x)]^T \\ Subject\ to \quad g_i(x) \le 0, \quad i = 1,2, \dots, p \\ \qquad\qquad h_j(x) = 0, \quad j = 1,2, \dots, p \\ \qquad\qquad\qquad x \in \mathbb{R}^n \end{cases} \tag{1}$$

where x is the decision vector (optimization variables), $f_1, f_2, \dots, f_k$ are the objective functions to be minimized, $g_i(x)$ represent inequality constraints, and $h_j(x)$ denote equality constraints.

3-1-2) Pareto Dominance

In multi-objective optimization problems, multiple objective functions are usually involved, which may conflict with one another (improving one objective may lead to the degradation of another). Under such circumstances, instead of a single optimal solution, a set of optimal solutions—known as the Pareto front—exists. Pareto dominance is a criterion that enables the comparison of these solutions.

A solution x_1 is said to dominate solution x_2 if:

1. It is better than or equal to x_2 in all objectives (as expressed in Equation (2)):

$$\forall i \in \{1,2, \dots, k\}: f_i(x_1) \le f_i(x_2) \tag{2}$$

2. It is strictly better than x_2 in at least one objective (as expressed in Equation (3)):

$$\exists j \in \{1,2,\dots,k\}: f_j(x_1) < f_j(x_2) \quad (3)$$

Pareto dominance is a tool for comparing solutions in the multi-dimensional objective space. A solution is non-dominated if no other solution dominates it.

3-1-3) Pareto Front

The Pareto front is the set of the best possible solutions in a multi-objective optimization problem, such that none of these solutions dominates any other. In other words, it is the set of non-dominated solutions for which no other solution exists that dominates them (as expressed in Equation (4)):

$$PF = \{ F(x) \mid \nexists x' \text{ s.t. } x' \prec x \} \quad (4)$$

In single-objective problems, the Pareto front consists of a single point, whereas in multi-objective problems it forms a curve or surface. The existence of a set of non-dominated solutions allows the decision-maker to choose among different options, each with its own advantages and disadvantages, according to the objectives.

3-1-4) Crowding Distance

Crowding distance is a metric used to measure the density of solutions in the objective space. In multi-objective algorithms, it is employed to maintain diversity on the Pareto front, prevent clustering of SOLUTIONS in specific regions, and select well-spread solutions during the optimization process.

For each solution i in a non-dominated set, the crowding distance is calculated in Equation 5:

$$CD_i = \sum_{j=1}^{k} \frac{f_j(i+1) - f_j(i-1)}{f_j^{\max} - f_j^{\min}} \quad (5)$$

where k is the number of objective functions, $f_j(i+1)$ and $f_j(i-1)$ are the objective j values of the neighboring solutions, and $f_j^{\max}$ and $f_j^{\min}$ are the maximum and minimum values of objective j in the population.

The steps to calculate the crowding distance are as follows: first, solutions are sorted based on each objective function; next, the boundary values are assigned; and finally, the distances for the intermediate solutions are computed. This metric helps the algorithm generate a diverse set of solutions across the Pareto front rather than concentrating in a particular region. Figure 1 illustrates the crowding-distance approach.

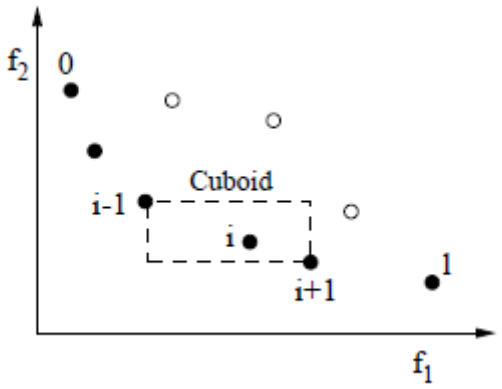


Figure 1. The Crowding Distance Diagram (Deb et al., 2000b)

3-1-5) Society Deciling Process Algorithm

Before presenting the multi-objective version of the SDP algorithm (Pira & Rouhi, 2024), we first provide a brief overview of how this algorithm works. The SDP algorithm is a metaheuristic algorithm inspired by the process of dividing a society into income deciles. By simulating economic and social changes, this algorithm divides the population into different deciles and then uses this classification to search for optimal solutions. In this algorithm, initially all individuals are placed in decile 1. Over time, some individuals become wealthier, and the society is divided into two deciles (1 and 2). This process continues until the society is divided into 10 deciles (Figure 2). Similar to other metaheuristic methods, the algorithm starts with a random initial population and then improves the solutions by evaluating optimization criteria.

The steps of this algorithm are as follows:

A) Initial Population Generation: The initial population consists of N vectors, where each vector represents an individual in the population. Each vector contains m variables representing the decile classification criteria (such as income, education, occupation). The variables are initialized randomly (according to Equation 6):

$$x_{i,j} = l_j + rand \times (u_j - l_j) \quad (6)$$

where $l_j$and $u_j$are the lower and upper bounds of variable j, respectively, and rand is a random number in the range [0,1].

B) Fitness Calculation: For each individual x_i, the fitness value is calculated using the objective function f(according to Equation 7):

$$fitness = f(X_i) = f(x_{i,1}, x_{i,2}, \dots, x_{i,m}) \quad (7)$$

C) Dividing the Population into Deciles: Initially, all individuals are placed in decile 1. After calculating fitness, individuals are sorted based on their fitness values, and the population is divided into two deciles:

- Decile 2: Half of the solutions with the highest fitness
- Decile 1: The remaining solutions with the lowest fitness

The next step is calculating the number of internal iterations using the formula: $diter = \frac{MaxIter}{9}$. MaxIter is the total number of algorithm iterations, which is divided into 9 parts since, in subsequent iterations, d increases from 2 to 10. This aims to control the number of iterations for each division level. This process continues until the population is divided into 10 deciles.

Then, the size of each section is determined using: $decN = \frac{N}{d}$, which represents the number of individuals in each section.

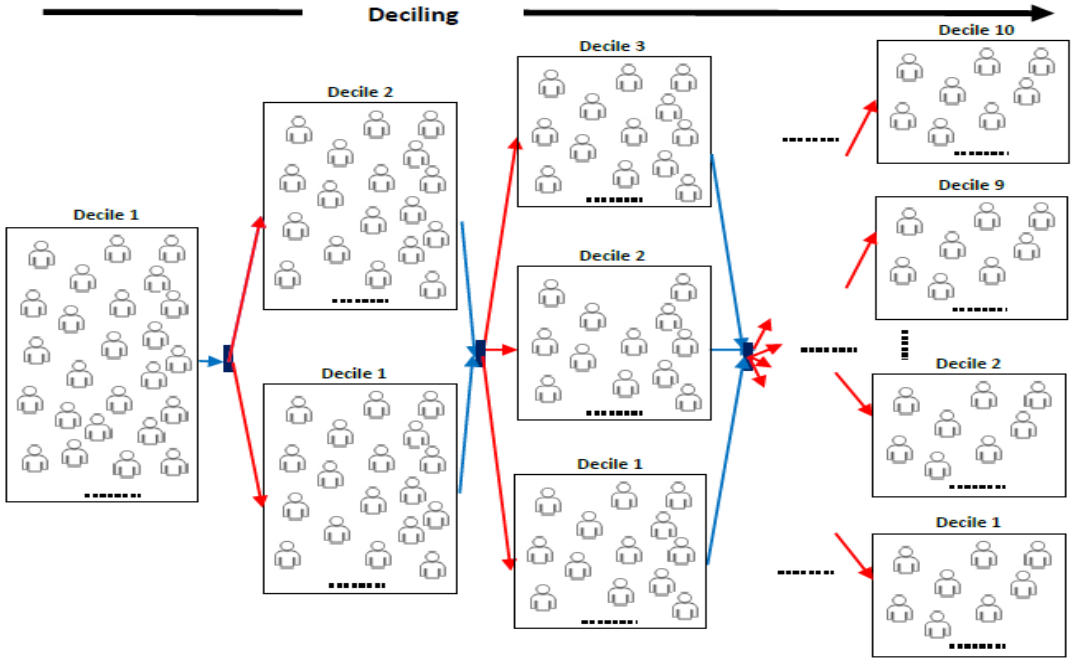


Figure 2. The Process of Dividing the Whole Society into Deciles (Pira & Rouhi, 2024)

D) Repositioning Solutions: In each d-decile state (when the population is divided into $d$deciles), the algorithm repeats the following steps:

1. Sort the population based on the fitness function.
2. Divide the population into d sections (each representing one decile).
3. Update the positions of the solutions using the repose function.

The repose function operates as follows: For each solution in decile k, a weighted average (wAvg) of the best solutions from the higher deciles is calculated (according to Equation 8):

$$wAvg = \frac{\sum_{j=k+1}^{d}(j-k)\ B_j}{\sum_{j=k+1}^{d}(j-k)} \tag{8}$$

Where $B_j$is the best solution in decile j. Then, the new position of the solution is calculated by combining wAvg with the best solution of the current decile (B_k).

The repose function is one of the key components of the SDP algorithm, responsible for updating solution positions based on the best individuals in higher deciles. By intelligently combining information from different deciles, it generates new solutions that move toward the optimal answer.

3-2) The Proposed Algorithm

The proposed algorithm is based on the society deciling process (SDP) algorithm and is termed MOSDP. In the SDP algorithm, chromosomes (solutions) are maintained in a deciling herarchical structure (DHS). This algorithm deals with a single objective function, where the value of the objective function represents the fitness of the solutions. Therefore, we could easily transform it into a declining hierarchical structure. However, during the implementation and evaluation of the SDP algorithm, the considered test functions were minimization optimizations, meaning solutions with the lowest fitness values occupy higher levels. The most critical issue in the MOSDP algorithm is how to create the DHS structure, since each solution corresponds to multiple objective function values. For sorting the solutions, we use the non-dominated sorting method employed in the NSGA-II algorithm (Deb et al., 2000a).

In the non-dominated sorting method, all solutions are initially compared with each other to determine which ones dominate the others. A solution dominates another if it is better in at least one objective

function and not worse in any of the others. After the comparisons, the solutions that are not dominated by others are considered as the first front (Pareto front). These solutions are then temporarily set aside, and the process is repeated to identify the second, third, and subsequent fronts. These fronts are labeled with numbers 1, 2, 3, and so on, indicating their non-domination level. In this way, solutions with lower ranks (e.g., the first front) have higher selection priority compared to others. Figure 3 illustrates an example of such sorting. After determining the front rank of each solution, the Crowding Distance metric is calculated for the solutions within each rank. Then, the current population is sorted according to the Crowding Distance (in descending order) and front rank (in ascending order). This process provides an effective combination of dominance information utilization and diversity preservation.

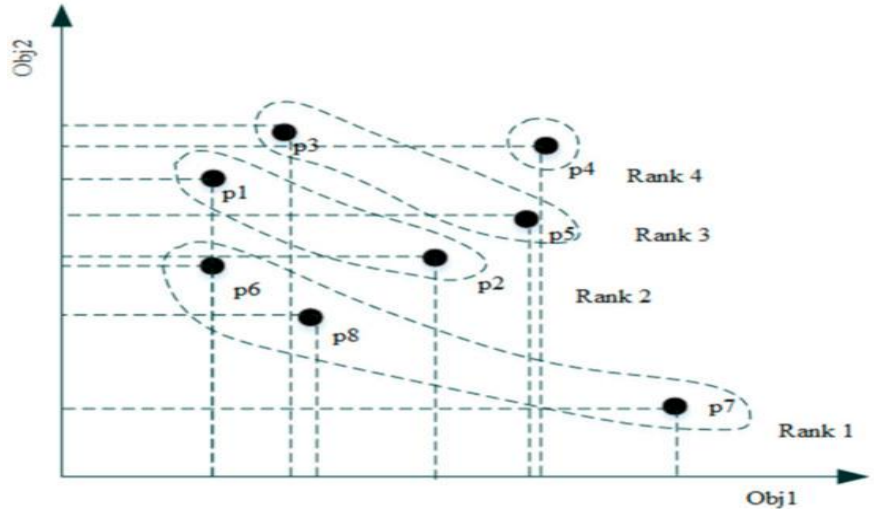


Figure 3. An Example of Non-Dominated Sorting (Xue et al., 2020)

After sorting the solutions, they create a hierarchical decile structure. In each iteration of the algorithm, each current solution is improved using the Repose function, taking into account the average of the following two solutions:

1- The solution obtained by the weighted average of the best solutions in the higher deciles (using Formula 8).

2- The solution selected by the SelectLeader function.

The SelectLeader function is designed to start a multi-step process to determine a suitable leader by receiving an archive containing a set of existing solutions, along with a selection pressure parameter (β). The archive contains information about all solutions and their positions in the network, and the parameter β controls the algorithm's tendency to select cells with less or more density than non-dominated solutions. The purpose of this function is to identify and select a solution as a leader that can act as a search guide in the subsequent stages of the algorithm.

In the first step, the network function examines all cells that contain at least one non-dominated solution. A non-dominated solution is a state in which there is no other solution in the archive that can simultaneously perform better than it in all evaluation criteria. After identifying these cells, the exact number of non-dominated solutions in each cell is calculated and denoted by the symbol k. This basic information is used to calculate the probability of selecting each cell in the next step.

Next, the probability of selecting each cell is calculated using equation (9), which is usually a mathematical relationship dependent on the value of k and the parameter β. This probability is defined in such a way that houses with lower or higher density (depending on the value of β) have different chances of being selected, ensuring search diversity and preventing premature convergence. After obtaining the probability vector, the algorithm uses the well-known roulette wheel selection method; in this method, houses occupy parts of a "virtual wheel" based on their probability value, and by spinning this wheel, a house is selected randomly but with weighted probabilities.

$$P = k^{-\beta} \tag{9}$$

Finally, after the selected house is determined, the function randomly selects one of the solutions in that house. This random selection within the house helps to maintain the diversity of solutions, and on the other hand, prevents the algorithm from locking onto a specific solution. The final output of this process is a selected solution that is returned as the "leader" and will be used as a reference and basis for decision-making in the subsequent stages of the algorithm.

After generating the leader solution and calculating the improved solution (Y), the status of this solution is checked with the current solution. If solution Y beats the current solution, it will be replaced. However, if the current solution beats the improved solution (Y), no change will be made. If neither of them beats one another, we replace solution Y with current with a probability of 50%.

Similar to any multi-objective metaheuristic algorithm, the MOSDP algorithm starts with an initial population of solutions that are initially randomly generated. After calculating the fitness of the current population and creating an archive of non-dominated solutions of the current population, the following steps are repeated until the number of calls to the fitness function is less than the FESmax value (the maximum predicted value):

Step 1: Grid the discovered target space (archive) according to the grid size and the grid inflation parameter (α).

Step 2: Calculate the position of each non-dominated solution in the created grid.

Step 3: Repeat the following steps for each d-decile (i.e. when the population is divided into d deciles), from d = 2 to d = 10 and for a number $diter = MaxIter/9$:

Step 3-1: Arrange non-dominated sorting of the current population based on the crowding distance

Step 3-2: Repeat the following steps for each of the m variables (deciling criteria):

Step 3-2-1: Divide the population into d parts (each part represents a decile).

Step 3-2-2: Calculate the average of the current solution and the leader solution, and update the position of the new solution using the repose function.

Step 3-3: Calculate the non-dominated members of the current population and update the existing archive.

Step 3-4: Maintain the number nArch of non-dominated solutions in the archive and remove the redundant ones.

In summary, the MOSDP algorithm consists of three main stages: (1) generating the initial population and creating an archive of non-dominated solutions, (2) dividing the population into deciles and updating the positions of individuals based on the Repose function, and (3) selecting a leader from the archive using the well-known probabilistic selection mechanism of the roulette wheel. The goal of these stages is to guide the population toward the real Pareto front while maintaining diversity and preventing premature convergence. Algorithm 1 shows the pseudo-code of the MOSDP algorithm. Initially, the algorithm creates the initial population through a loop from line 1 to 4. The population size is N. In each iteration of this loop (line 2), the position of each member is randomly generated between the lower bound (lb) and upper bound (ub), according to the problem dimension (dim). Then, in line 3, the cost or objective function value for this position is calculated and stored in the data structure of each member. This section essentially prepares an initial set of potential solutions for the problem, which serves as the foundation of the algorithm.

After creating the initial population, non-dominated sorting is performed on the population in line 5 to assign solutions to different fronts. In line 6, the crowding distance for each member is first calculated, and then the population is sorted based on it to maintain solution diversity. In line 7, the best non-dominated solutions are extracted and stored as the initial "archive." Then, in line 8, the space is partitioned into hypercubes to specify the location of each solution in the objective space. In line 9, the grid index is determined for each solution. Finally, in line 10, the number of internal iterations (dIter) is adjusted relative to the total number of iterations (MaxIter), which will be subsequently used.

From lines 11 to 32, the main process of solution evolution is executed. The outer loop (line 11) is repeated for values of d from 2 to 10. In line 12, the number of population members that are partitioned

into each partition is calculated (decN = N/d). Then another inner loop (line 13) is started for dIter iterations. In each iteration, as before, the nondominated sort and the crowding distance are calculated, and the population is sorted (lines 14 and 15). Then, in lines 16 to 28, the process of searching for a leader and generating new solutions is executed. In line 17, a loop of k is executed in descending order, and lines 18 and 19 specify the range of members to be updated. In line 20, a leader is selected from the archive according to the parameter β. Line 21 creates a temporary position as the average between the current member and the leader, and line 22 modifies it according to the repose function in the range of variables. Lines 23 to 25 then check whether the new solution (Y) is better; if it is superior, it replaces the current member, or with probability 0.5 if it is not clearly superior. After updating all members, in line 29 the archive is updated to include the new non-dominated solutions obtained from the population. Then, in line 30, if the archive size exceeds the allowed limit (nArch), the excess solutions are removed, according to the parameter γ, so that only the best and most diverse solutions remain. This process of inner loop iteration continues until the end of the number dIter, and then the outer loop continues with the new value of d. Finally, when all the loops are finished, in line 33 the final archive, containing the best non-dominated solutions obtained during the entire process, is returned as the output of the algorithm.

Algorithm 1 The MOSDP Algorithm

Input: *N*, *MaxIter*, *m*, *lb, ub, dim, nArch, nGrid, α, β, γ, f;*
Output: The best non-dominated solutions;
1: for *i* = 1 to *N* do
2: *M* [*i*].*pos* = *lb* + (*ub*-*lb*)*rand;
3: *M* [*i*].*cost* = *f* (*M* [*i*].*pos*);
4: end
5: (M,F) = NDS (M); //Non-dominated sorting
6: M = CalcCrowdingDistance (M,F); M = sortPopulation (M);
7: *Archive* = getNonDom (*M)*;
8: *Grid* = createHypercubes (*Archive*, *nGrid*, *α*);
9: setGridIndex (*Archive, Grid*);
10: *diter* = *MaxIter/9*;
11: for *d* =2 to 10 do
12: *decN* = *N/d;*
13: for *iter* = 1 to *dIter* do
14: (M,F) = NDS (M); //Non-dominated sorting
15: M = CalcCrowdingDistance (M,F); M = sortPopulation (M);
16: for *cr* = 1 to *m* do
17: for *k* = *d* downto 1 do
18: *sIndex*= (*d*-*k*)**decN*+1; *eIndex*= (*d*-*k*+1)**decN*;
19: for *j* = *sIndex* to *eIndex* do
20: *leader* = selectLeader (*Archive*, *β*);
21: *temp* = (M[*j*].*pos* + *leader*.*pos*)/2;
22: *Y* = *repose* (*temp*, *k*, *d*,*lb*,*ub*,*cr*);
23: if dominates (*Y*, M[*j*]) then M[*j*] = *Y*;
24: elseif dominates (M[*j*], *Y*) then nothing;
25: elseif rand <0.5 then M[*j*] = *Y*;
26: end for
27: end for
28: end for
29: *Archive* = *Archive* ∪ getNonDom (*M)*;
30: deleteExtra (*Archive*, *archSize*, *γ*);
31: end for
32: end for
33: return *Archive*;

4) Findings

To comprehensively evaluate the performance of the MOSDP algorithm in finding solutions close to the optimal answer and to measure its success rate, this algorithm was implemented in MATLAB 2019 and tested on a set of 18 standard multi-objective test functions, including the well-known UF (Mirjalili, Saremi, et al., 2016) and IMOP (Cao et al., 2021) functions. In this study, the performance of MOSDP was compared with four advanced multi-objective optimization algorithms: MOCCE, MOALO, MOSMA, and MOAHA. To ensure the accuracy of the results, all algorithms were evaluated in 30 independent runs with identical configurations, using MATLAB 2017 on an Intel® Core™ i5 processor with 6GB of RAM. The evaluation metrics included Generational Distance (GD), Inverted Generational Distance (IGD), and Maximum Spread (MS), and the results from the 30 final runs were analyzed.

In the following, we first introduce the evaluation criteria of Generational Distance (GD), Inverted Generational Distance (IGD), and Maximum Spread (MS) as well as the UF and IMOP test functions. Then, we examine the efficiency and effectiveness of the MOSDP algorithm from the perspective of the introduced criteria on the 18 proposed test functions.

Generational Distance Criterion (GD): It is a criterion that measures the average distance of the solutions found from the real Pareto front and indicates how close the obtained results are to the ideal solution set. This criterion actually indicates the accuracy of the algorithm's convergence to the optimal solutions and determines how close the search process has been to the desired and optimal areas of the target space. This criterion is calculated using the following equation (10):

$$GD = \frac{(\sum_{i=1}^{n} d_i^p)^{1/p}}{n} \tag{10}$$

where di is the Euclidean distance between the i-th point found on the Pareto front and the closest point on the true and correct Pareto front. The norm p used is usually p = 2 for the Euclidean distance. n is also the number of points found on the Pareto front. The lower the value of this criterion, the closer the algorithm's solutions are to the true ray front and the higher the convergence accuracy.

Inverse Generational Distance (IGD) Criterion: This criterion determines the average distance of the true Pareto front from the closest point in the algorithm's solutions. The use of this criterion is to simultaneously measure the accuracy and diversity of the solutions. The advantage of this criterion over the generational distance (GD) criterion is that it is more sensitive to the uniform distribution of the solutions. This criterion measures both the convergence accuracy and the dispersion of the solutions, because if part of the real front is not covered, the IGD value increases. The lower the value of this criterion, the closer the algorithm's solutions are to the real front and the higher the convergence accuracy.

Maximum Spread Criterion (MS): It is a criterion used to measure the breadth of the Pareto front found in the solution space. This criterion determines to what extent the algorithm has been able to discover the outer boundaries of the Pareto front. The standard MS for m-objective problems is expressed by Equation (11):

$$MS = \sqrt{\sum_{k=1}^{m} \left(\frac{max_i f_k^i - min_i f_k^i}{f_k^{max,treu} - f_k^{min,true}}\right)^2} \tag{11}$$

Where f_k^i is the k-th objective function value for the i-th solution point, and $f_k^{max,true}$ and $f_k^{min,true}$ are the maximum and minimum values on the real Pareto front. In fact, the numerator is the numerator of the found solutions and the denominator is the numerator of the found solutions. This criterion, especially in combination with GD and IGD, can provide a complete picture of the efficiency of the algorithm's solution space. The higher the value of this criterion, the better the algorithm at discovering the outer bounds of the Pareto front.

UF Function Set: These functions were proposed by Zhang et al. (2008). They include a set of functions with complex perspectives of unconstrained multi-objective problems designed to evaluate the convergence and diversity of solutions. These problems have controlled optimization challenges. Some examples of UF functions include UF1 with 2 objectives and a non-convex Pareto front, UF4 with

2 objectives and a curved Pareto front, UF7 with 2 objectives and a separated Pareto front, and UF8 which is designed for three-objective problems. In addition, UF5 has flat regions that can confuse the algorithm, and UF10 combines the two features of asymmetry and local optima to make it more challenging.

IMOP Function Set: Developed by Tian et al. (2018). The main advantage of these functions is that they can be used to simulate real-world problems with irregular Pareto fronts, heterogeneous solution spaces, and non-uniform distributions of optimal points. Testing the flexibility of algorithms, evaluating performance under realistic irregular conditions, and studying convergence behavior in complex spaces are some of the special applications of these functions. Types of IMOP functions are IMOP1 which includes a toroidal Pareto front, IMOP4 which includes a front with separate branches, and IMOP8 which includes a quasi-fractal front. These functions are usually used for problems with 2 to 5 objectives and advanced algorithms.

In this study, the performance of the proposed MOSDP algorithm has been evaluated in comparison with benchmark algorithms on a set of standard multi-objective test functions. To ensure the reliability of the results, each algorithm has been run 30 times independently on each test function. The performance measures used include the average (Avg) and standard deviation (Std), which are comprehensively presented in the results tables. Additionally, for a systematic comparison of the performance of the algorithms, the Friedman test was used. This non-parametric test was selected because it is suitable for comparing algorithms on a set of test functions (Friedman, 1940). In addition, to ensure a fair comparison, all algorithms were evaluated under the same conditions in MATLAB version 2017 with hardware, Intel® Core™ i5 processor and 6G RAM.

The MOSDP algorithm does not have any specific parameters, but similar to the considered algorithms, it has 8 general parameters. Table 1 shows the names, descriptions, and appropriate values for these parameters. Moreover, the relationship between the maximum number of fitness function calls and other parameters is determined by the formula FESmax = MaxIt * (N*dim*4).

Table 1. Names, Descriptions, and Appropriate Values for the Algorithm Parameters

Parameter name	Description	Appropriate value
N	Population size	40
nArch	Archive size	100
FESmax	Maximum number of fitness function calls	10E+5
nGrid	Grid size	10
dim	Number of variables (number of performance criteria)	10
α	Grid inflation size	0.1
β	Selection pressure to choose a leader	4
γ	Selection pressure to remove from archive.	2

Based on the data presented in Tables 2 to 4, which evaluate the performance of various algorithms on the UF test function suite using the GD, IGD, and MS metrics, the MOSDP algorithm has demonstrated considerable superiority in most cases. It is worth noting that lower values for the GD and IGD metrics are better, while higher results for the MS metric indicate better algorithm performance. This algorithm achieved first place across all test functions for the IGD and MS metrics, and second

place for the GD metric in the UF test functions, indicating its superior efficiency compared to the other algorithms examined.

Table 2. Results of the Algorithms (IGD Metric) on the UF Test Functions

		MOSDP	MOCCE	MOAHA	MOSMA	MOALO
UF1	Avg	0.01303	0.01285	0.01211	0.04243	0.11467
	Std	0.00268	0.00097	0.00167	0.00242	0.01209
UF2	Avg	0.00979	0.00803	0.01267	0.02835	0.08776
	Std	0.00069	0.00079	0.0013	0.00156	0.01969
UF3	Avg	0.0778	0.10955	0.35799	0.08661	0.5993
	Std	0.00847	0.01744	0.02487	0.0128	0.109
UF4	Avg	0.03251	0.04011	0.03923	0.04434	0.08237
	Std	0.00308	0.00182	0.00058	0.00078	0.02068
UF5	Avg	0.07194	0.07002	0.16434	0.3908	1.12983
	Std	0.01187	0.01133	0.01988	0.102	0.20405
UF6	Avg	0.06888	0.10832	0.15177	0.11596	0.52478
	Std	0.0106	0.06102	0.06641	0.0222	0.27023
UF7	Avg	0.02731	0.02284	0.01106	0.01739	0.09658
	Std	0.00465	0.0054	0.00057	0.00201	0.0158
UF8	Avg	0.18045	0.13993	0.13497	0.29876	0.37075
	Std	0.02081	0.01615	0.00631	0.0128	0.05629
UF9	Avg	0.01303	0.06461	0.07765	0.25876	0.31445
	Std	0.00268	0.01029	0.00745	0.00229	0.09327
UF10	Avg	0.00979	0.12476	0.31462	0.39607	1.54821
	Std	0.00069	0.01325	0.01619	0.0947	0.5343
Friedman Test	Mean Rank	1.90	2.10	2.50	3.50	5.00
	Rank	1	2	3	4	5

Table 3. Results of the Algorithms (Gd Metric) on the Uf Test Functions

		MOSDP	MOCCE	MOAHA	MOSMA	MOALO
UF1	Avg	0.00031	0.0003	0.0041	0.00012	0.00848
	Std	0.00012	0.0001	0.00456	0.00001	0.00516
UF2	Avg	0.00041	0.00025	0.00122	0.0009	0.02904
	Std	0.00008	0.00004	0.00009	0.00041	0.03831
UF3	Avg	0.00694	0.01087	0.00838	0.00362	0.02894
	Std	0.00051	0.00383	0.00527	0.00261	0.01882
UF4	Avg	0.0031	0.00392	0.0042	0.00458	0.00469
	Std	0.00013	0.00019	0.00009	0.00006	0.00053
UF5	Avg	0.0089	0.00712	0.01524	0.01916	0.18134
	Std	0.00261	0.00136	0.00306	0.0222	0.0199
UF6	Avg	0.01042	0.00989	0.11417	0.00186	0.08056
	Std	0.00277	0.00218	0.11223	0.00261	0.03933
UF7	Avg	0.00039	0.00043	0.00154	0.00025	0.00777
	Std	0.0001	0.00013	0.00065	0.00019	0.0023
UF8	Avg	0.00091	0.00123	0.1102	0.08487	0.03095
	Std	0.00029	0.00029	0.01945	0.102	0.01236
UF9	Avg	0.00792	0.00465	0.10968	0.031	0.0389
	Std	0.00768	0.00202	0.03856	0.00391	0.02745
UF10	Avg	0.00194	0.00179	1.22294	0.00092	0.40671
	Std	0.0006	0.0008	0.05325	0.00019	0.09135

Friedman Test	Mean Rank	2.10	2.00	4.10	2.30	4.50
	Rank	2	1	4	3	5

Table 4. Results of the Algorithms (MS Metric) on the UF Test Functions

		MOSDP	MOCCE	MOAHA	MOSMA	MOALO
UF1	Avg	1.2602	1.3499	0.48687	1.06433	1.19877
	Std	0.05432	0.2113	0.31518	0.23150	0.20216
UF2	Avg	0.89022	0.82817	0.25009	1.21619	1.64531
	Std	0.04453	0.07934	0.03491	0.13330	0.16741
UF3	Avg	1.27131	1.22379	1.44716	1.39777	1.12943
	Std	0.25187	0.13266	0.31786	0.16015	0.11152
UF4	Avg	1.1608	1.08845	0.27639	0.94874	1.02313
	Std	0.08777	0.05889	0.04347	0.14161	0.14401
UF5	Avg	1.89988	1.8917	1.76628	0.99517	1.21441
	Std	0.10464	0.11811	0.05814	0.01936	0.15362
UF6	Avg	1.28328	1.5454	1.56929	1.00158	1.04516
	Std	0.27129	0.26393	0.25819	0.00160	0.01725
UF7	Avg	1.31798	1.46824	0.32605	0.96642	1.39706
	Std	0.133	0.06049	0.0487	0.16564	0.16499
UF8	Avg	1.34899	1.32235	1.00967	1.10090	1.31489
	Std	0.08078	0.01616	0.30715	0.09679	0.11577
UF9	Avg	0.84939	0.88352	1.001	1.11281	1.38943
	Std	0.20903	0.06084	0.17068	0.05751	0.07445
UF10	Avg	1.35067	1.41976	0.52542	1.07452	1.41131
	Std	0.04928	0.07117	0.08173	0.06601	0.06711
Friedman Test	Mean Rank	3.60	3.60	2.20	2.40	3.20
	Rank	1	1	4	3	2

Based on the data presented in Tables 5 to 7, which evaluate the performance of various algorithms on the IMOP test function suite using the GD, IGD, and MS metrics, the MOSDP algorithm demonstrates better performance than other methods across most test functions. Specifically, lower values for the GD and IGD metrics indicate the algorithm's strong convergence toward the true Pareto front, while a higher MS value reflects adequate coverage of the objective space. These results confirm the algorithm's ability to maintain a balance between convergence and diversity.

Table 5. Results of the Algorithms (IGD Metric) on the IMOP Test Functions

		MOSDP	MOCCE	MOAHA	MOSMA	MOALO
IMOP1	Avg	0.41642	0.57920	0.08472	0.49059	0.16354
	Std	0.16814	0.10139	0.00260	0.10628	0.06290
IMOP2	Avg	0.38084	0.34743	0.10976	0.29143	0.51038
	Std	0.01332	0.12599	0.01392	0.05591	0.24841
IMOP3	Avg	0.18269	0.12676	0.16682	0.50325	0.09305
	Std	0.02139	0.04446	0.00313	0.11569	0.02492
IMOP4	Avg	0.6265	0.47196	0.14407	0.48560	0.35652
	Std	0.01462	0.17573	0.04762	0.09367	0.14569
IMOP5	Avg	0.31904	0.40717	0.22467	0.56900	0.71298
	Std	0.00067	0.04299	0.03041	0.02128	0.00000
IMOP6	Avg	0.32379	0.28303	0.17931	0.54018	0.45419
	Std	0.08854	0.03423	0.03763	0.05583	0.00791

IMOP7	Avg	0.6083	0.60277	0.33191	0.52446	0.84290
	Std	0.00628	0.17714	0.05369	0.02399	0.15706
IMOP8	Avg	0.26006	0.24184	0.21809	0.71596	0.57729
	Std	0.01747	0.02425	0.02806	0.04916	0.08753
Friedman Test	Mean Rank	3.50	2.88	1.25	3.88	3.50
	Rank	3	2	1	4	3

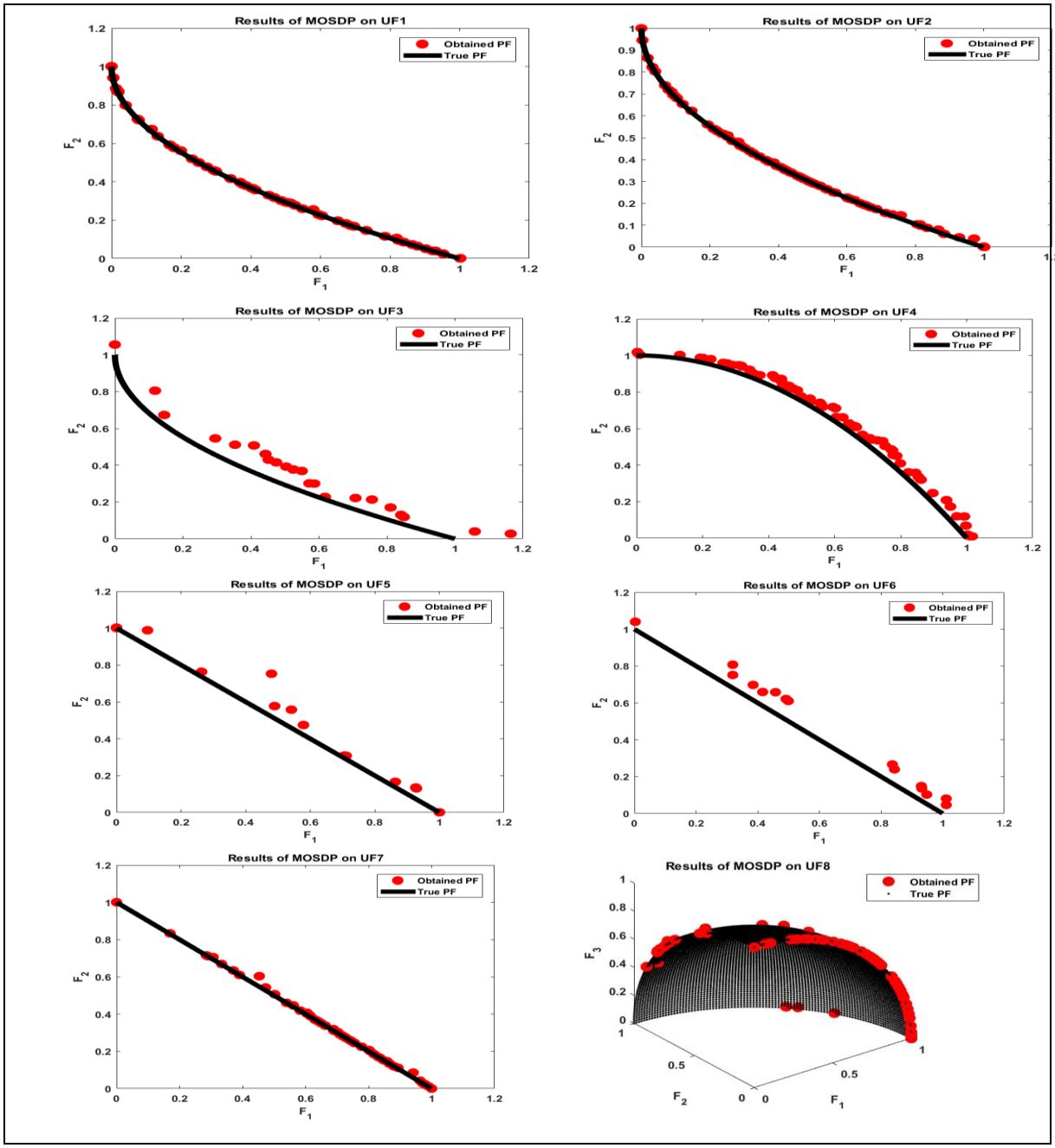
Table 6. Results of the Algorithms (GD Metric) on the UF Test Functions

		MOSDP	MOCCE	MOAHA	MOSMA	MOALO
IMOP1	Avg	0.00001	0.00001	0.00004	0.03404	0.00013
	Std	0.00000	0.00001	0.00001	0.00125	0.00016
IMOP2	Avg	0.00001	0.00001	0.00020	0.00515	0.00593
	Std	0.00000	0.00000	0.00016	0.00534	0.00515
IMOP3	Avg	0.00084	0.00185	0.00193	0.01237	0.00348
	Std	0.00022	0.00173	0.00119	0.00107	0.00166
IMOP4	Avg	0.00001	0.00002	0.00150	0.02871	0.03713
	Std	0.00000	0.00001	0.00033	0.02031	0.02451
IMOP5	Avg	0.00027	0.00035	0.00168	0.00386	0.00000
	Std	0.00001	0.00001	0.00007	0.00080	0.00000
IMOP6	Avg	0.00037	0.00040	0.00320	0.05475	0.01046
	Std	0.00002	0.00001	0.00090	0.03897	0.00551
IMOP7	Avg	0.0003	0.00232	0.00135	0.02358	0.01145
	Std	0.00001	0.00458	0.00014	0.00507	0.01226
IMOP8	Avg	0.00618	0.00678	0.01221	0.06104	0.03129
	Std	0.0007	0.00085	0.00082	0.02814	0.01187
Friedman Test	Mean Rank	1.25	2.13	3.00	4.75	3.88
	Rank	1	2	3	5	4

Table 7. Results of the Algorithms (MS Metric) on the IMOP Test Functions

		MOSDP	MOCCE	MOAHA	MOSMA	MOALO
IMOP1	Avg	1.39678	1.06819	0.94579	1.42164	1.43902
	Std	0.50599	0.05528	0.00897	0.10662	0.32829
IMOP2	Avg	1.01766	1.26031	1.15801	1.56909	1.44119
	Std	0.00359	0.38454	0.06535	0.28765	0.38553
IMOP3	Avg	1.78258	1.40578	1.52993	1.74366	1.67845
	Std	0.01893	0.13130	0.03319	0.12057	0.05808
IMOP4	Avg	1.01403	1.57230	1.21758	1.54812	1.37886
	Std	0.0245	0.41840	0.11060	0.20374	0.08276
IMOP5	Avg	1.12129	1.02386	0.63850	0.96379	1.00000
	Std	0.01783	0.04822	0.02501	0.07361	0.00000
IMOP6	Avg	1.26428	1.18548	0.75281	1.10523	1.12528
	Std	0.31711	0.10873	0.21984	0.38016	0.06513
IMOP7	Avg	1.12995	1.21415	1.42318	1.49893	1.05289
	Std	0.23958	0.27403	0.02658	0.17034	0.06492
IMOP8	Avg	1.17175	1.01307	0.62668	0.98974	0.93459
	Std	0.00345	0.06180	0.05114	0.17574	0.05497
Friedman Test	Mean Rank	3.48	3.25	1.75	3.43	3.00
	Rank	1	3	5	2	4

The results presented in Figures 4 and 5 illustrate a comparison between the true Pareto-optimal (reference) set and the solutions obtained by the MOSDP algorithm. Analysis of these results indicates that the MOSDP algorithm, in most cases, successfully generates solutions with the minimum distance from the optimized Pareto front. The aforementioned advantages establish MOSDP as a superior option compared to other existing methods in the field of multi-objective optimization. It is noteworthy that this favorable performance has been observed across a wide range of test functions, which demonstrates the generalizability of the proposed method. Due to the simple and non-parametric structure of the MOSDP algorithm, it can be applied to solve real-world engineering problems, such as wireless sensor network design, energy consumption optimization in smart systems, multi-objective production scheduling, and path planning for autonomous robots.



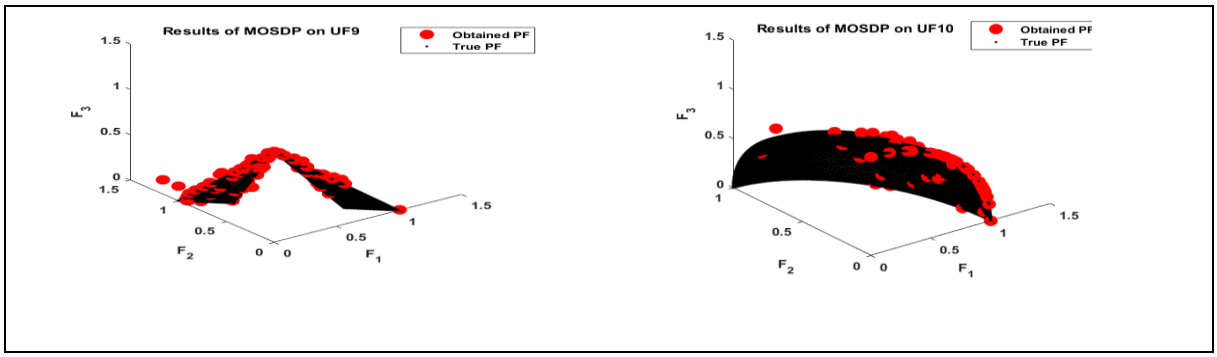
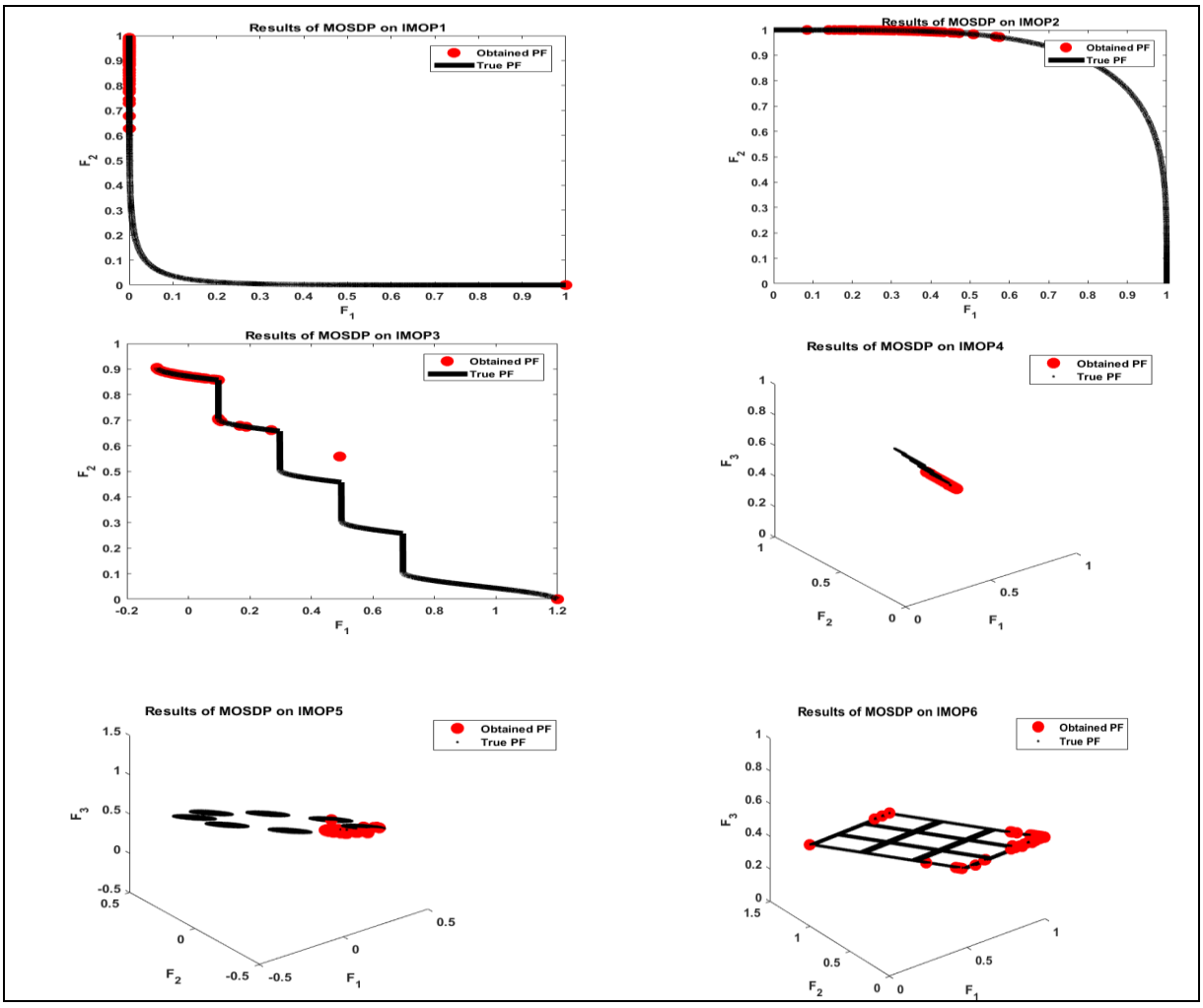


Figure 4. The True and Obtained Pareto-Optimal Sets by the MOSDP Algorithm on the UF Test Functions



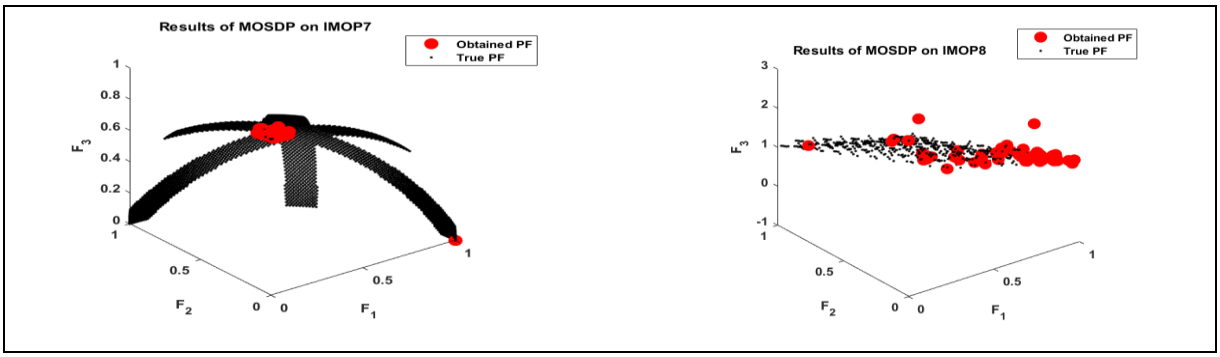


Figure 5. The True and Obtained Pareto-Optimal Sets by the MOSDP Algorithm on the IMOP Test Functions

4-1) Statistical Analysis and the Stability of Results

To evaluate the stability of the algorithms' performance, the mean (Avg) and standard deviation (Std) values obtained from 30 independent runs were analyzed. Lower standard deviation values indicate higher stability and lower sensitivity of the algorithm to random initial conditions. The results in Tables 2 to 7 indicate that the MOSDP algorithm has a smaller standard deviation compared to other algorithms across most test functions, indicating greater stability and consistency in its performance. Particularly, in test functions UF1 to UF5, the Std value for the IGD and GD metrics in the MOSDP algorithm was on average less than 0.01, while in other algorithms, this value sometimes increased up to 0.1. This demonstrates that MOSDP is not only effective in achieving optimal values but is also less susceptible to random fluctuations. Furthermore, analysis of the mean (Avg) values alongside the Friedman test shows that MOSDP has a better average performance across all evaluation metrics (IGD, GD, and MS) compared to other methods. Therefore, it can be concluded that the proposed algorithm not only exhibits superior average performance but also demonstrates high statistical stability and low variance across different execution runs, a feature that is of great importance for real-world applications.

5) Conclusion and Recommendations

In this paper, the proposed algorithm was designed and implemented to address the challenges of multi-objective optimization. This algorithm focuses on improving efficiency in complex problems that require the simultaneous optimization of multiple conflicting objectives. The main goal of the algorithm is to provide a set of solutions that, while close to the true Pareto front, also maintain sufficient diversity among the solutions. To evaluate its performance, three key metrics—Generational Distance (GD), Inverted Generational Distance (IGD), and Maximum Spread (MS)—were used to quantitatively assess the quality of the obtained solutions. The evaluation results showed that the proposed algorithm achieved highly favorable values for the GD and IGD metrics, indicating strong convergence and broad, uniform coverage of the objective space. In particular, the low IGD value demonstrates the algorithm’s ability to closely approach the true Pareto front and comprehensively cover its different regions by providing suitable samples of key points.

The algorithm proposed in this paper is primarily designed for bi-objective or multi-objective problems with a limited number of objectives and can, in the future, be extended to optimization problems with a much larger number of objectives, thereby assessing its ability to handle the complexities of broader objective spaces. Furthermore, one of the emerging areas in this field is the integration of metaheuristic algorithms with machine learning and reinforcement learning methods, which could bring significant improvements in.

References

Abbass, H. A., Sarker, R., & Newton, C. (2001). PDE: A Pareto-frontier differential evolution approach for multi-objective optimization problems. In Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546).

Akbari, R., Hedayatzadeh, R., Ziarati, K., & Hassanizadeh, B. (2012). A multi-objective artificial bee colony algorithm. Swarm and Evolutionary Computation, 2, 39-52.

Alaya, I., Solnon, C., & Ghedira, K. (2007). Ant colony optimization for multi-objective optimization problems. In 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007).
Cao, J., Zhang, J., Zhao, F., & Chen, Z. (2021). A two-stage evolutionary strategy based MOEA/D to multi-objective problems. Expert Systems with Applications, 185, 115654.
Coello, C. C., & Lechuga, M. S. (2002). MOPSO: A proposal for multiple objective particle swarm optimization. In Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600).
Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2000a). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In International Conference on Parallel Problem Solving from Nature.
Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2000b, September 18–20). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. Parallel Problem Solving from Nature PPSN VI. In 6th International Conference. Paris, France.
Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. IEEE Computational Intelligence Magazine, 1(4), 28-39.
Faramarzi, A., Heidarinejad, M., Stephens, B., & Mirjalili, S. (2020). Equilibrium optimizer: A novel optimization algorithm. Knowledge-Based Systems, 191, 105190.
Friedman, M. (1940). A comparison of alternative tests of significance for the problem of m rankings. The Annals of Mathematical Statistics, 11(1), 86-92.
Ghaffar Alishahi, M., Pira, E., & Rouhi, A. (2023). Development of city councils evolution algorithm for multi-objective optimization problems. Soft Computing Journal.
Gómez, R. H., & Coello, C. A. C. (2013). MOMBI: A new metaheuristic for many-objective optimization based on the R2 indicator. 2013 IEEE Congress on Evolutionary Computation.
Holland, J. H. (1992). Genetic algorithms. Scientific American, 267(1), 66-73.
Horn, J., Nafpliotis, N., & Goldberg, D. E. (1994). A niched Pareto genetic algorithm for multiobjective optimization. In Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress On Computational Intelligence.
Kaur, H., Rai, A., Bhatia, S. S., & Dhiman, G. (2020). MOEPO: A novel multi-objective emperor penguin optimizer for global optimization: Special application in ranking of cloud service providers. Engineering Applications of Artificial Intelligence, 96, 104008.
Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In Proceedings of ICNN'95-International Conference on Neural Networks.
Khishe, M., Orouji, N., & Mosavi, M. R. (2023). Multi-objective chimp optimizer: an innovative algorithm for multi-objective problems. Expert Systems with Applications, 211, 118734.
Khodadadi, N., Mirjalili, S. M., Zhao, W., Zhang, Z., Wang, L., & Mirjalili, S. (2022). Multi-objective artificial hummingbird algorithm. In Advances in Swarm Intelligence: Variations and Adaptations for Optimization Problems (pp. 407-419). Springer.
Mirjalili, S., Jangir, P., & Saremi, S. (2017). Multi-objective ant lion optimizer: a multi-objective optimization algorithm for solving engineering problems. Applied Intelligence, 46(1), 79-95.
Mirjalili, S., Mirjalili, S. M., & Hatamlou, A. (2016). Multi-verse optimizer: a nature-inspired algorithm for global optimization. Neural Computing and Applications, 27(2), 495-513.
Mirjalili, S., Saremi, S., Mirjalili, S. M., & Coelho, L. d. S. (2016). Multi-objective grey wolf optimizer: A novel algorithm for multi-criterion optimization. Expert Systems with Applications, 47, 106-119.
Murata, T., & Ishibuchi, H. (1995). MOGA: multi-objective genetic algorithms. In IEEE International Conference on Evolutionary Computation.
Naruei, I., & Keynia, F. (2021). Wild horse optimizer: A new meta-heuristic algorithm for solving engineering optimization problems. Engineering with Computers, 1-32.
Pira, E. (2022). City councils evolution: A socio-inspired metaheuristic optimization algorithm. Journal of Ambient Intelligence and Humanized Computing, 1-50.
Pira, E., & Rouhi, A. (2024). Society deciling process: A socio-inspired meta-heuristic algorithm. Journal of Electrical and Computer Engineering Innovations (JECEI), 12(2), 535-556.
Połap, D., & Woźniak, M. (2021). Red fox optimization algorithm. Expert Systems with Applications, 166, 114107.
Premkumar, M., Jangir, P., & Sowmya, R. (2021). MOGBO: A new multiobjective gradient-based optimizer for real-world structural optimization problems. Knowledge-Based Systems, 218, 106856.
Premkumar, M., Jangir, P., Sowmya, R., Alhelou, H. H., Heidari, A. A., & Chen, H. (2020). MOSMA: Multi-objective slime mould algorithm based on elitist non-dominated sorting. IEEE Access, 9, 3229-3248.
Premkumar, M., Jangir, P., Sowmya, R., Alhelou, H. H., Mirjalili, S., & Kumar, B. S. (2022). Multi-objective equilibrium optimizer: Framework and development for solving multi-objective optimization problems. Journal of Computational Design and Engineering, 9(1), 24-50.
Sadollah, A., Eskandar, H., & Kim, J. H. (2015). Water cycle algorithm for solving constrained multi-objective optimization problems. Applied Soft Computing, 27, 279-298.
Xue, L., Zeng, P., & Yu, H. (2020). SETNDS: A SET-based non-dominated sorting algorithm for multi-objective optimization problems. Applied Sciences, 10(19), 6858.
Zeng, S.-Y., Chen, G., Zheng, L., Shi, H., de Garis, H., Ding, L., & Kang, L. (2006). A dynamic multi-objective evolutionary algorithm based on an orthogonal design. In 2006 IEEE International Conference on Evolutionary Computation.
Zhang, Q., & Li, H. (2007). MOEA/D: A multiobjective evolutionary algorithm based on decomposition. IEEE Transactions on Evolutionary Computation, 11(6), 712-731.

Zitzler, E., & Künzli, S. (2004). Indicator-based selection in multiobjective search. In International Conference on Parallel Problem Solving from Nature.
Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA2: Improving the strength Pareto evolutionary algorithm. TIK-Report, 103.