# A Novel Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: DSGA

**Qiang Long** ( 27131375@qq.com )
SWUST: Southwest University of Science and Technology https://orcid.org/0000-0002-9672-5895
**Guoquan Li**
Chongqing Normal University - University Town Campus: Chongqing Normal University
**Lin Jiang**
Curtin University

**Research Article**

**Keywords:** Multi-objective evolutionary algorithm , Multi-objective genetic algorithm , Non-dominated sorting , Pareto frontier

**Posted Date:** April 12th, 2021

**DOI:** https://doi.org/10.21203/rs.3.rs-304472/v1

**Noname manuscript No.**
(will be inserted by the editor)

# A novel non-dominated sorting genetic algorithm for multi-objective optimization: DSGA

**Qiang Long · Guoquan Li · Lin Jiang**

Received: date / Accepted: date

**Abstract** Non-dominated sorting is a critical component of all multi-objective evolutionary algorithms (MOEAs). A large percentage of computational cost of MOEAs is spent on non-dominated sorting. So the complexity of non-dominated sorting method in a large extent decides the efficiency of the MOEA. In this paper, we present a novel non-dominated sorting method called the dynamic non-dominated sorting (DNS). It is based on the sorting of each objective instead of dominance comparisons. The computational compelxity of DNS is $O(mN \log N)$ ($m$ is the number of objectives, $N$ is the population size), which equals to the best record so far. Based on DNS, we introduce a novel multi-objective genetic algorithm (MOGA) called the dynamic non-dominated sorting genetic algorithm (DSGA). Then, some numerical comparisons between different non-dominated sorting method are presented. The results shows that DNS is efficient and promising. Finally, numerical experiments on DSGA are also given. The results show that DSGA outperforms some other MOEAs both on general-scale and large-scale multi-objective problems.

**Keywords** Multi-objective evolutionary algorithm · Multi-objective genetic algorithm · Non-dominated sorting · Pareto frontier

Q. Long
School of Science, Southwest University of Science and Technology, Mianyang 621010, China
E-mail: longqiang@swust.edu.cn

G.Q. Li
School of Mathematical Science, Chongqing Normal University, Chonqing 400047, China
E-mail: ligq@cqnu.edu.cn

L. Jiang
School of Engineering, Computing and Mathematics, Curtin University, 6102, Australia
E-mail: lydiajianglin@gmail.com

## 1 Introduction

Multi-objective optimization has extensive application in engineering and management. Many optimization problems in the real-world can be modeled as multi-objective optimization problems (MOPs) [10], [23], [35]. However, due to the theoretical and computational challenges, it is not easy to solve MOPs. Therefore, numerical multi-objective optimization attracts a wide range of research over the last decades.

One popular way to solve MOP is to reformulate it into a single-objective optimization problem. We call this technique the *indirect method*. Typical indirect methods are weighted sum method [31], $\varepsilon-$constraint method [7] and their variations [11]. One difficulty for the weighted sum method is the selection of proper weights so as to satisfy the decision-maker's preference. Since the weighted sum is a linear combination of the objective functions, the concave part of the Pareto frontier cannot be obtained using the weighted sum method. The $\varepsilon-$constraint method converts multiple objectives, except one, to constraints. However, it is difficult to determine the upper bounds of these objectives. On the one hand, small upper bounds could exclude some Pareto solutions; on the other hand, large upper bounds could enlarge the searching area, which yields some sub-Pareto solutions. Additionally, indirect method can only obtain a single Pareto solution in each run, but in the real-world application, decision-makers often prefer a number of optional strategies so that they can choose one according to their preference.

Another strategy to solve MOP is to explore the entire objective function value space directly in order to obtain its Pareto frontier. We call this strategy the *direct method*. Population-based heuristic methods are ideal direct methods, because their iterative units are populations instead of a single point, so they can obtain a set of solutions in a single run. In the past few years, many heuristic methods were applied to solve MOP, such as the evolutionary algorithm [19], [25], [39], genetic algorithm [34] and differential evolution [22], [24]. Among them, genetic algorithm attracted a great deal of attention and lots of good methods has been presented [3], [13], [16], [17].

A combination of direct and indirect methods is another strategy to solve MOP. We call this type the *hybrid method*. A representative of this type is MOEA/D [36]. It combines the evolutionary algorithm and three scalarization methods.

When solving MOP using direct methods, two important issues [42] need to be addressed:

- *Elitism:* In the process of multi-objective evolutionary algorithm (MOGA), we always prefer solutions whose function values are closer to the real Pareto frontier. In the selection procedure, these solutions should be selected as parents for the next generation, which leads the task of non-dominated sorting in each iteration. According to the definition of efficient point, numerous amount of comparisons are needed to identify the non-dominated state of each point. Therefore, reasonably reducing the computational cost is one of the key research issues in designing MOEA.

– *Diversity:* Diversity reflects the distribution of the obtained Pareto frontier. Obviously, a uniformly distributed Pareto frontier is preferred than a unevenly distributed one. In other words, a good algorithm should avoid obtaining solutions which are excessively concentrate on one or two isolated areas.

Among them, elitism is realized by non-dominated sorting whose core operation is comparison. Since non-dominated sorting is needed in each iteration, the efficiency of non-dominated sorting method is very important for the performance of MOEA. In this paper, we first propose a novel non-dominated sorting method called the *dynamic non-dominated sorting (DNS)*, and then a new multiobjective genetic algorithm (MOGA) based on DNS.

The rest of the paper is organized as follows: In Section 2, we review some existing non-dominated sorting methods and the framework of genetic algorithm. In Section 3, we proposed DNS and a new MOGA based on DNS. In Section 4, we compare DNS with other existing non-dominated sorting methods. In Section 5, we test the proposed MOGA using some numerical benchmarks and compare its numerical performance with other MOEAs. Section 6 concludes the paper.

## 2 Related works

In this section, we first review some basic definitions of multiobjective optimization, then some existing non-dominated sorting methods, and finally propose the framework of genetic algorithm.

### 2.1 Some basic definitions of MOP

The general mathematical model of MOP is

$$(\text{MOP}) \quad \begin{cases} \text{Minimize} \quad F(\boldsymbol{x}) \\ \text{Subject to } \boldsymbol{x} \in X. \end{cases} \tag{1}$$

Here $F : \mathbb{R}^n \mapsto \mathbb{R}^m$ is a vector function and $X \subseteq \mathbb{R}^n$ is a box constraint. In this paper, we also call $X$ the *decision variable space*, and its image set $F(X) = \{F(\boldsymbol{x}) | \boldsymbol{x} \in X\}$ the *objective function value space*.

Given two vectors $\boldsymbol{y} = (y_1, y_2, \cdots, y_m)^T$ and $\boldsymbol{z} = (z_1, z_2, \cdots, z_m)^T \in \mathbb{R}^m$, define:

(1) $\boldsymbol{y} = \boldsymbol{z} \Leftrightarrow y_i = z_i$ for all $i = 1, 2, \cdots, m$;
(2) $\boldsymbol{y} \prec \boldsymbol{z} \Leftrightarrow y_i < z_i$ for all $i = 1, 2, \cdots, m$;
(3) $\boldsymbol{y} \preceq \boldsymbol{z} \Leftrightarrow y_i \leq z_i$ for all $i = 1, 2, \cdots, m$, and $\boldsymbol{y} \neq \boldsymbol{z}$.

Obviously, if $\boldsymbol{y} \prec \boldsymbol{z}$, then $\boldsymbol{y} \preceq \boldsymbol{z}$. In this paper, if $\boldsymbol{y} \preceq \boldsymbol{z}$, we say $\boldsymbol{y}$ *dominates* $\boldsymbol{z}$ or $\boldsymbol{z}$ *is dominated by* $\boldsymbol{y}$; if $\boldsymbol{y} \not\preceq \boldsymbol{z}$ and $\boldsymbol{z} \not\preceq \boldsymbol{y}$, we say $\boldsymbol{y}$ and $\boldsymbol{z}$ are *non-dominated*.

Let $Y \subseteq \mathbb{R}^m$ and $\boldsymbol{y}^* \in Y$, if there is no $\boldsymbol{y} \in Y$ such that $\boldsymbol{y} \preceq \boldsymbol{y}^*$ (or $\boldsymbol{y} \prec \boldsymbol{y}^*$), then $\boldsymbol{y}^*$ is called an *efficient point* (or *weakly efficient point*) of $Y$. Suppose that

$\boldsymbol{x}^* \in X$, if there is no $\boldsymbol{x} \in X$ such that $F(\boldsymbol{x}) \preceq F(\boldsymbol{x}^*)$ (or $F(\boldsymbol{x}) \prec F(\boldsymbol{x}^*)$), i.e. $F(\boldsymbol{x}^*)$ is an efficient point (or weakly efficient point) of the objective function value space $F(X)$, then $\boldsymbol{x}^*$ is called an *efficient solution* (or *weakly efficient solution*) of Problem (1).

Efficient and weakly efficient solutions can also be defined using cone or partial order. Another name of the efficient solution is *Pareto solution*. The meaning of Pareto solution is that, if $x^*$ is a Pareto solution, there is no feasible solution $x \in X$, such that $f_i(x) \leq f_i(x^*)$ for all $i \in \{1, 2, \cdots, m\}$ and there is at least one $i_0 \in \{1, 2, \cdots, m\}$ such that $f_{i_0}(x) < f_{i_0}(x^*)$. In other words, $x^*$ is the best solution in the sense of "$\preceq$". Another intuitive interpretation of Pareto solution is that it cannot be improved with respect to any objective without worsening at least one of the others. Weakly efficient solution means that if $x^*$ is a weakly efficient solution, then there is no feasible solution $x \in X$, such that any $f_i(x)$ of $F(x)$ is strictly better than that of $F(x^*)$. In other words, $x^*$ is the best solution in the sense of "$\prec$". Obviously, an efficient solution is also a weakly efficient solution. The set of efficiet solutions is denoted by $\mathcal{P}^*$, its image set $F(\mathcal{P}^*)$ is called the *Pareto frontier*, denoted by $\mathcal{PF}^*$.

### 2.2 Existing non-dominated sorting methods

Non-dominated sorting is one of the critical step in MOEAs. A large percentage of computation cost is spent on non-dominated sorting for it involves numerous comparisons. Until now, there are more than ten different non-dominated sorting methods [18]. The earliest one is the naive non-dominated sorting [26]. In this method, in order to find if a solution is dominated, the solution has to do dominance comparisons with all the other solutions, which leads a numerical complexity of $O(mN^3)$ in the worst case. Here, $m$ stands for the number of objectives, $N$ stands for the population size, the same below. The naive approach does not record any dominance comparison result, which in return cause a lot of repeated computation. The fast non-dominated sorting [3] overcomes this drawback by calculating two entities for each solution $p$: (i) domination count $n_p$, the number of solutions which dominate the solution $p$; and (ii) dominate set $S_p$, a set of solutions that the solution $p$ dominates. This modification reduces the computational complexity of the fast non-dominated sorting to $O(mN^2)$.

In [20] McClymont and Keedwell proposed climbing sorting and deductive sorting. The climbing sorting is similar to the bubble sorting for real number sequence. In order to locate a non-dominated solution, the candidate is shifted from dominated solution to the dominating one until all the solutions in the current population are visited. The final candidate must be a non-dominated solution. The deductive sorting is similar to the selection sorting for real number sequence. In the deductive sorting, one successively check each solution, once the current solution is found to be dominated, abandon it. If the current solution dominates all the other solutions, or is non-dominated with the others, then it is identified as a non-dominated solution. The climbing sorting works

well on populations with large number of Pareto frontiers. On the contrary, the deductive sorting is good at sorting population with small number of Pareto frontiers. The average computational complexity of the climbing sorting and deductive are both $O(mN^2)$.

Another idea to save dominance comparisons is to discard the identified non-dominated solutions, Corner sorting [30] applies this idea. Some of the non-dominated sorting method consider the partial order of the solutions in the populations. For example, the efficient non-dominated sorting [38] first sorts solutions using the common partial order of $m-$dimensional vector. This sorted population has a very important feature: solution $\boldsymbol{p}_m$ will never be dominated by a solution $\boldsymbol{p}_n$ if $m < n$. As shown in Table (III) in [38], computational complexity of the efficient non-dominated sorting is $O(mN^2)$ in the worst case, while $O(mN\sqrt{N})$ or $O(mN \log N)$ in the best case (depends on the different methods used in partial order.)

In order to avoid repeated dominance comparisons, one can record the results of dominance comparisons in a matrix. This idea is applied in the dynamic non-dominated sorting [18] where a $N \times N$ matrix with entries $1, -1, 0$ is used to record the dominance relationships between solutions. The computational complexity of the dynamic non-dominated sorting is $O(mN^2)$. Another method applying this idea is the dominance degree sorting [41]. This method first calculate a comparison matrix for each objective, then builds a dominance degree matrix by adding these comparison matrixes together. The dominance relationship can be found in the dominance degree matrix. In the dominance degree sorting, only the real number sequence sorting is needed. This makes it much faster than the method based on dominance comparison. In this paper, we improve the dynamic non-dominated sorting by hybridizing it with the dominance degree sorting. The computational complexity of the improved dynamic non-dominated sorting will decrease to $O(mN \log N)$.

### 2.3 Genetic algorithm

In computer science and operations research, genetic algorithm (GA) which is inspired by the process of natural selection, is one of the most popular evolutionary algorithms. It is introduced by John Holland in 1960s, and then developed by his students and colleagues at the University of Michigan between 1960s and 1970s [9]. Over the last two decades, GA was increasingly enriched by plenty of literatures [12], [33], [21]. Nowadays, GA are applied in a wide range of areas, such as mathematical programming, combinational optimization, automatic control, image processing, etc..

Suppose $P(t)$, $O(t)$ and $S(t)$ represent parents, offsprings and selection pool of the $t^{th}$ generation, respectively. The general structure of GA is described in Algorithm 1.

**Algorithm 1:** Genetic algorithm

**Input**: Population size $N_p$, crossover rate $\alpha_c$, mutation rate $\alpha_m$, maximal generation number $t_{max}$ and problem parameters.
**Output**: Obtained population and their corresponding evaluation.
Step 1: Initialization
  Step 1.1: Generate the initial population $P(0)$,
  Step 1.2: Evaluate $P(0)$,
  Step 1.3: Let $t \leftarrow 1$.
Step 2: While $t$ has not reached $t_{max}$, do
  Step 2.1: *Crossover operator:* generate crossover offspring $O_c(t)$,
  Step 2.2: *Mutation operator:* generate mutation offspring $O_m(t)$,
  Step 2.3: *Evaluation:* evaluate $O_c(t)$ and $O_m(t)$ and build the selection pool by

$$S(t) = P(t) \cup O_c(t) \cup O_m(t).$$

  Step 2.4: *Selection operator:* select $P(t+1)$ from $S(t)$,
  Step 2.5 Let $t \leftarrow t+1$, go back to Step 2.

The implementation of GA may be various in different situations. For example, some implementation generate $O_c(t)$ and $O_m(t)$ independently based on $P(t)$, while some implementation first generate $O_c(t)$ based on $P(t)$, then yields $O_m(t)$ based on $O_c(t)$. Furthermore, crossover, mutation and selection operators are alterable in GA. Different designs of these operators lead to different numerical performance of GA.

It is worth to notice that, for notation $P(t)$, $O(t)$ and $S(t)$, we do not specify whether they are decision variables (i.e., $P(t)/O(t)/S(t) \subset X$) or objective function values (i.e., $P(t)/O(t)/S(t) \subset F(X)$) since they are bijection through objective function $F(x)$. One can distinguish them according to the contexts. For example, when calculating the Pareto frontier, they are seen as objective function values; when running crossover and mutation, they are seen as decision variables.

## 3 Dynamic sorting genetic algorithm

In this section, we first propose an improved dynamic non-dominated sorting, then based on the improved dynamic non-dominated sorting, design a novel multiobjective optimization genetic algorithm titled the dynamic sorting genetic algorithm.

### 3.1 Dynamic non-dominated sorting

According to the definition of efficient point, in order to detect the non-dominated points in a selection pool, each solution must compare with the others in the selection pool to find if it is dominated. In the naive sorting

[2], the non-dominated points in different Pareto frontiers are detected one by one, so there exists numerous repetitive dominance comparisons between some candidate pairs, which in a large extent increases the computational complexity of native sorting. In this subsection, we tackle this shortage by recording the result of dominance comparison between each candidate pairs, which in return, avoids any repeated comparison. This idea is inspired by the dynamic programming, so we call it the *dynamic non-dominated sorting*.

Suppose $S(t)$ is the current selection pool who has $N$ solutions. The aim of non-dominated sorting is to identify all the Pareto frontiers in $S(t)$. We denote the $i^{th}$ Pareto frontier $l_i$ $(1 \leq i \leq i_{max})$. Obviously, we have $1 \leq i_{max} \leq N$, $i_{max} = 1$ means that all the solutions in $S(t)$ are non-dominated, $i_{max} = N$ means that each Pareto frontier has only one solution. We use a $N \times N$ matrix $D$ to record the dominance relationship in $S(t)$, where

$$d_{ij(i \neq j)} = \begin{cases} 1, & \text{if } \boldsymbol{y}_i \text{ dominates } \boldsymbol{y}_j, \text{i.e.}, \boldsymbol{y}_i \preceq \boldsymbol{y}_j; \\ -1, & \text{if } \boldsymbol{y}_i \text{ is dominated by } \boldsymbol{y}_j, \text{i.e.}, \boldsymbol{y}_i \succeq \boldsymbol{y}_j; \\ 0, & \text{others .} \end{cases} \tag{2}$$

The non-dominated solutions can be detected using $D$. Each row of $D$ stands for a solution. If there is no $-1$ in the $i^{th}$ row, that means $\boldsymbol{y}_i$ is not dominated by any other solution, i.e., $\boldsymbol{y}_i$ is non-dominated. Finding all rows with this feature, we can detect all the non-dominated solutions. These solutions consist the first Pareto frontier of the selection pool $S(t)$. Assign $l_i = 1$ to these points. When detecting the second Pareto frontier, solutions on the first Pareto frontier (already identified) should not be involved any more. So before detecting the second Pareto frontier, we first shrink $D$ by discarding the rows and columns whose solutions are already identified as in the first Pareto frontier. Then repeat the same process to detect the second Pareto frontier and so forth, until all the solutions are identified, i.e., the matrix $D$ becomes empty.

The procedure of the dynamic non-dominated sorting is presented in Algorithm 2. The input is the current selection pool $S(t)$, its size is $N$; the output is the Pareto frontier index $l_i,\ i = 1, 2, \cdots, N$. Step 1 computes the $N \times N$ dynamic matrix $D$. Step 2 detects the current non-dominated solution using the current $D$. Step 3 shrinks the matrix $D$ by removing rows and columns corresponding to the detected non-dominated solutions in Step 2. Then if $D$ is not empty, go back to Step 2 to detect non-dominated solution in the next Pareto frontier; otherwise, the process of non-dominated detection finishes. It is worth to note that after some rows and columns have been removed, the index of the matrix $D$ and the index of solution in the selection pool are not corresponding any more. So in Algorithm 2, we use an index set $I_{ij}$ to virtually shrink the matrix $D$. Here the index $i$ always means the index of a solution in the selection pool.

**Algorithm 2:** Dynamic non-dominated sorting

**Input**: Selection pool $S(t)$ and its size $N = |S(t)|$, an index set $I_{ij} = \{1, 2, \cdots, N\}$, a counter $k = 0$.

**Output**: Pareto frontier index $l_i,\ i = 1, 2, \cdots, N$.

Step 1: Compute a $N \times N$ dynamic matrix $D$ whose element is

$$d_{ij(i\neq j)} = \begin{cases} 1, & \text{if } \boldsymbol{y}_i \text{ dominates } \boldsymbol{y}_j, \text{i.e.}, \boldsymbol{y}_i \preceq \boldsymbol{y}_j; \\ -1, & \text{if } \boldsymbol{y}_i \text{ is dominated by } \boldsymbol{y}_j, \text{i.e.}, \boldsymbol{y}_i \succeq \boldsymbol{y}_j; \\ 0, & \text{others .} \end{cases}$$

If $i = j$, then $d_{ij} = 0$.

Step 2: Let $k := k + 1$, search by rows, find set

$$I = \{i \in I_{ij} \mid d_{ij} \neq -1,\ \forall j \in I_{ij}\},$$

then $y_i$ $(i \in I)$ are non-dominated solutions. Set $l_i = k,\ i \in I$.

Step 3: Shrink the matrix $D$ by removing indexes $I$ from the index set $I_{ij}$, i.e., $I_{ij} = I_{ij} \setminus I$.

Step 4: If $I_{ij}$ is not empty, go back to Step 2; otherwise stop the loop.

The index $l_i$ is called the *Pareto frontier index*. The smaller $l_i$ is, the better elitism $\boldsymbol{y}_i$ is, i.e., the solution with smaller Pareto frontier index is closer to the real Pareto frontier.

### 3.2 Dynamic matrix

The computation of the dynamic matrix $D$ is the core step of the dynamic non-dominated sorting, most of the computational cost is spent on this step. If use the dominance comparison to calculate the dynamic matrix $D$, $N(N-1)/2$ times of dominance comparisons are needed, which makes the computational complexity of the dynamic non-dominated sorting $O(mN^2)$. Through the transitivity of dominating can be used in practice, it still cannot reduce the computational complexity dramatically. In this subsection, we apply the idea presented in the dominance degree sorting [41], introducing a faster method to calculate the dynamic matrix.

The process of calculating the dominance matrix is as follows. Firstly, for each objective, we calculate a $N \times N$ comparison matrix $C_{f_k}$ $(1 \leq k \leq m)$ which records the comparison relationship of solutions on this objective. Take the first objective for example, suppose vector $\boldsymbol{w}^{f_1} = (\boldsymbol{p}_1^{f_1}, \boldsymbol{p}_2^{f_1}, \cdots, \boldsymbol{p}_N^{f_1})$, where $\boldsymbol{p}_j^{f_1}$ is the first objective value of solution $\boldsymbol{p}_i$ $(0 \leq i \leq N)$, then the entry of $C^{f_1}$ is

$$C_{ij}^{f_1} = \begin{cases} 1, \text{ if } \boldsymbol{p}_i^{f_1} \leq \boldsymbol{p}_j^{f_1}, \\ 0, \text{ otherwise.} \end{cases}$$

$C^{f_1}$ can be obtained very fast by sorting the members of $\boldsymbol{w}^{f_1}$.

Secondly, adding all the comparison matrix together to get a dominance degree matrix $C$, i.e.,

$$C = C^{f_1} + C^{f_2} + \cdots + C^{f_m}.$$

To eliminate the effect of these solutions with identical values for all objectives, we set the corresponding element of $C$ to be zero. For example, if $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ are identical respect to all objectives, we set $C_{ij} = 0$. Obviously, according to this rule, $C_{ii} = 0$ for all $i = 1, 2, \cdots, N$.

Thirdly, the elements of $C$ reflect the dominance relationship between solutions. For example, $\boldsymbol{p}_i$ dominates $\boldsymbol{p}_j$ means $\boldsymbol{p}_i^{f_k} \leq \boldsymbol{p}_j^{f_k}$ (i.e., $C_{ij}^{f_k} = 1$) for any $k \in \{1, 2, \cdots, m\}$, which yields $C_{ij} = m$. So we have $\boldsymbol{p}_i$ dominates $\boldsymbol{p}_j$ if and only if $C_{ij} = m$. The dynamic matrix can be directly obtained according to the dominance degree matrix $C$. That is, if $C_{ij} = m$, set $D_{ij} = 1$ and $D_{ji} = -1$; otherwise, reset $D_{ij} = 0$.

The pseudocode of calculating the dynamic matrix is presented in Algorithm 3

**Algorithm 3:** Dynamic matrix

**Input**: Selection pool $S(t)$ and its size $N = |S(t)|$, number of objectives $m$, an iteration counter $k = 1$.
**Output**: Dynamic matrix $D$.
Step 1: Let $C^{f_k}$ be a $N \times N$ zero matrix, let

$$\boldsymbol{w}^{f_k} = (\boldsymbol{p}_1^{f_1}, \boldsymbol{p}_2^{f_1}, \cdots, \boldsymbol{p}_N^{f_1}),$$

then let

$$C_{ij}^{f_1} = \begin{cases} 1, \text{ if } \boldsymbol{p}_i^{f_1} \leq \boldsymbol{p}_j^{f_1}, \\ 0, \text{ otherwise.} \end{cases}$$

Step 2: If $k + 1 \leq m$, then let $k = k + 1$, go back to step 1; otherwise, let

$$C = C^{f_1} + C^{f_2} + \cdots + C^{f_m}.$$

Step 3: Build dynamic matrix $D$. If $C_{ij} = m$, then let $D_{ij} = 1$ and $D_{ji} = -1$; otherwise, let $D_{ij} = 0$.
Step 4: Output the dynamic matrix $D$.

The main computational effort of the dynamic matrix is on the computation of the comparison matrixes $C^{f_k}$ $(k = 1, 2, \cdots, m)$. Intuitively, for each objective, function values are compared with each other, so $N(N-1)/2$ real number comparisons are needed, which makes the numerical complexity of dynamic matrix $O(mN^2)$. However, this process can be improved by applying a real number sequence sorting method, such as quick sort [41]. The computational complexity of the quick sort is $O(N \log N)$, so the computational complexity of dynamic matrix is $O(mN \log N)$.

### 3.3 Dynamic sorting genetic algorithm (DSGA)

In this subsection, we present a novel multi-objective genetic algorithm. Non-dominated sorting in this algorithm applies the dynamic non-dominated sort-

ing presented above, so we call this algorithm the dynamic sorting genetic algorithm, abbreviated as DSGA. The process of DSGA is presented in Fig. 1.

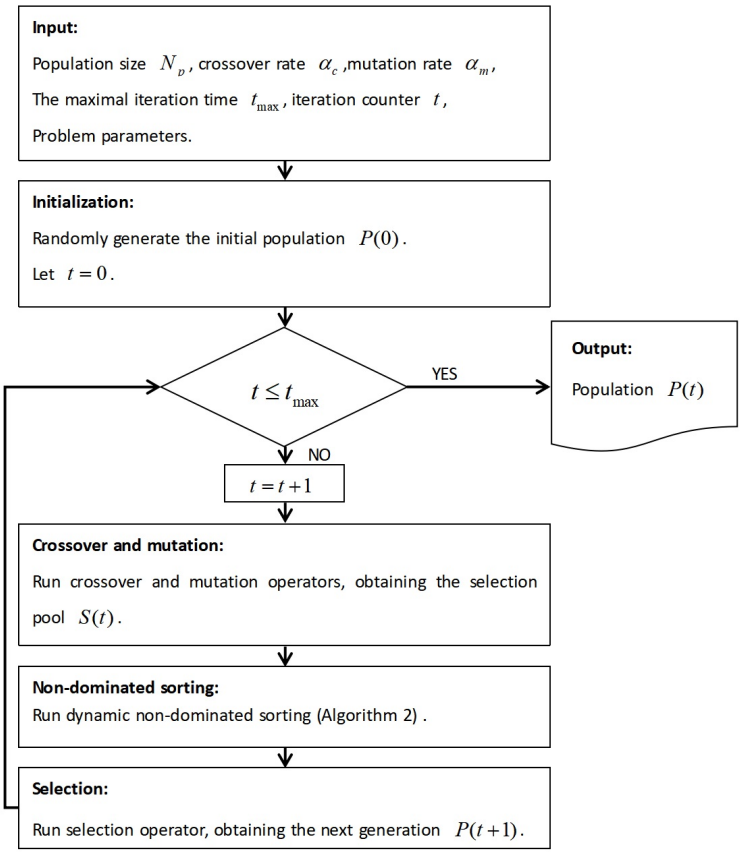



**Fig. 1** DSGA algorithm.

In the step of crossover and mutation, self-adaptive simulated binary crossover operator (SSBX) [4] and power mutation operator (PM) [6] are applied. The SSBX operator is a real-parameter recombination operator which is commonly used in the evolutionary algorithm (EA) literature. The operator involves a parameter which dedicates the spread of offspring solutions vis-a-vis that of the parent solutions. The PM operator is based on the power distribution. It is proved to have the same performance as the widely used non-uniform mutation operator [6].

For the selection step, the binary tournament selection operator used in NSGA-II is still applied in DSGA. The binary tournament selection operator

is mainly constituted by the fast non-dominated sorting and the crowded-comparison operator. In DSGA, we replace the fast non-dominated sorting by the dynamic non-dominated sorting proposed above.

## 4 Comparison of non-dominated sorting methods

In order to clarify the improvement of the dynamic non-dominated sorting (DNS), this subsection compares it with other non-dominated sorting methods. Except DNS proposed in this paper, four other referential non-dominated sorting methods are considered: the fast non-dominated sorting (FNS) [3], climbing sorting (CS) [20], deductive sorting (DS) [20] and the dominance degree non-dominated sorting (DDNS) [41]. FNS is one of the earliest Pareto sorting approaches. It is updated from the naive non-dominated sorting [2]. The computational complexity of FNS is $O(mN^2)$. CS follows dominating relationships between solutions and climbs up the graph toward the Pareto frontier. The key process of CS is to change the considering solution from any dominated one to dominating one until a non-dominated solution (at the current Pareto frontier) has been identified. The computational complexity of CS is $O(mN^2)$. DS accesses each solution based upon the natural order of the population. The candidate solution compares with all its following solutions but not with its previous ones. The average numerical complexity of DS is $O(mN^2)$, but in the best case that each Pareto frontier has only one solution, it decreases to $O(mN)$. DDNS is one of the state-of-the-art non-dominated sorting algorithms. Differently from the other non-dominated sorting algorithms, DDNS does not compare two solutions to identify dominating relationship. Instead, it constructs a comparison matrix which stores the relationship of all solutions with respect to each objective. Then a dominance degree matrix can be obtained by adding these comparison matrices together. Finally, Pareto ranking of the population can be obtained by analyzing the dominance degree matrix. If use quick sort in ranking each objective, the computational complexity of DDNS is $O(mN \log N)$.

We compare DNS with the other referential non-dominated sorting methods introduced above. For metrics of numerical performance, time consumption and number of comparisons are considered. Experiments are divided into three groups: (i) performance with respect to the variation of the the population size, (ii) performance with respects to the variation of the number of Pareto frontiers, and (iii) performance with respect to the variation of the number of objectives. We use the fixed features population generator [18] to generate test populations. This generator can generate test populations with certain features, such as having prefixed number of points, prefixed number of Pareto frontier and prefixed number of points in each Pareto frontier. The generated test populations are listed in Table 1 where $m$ stands for the number of objectives, $k$ stands for the number of Pareto frontiers and $N$ is a vector standing for the number of points in each Pareto frontiers. In the series (i) test populations, the number of objectives and Pareto frontiers are fixed, while the

**Table 1** Three series of test populations

| Series No. | Description | $m$ | $k$ | $N$ |
|---|---|---|---|---|
| Series (i) | fixed $m$ | 3 | 5 | $N = (12, 12, 12, 12, 12)$ |
| | fixed $k$ | 3 | 5 | $N = (14, 14, 14, 14, 14)$ |
| | various $N$ | 3 | 5 | $N = (16, 16, 16, 16, 16)$ |
| | | 3 | 5 | $N = (18, 18, 18, 18, 18)$ |
| | | 3 | 5 | $N = (20, 20, 20, 20, 20)$ |
| | | 3 | 5 | $N = (22, 22, 22, 22, 22)$ |
| | | 3 | 5 | $N = (24, 24, 24, 24, 24)$ |
| | | 3 | 5 | $N = (26, 26, 26, 26, 26)$ |
| | | 3 | 5 | $N = (28, 28, 28, 28, 28)$ |
| | | 3 | 5 | $N = (30, 30, 30, 30, 30)$ |
| Series (ii) | fixed $m$ | 3 | 1 | $N = (150)$ |
| | various $k$ | 3 | 2 | $N = (75, 75)$ |
| | $\sum N = 150$ | 3 | 3 | $N = (50, 50, 50)$ |
| | | 3 | 4 | $N = (37, 37, 37, 39)$ |
| | | 3 | 5 | $N = (30, 30, 30, 30, 30)$ |
| | | 3 | 6 | $N = (25, 25, 25, 25, 25, 25)$ |
| | | 3 | 7 | $N = (21, 21, 21, 21, 21, 21, 24)$ |
| | | 3 | 8 | $N = (18, 18, 18, 18, 18, 18, 18, 24)$ |
| | | 3 | 9 | $N = (16, 16, 16, 16, 16, 16, 16, 16, 22)$ |
| | | 3 | 10 | $N = (15, 15, 15, 15, 15, 15, 15, 15, 15, 15, )$ |
| Series (ii) | various $m$ | 2 | 5 | $N = (30, 30, 30, 30, 30)$ |
| | fixed $k$ | 3 | 5 | $N = (30, 30, 30, 30, 30)$ |
| | fixed $N$ | 4 | 5 | $N = (30, 30, 30, 30, 30)$ |
| | | 5 | 5 | $N = (30, 30, 30, 30, 30)$ |
| | | 6 | 5 | $N = (30, 30, 30, 30, 30)$ |
| | | 7 | 5 | $N = (30, 30, 30, 30, 30)$ |
| | | 8 | 5 | $N = (30, 30, 30, 30, 30)$ |
| | | 9 | 5 | $N = (30, 30, 30, 30, 30)$ |
| | | 10 | 5 | $N = (30, 30, 30, 30, 30)$ |

number of points in each Pareto frontier are even and increases from 12 to 30 with a step of 2. In the series (ii) test populations, the number of objectives is fixed, while the number of Pareto frontiers increases from 1 to 10 with step of 1, each test population has 150 points in total evenly distributed in each Pareto frontier. In the series (iii) test populations, the number of Pareto frontiers and number of points in each Pareto frontier are fixed, while the number of objectives arises from 2 to 10 with step of 1.

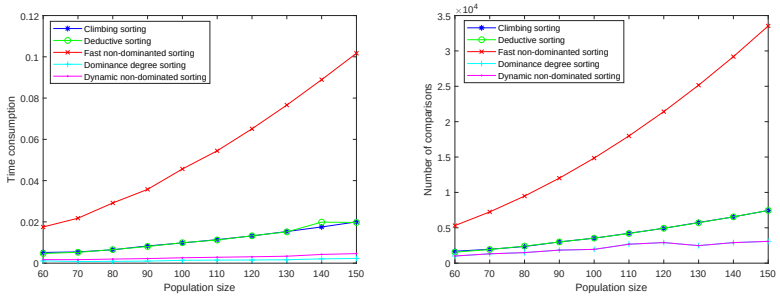
Results of the numerical experiments are illustrated in Fig. 2, 3 and 4. In Fig. 2(b), FNS needs much more comparisons than the other four methods, CS and DS need exact the same amount of comparisons, DDNS and DNS too but less than CS and DS. As the increase of population size, number of comparisons for FNS increases much faster than the other four method, then DS and CS, the increasing trend of DDNS and DNS are relative gentle. The same features demonstrated in Fig. 2(a) for time consumption. It is worth to note that DDNS spent slightly less time than DNS.

Fig. 3 shows that, as the number of Pareto frontiers increase, the time consumption and number of comparisons decrease for CS and DS. DDNS and DNS stay in a lower level stably, while FNS is stably appears in a very high level.

It is showed in Fig. 4 that the time consumption and the number of comparisons increase as the arise of the number of objectives. This is reasonable,

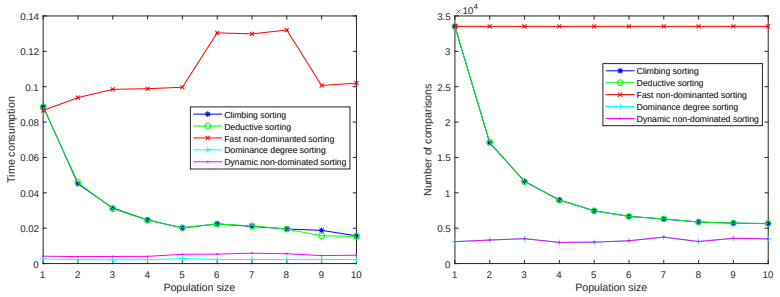
because the increase of the number of objectives must increases the number of comparisons, which in return increases the time consumption. But the increase rates are different. FNS has the steepest trend, CS and DS are less, DDNS and DNS only have slight increase.

In summary, the computation complexity of DNS keeps stable if the population size is fixed, and has a slight increase if the population size and number if objectives increase. This statement agrees with the theoretical complexity analysis of DNS. DNS outperforms FNS, CD and DS, and performs the same as DDNS which is considered as the most efficient non-dominated sorting method [41].



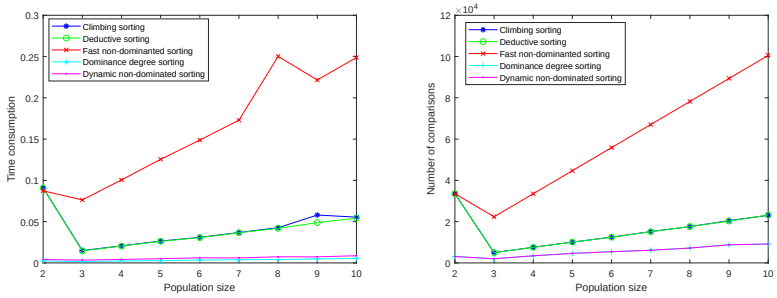

(a) Time consumptions (b) Number of comparisons

**Fig. 2** Numerical performance with respect to the variation of population size. In this test, the number of objectives $m = 3$, the number of Pareto frontiers $k = 5$, the population size is from 60 to 150 with increment 10, each Pareto frontier has the same number of solutions.




(a) Time consumptions (b) Number of comparisons

**Fig. 3** Numerical performance with respect to the variation of Pareto frontiers. In this test, the number of objectives $m = 3$, the number of Pareto frontiers increase from 1 to 10, the population size $N = 150$, each Pareto frontier has almost the same number of solutions.




(a) Time consumptions (b) Number of comparisons

**Fig. 4** Numerical performance with respect to the variation of of objectives. In this test, the number of objectives increase from 2 to 10, the number of Pareto frontiers $k = 5$, the population size is $N = 150$, and each Pareto frontier has the same number of solutions.

## 5 Numerical experiments

In this section, we investigate the numerical performance of DSGA. Firstly, we further compare the sorting methods FNS and DNS by embedding them into the same MOEA (NSGAII [3]). Secondly, we compare DSGA with some of the other popular MOEAs, including MOEA/D [36], SparseEA [28], PPS [8] and LSMOP [29]. Finally, we investigate the numerical performance of DSGA when scaling the number of variables.

### 5.1 Test problems

We use five series of test problems in the numerical experiments. They are ZDT series [42], DTLZ series [5], UF series [37], BT series [15] and LSMOP series [1]. Features of these test problems including number of objective functions $m$, number of dimensions $n$, variable bounds $X$ and references are demonstrated in Table 2. All test problems are scalable respect to the number of dimensions, the DTLZ series is also scalable respect to the number of objective functions. Numbers in the brackets are the number of dimensions or the number of objectives we set in our experiments. Details of the objective functions, referential Pareto frontiers and Pareto solutions refer to the references.

### 5.2 Referenial algorithms and parameter setting

We use five referential MOEAs in the numerical experiments. They are NSGAII [3], MOEAD [36], SparseEA [28], PPS [8] and LSMOF [29]. Among them, NSGAII is one of the most popular MOEA based on genetic algorithm. In the past decades, NSGAII got thousands of citations. MOEAD is a successful multiobjective optimization method based on decomposition, it is often used as a

**Table 2** Test problems

| Pro. | $m$ | $n$ | $X$ | Ref. |
|---|---|---|---|---|
| ZDT1 | 2 | $n(30)$ | $[0,1]^n$ | [3], [42] |
| ZDT2 | 2 | $n(30)$ | $[0,1]^n$ | [3], [42] |
| ZDT3 | 2 | $n(30)$ | $[0,1]^n$ | [3], [42] |
| ZDT4 | 2 | $n(30)$ | $[0,1]^n$ | [3], [42] |
| ZDT6 | 2 | $n(30)$ | $[0,1]^n$ | [3], [42] |
| DTLZ1 | $m(2)$ | $m+4(6)$ | $[0,1]^n$ | [5] |
| DTLZ2 | $m(2)$ | $m+9(11)$ | $[0,1]^n$ | [5] |
| DTLZ3 | $m(2)$ | $m+9(11)$ | $[0,1]^n$ | [5] |
| DTLZ4 | $m(2)$ | $m+9(11)$ | $[0,1]^n$ | [5] |
| DTLZ5 | $m(2)$ | $m+9(11)$ | $[0,1]^n$ | [5] |
| DTLZ6 | $m(2)$ | $m+9(11)$ | $[0,1]^n$ | [5] |
| DTLZ7 | $m(2)$ | $m+19(21)$ | $[0,1]^n$ | [5] |
| UF1 | 2 | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [37] |
| UF2 | 2 | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [37] |
| UF3 | 2 | $n(30)$ | $[0,1]^n$ | [37] |
| UF4 | 2 | $n(30)$ | $[0,1]\times[-2,2]^{n-1}$ | [37] |
| UF5 | 2 | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [37] |
| UF6 | 2 | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [37] |
| UF7 | 2 | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [37] |
| UF8 | 3 | $n(30)$ | $[0,1]^2\times[-2,2]^{n-2}$ | [37] |
| UF9 | 3 | $n(30)$ | $[0,1]^2\times[-2,2]^{n-2}$ | [37] |
| BT1 | 2 | $n(30)$ | $[0,1]^n$ | [15] |
| BT2 | 2 | $n(30)$ | $[0,1]^n$ | [15] |
| BT3 | 2 | $n(30)$ | $[0,1]^n$ | [15] |
| BT4 | 2 | $n(30)$ | $[0,1]^n$ | [15] |
| BT5 | 2 | $n(30)$ | $[0,1]^n$ | [15] |
| BT6 | 2 | $n(30)$ | $[0,1]^n$ | [15] |
| BT7 | 2 | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [15] |
| BT8 | 2 | $n(30)$ | $[0,1]^n$ | [15] |
| BT9 | 3 | $n(30)$ | $[0,1]^n$ | [15] |
| LSMOP1 | m(2) | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [14] |
| LSMOP2 | m(2) | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [14] |
| LSMOP3 | m(2) | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [14] |
| LSMOP4 | m(2) | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [14] |
| LSMOP5 | m(2) | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [14] |
| LSMOP6 | m(2) | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [14] |
| LSMOP7 | m(2) | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [14] |
| LSMOP8 | m(2) | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [14] |
| LSMOP9 | m(2) | $n(30)$ | $[0,1]\times[-1,1]^{n-1}$ | [14] |

standard in numerical experiments. SparseEA, PPS and LSMOF are three of the latest MOEAs, note that SparseEA and LSMOF are originally designed for solving large-scale multiobjective optimization problems. Among the five referential algorithms, NSGAII is used to verify the improvement of the sorting method DNS comparing with the traditional one FNS, while the other four referential algorithm are used to investigate the numerical performance of the proposed method DSGA. The implementation of these algorithms are based on the PlatEMO [27].

For the sake of fair comparison, parameters for all algorithms are uniformly set as far as possible. To be specific, the population size is set to be 100, the

maximum number of objective function evaluations is set to be 100000, the maximum number of iterations is set to be 500. The maximum number of objective function evaluations and iterations are taken as stop criteria for all algorithms. In order to achieve statistic performance, all the test are run 30 times independently, and the mean and standard deviation of the performance metrics are recorded. The other parameters for certain algorithms are set as the default in PlatEMO.

### 5.3 Performance metrics

Many performance metrics have been proposed to evaluate the numerical performance of MOGAs [40]. There are two goals for evaluation metrics: (i) measure the convergence of the obtained Pareto frontier, and (ii) measure the diversity of the obtained Pareto frontier.

We use the performance metric IGD [17] to evaluate the numerical performance. Suppose that $P^*$ is a set of uniformly distributed points belonging to the real Pareto frontier. It can be taken as a standard representation of the real Pareto frontier. Let $A$ be a set of solutions obtained by a certain solver, then IGD is defined as the average distance from $P^*$ to $A$:

$$IGD(A, P^*) = \frac{\sum_{v \in P^*} d(\boldsymbol{v}, A)}{|P^*|},$$

where $d(v, A)$ is the minimum Euclidean distances between $v$ and the points in $A$, i.e.,

$$d(\boldsymbol{v}, A) = \min_{y \in A} \parallel \boldsymbol{v} - \boldsymbol{y} \parallel \quad \boldsymbol{v} \in P^*.$$

In fact, $P^*$ is a sample set of the real Pareto frontier. If $|P^*|$ is large enough to approximate the Pareto frontier very well, $IGD(A, P^*)$ could measure both the diversity and convergence of $A$. This is also the reason that we choose IGD as the evaluation metric for this paper. A smaller $IGD(A, P^*)$ means the set $A$ is closer to the real Pareto frontier and has better diversity.

Another well-known performance metric is the hypervolume value (HV) [32] of the obtained non-dominated solutions. The calculation of HV value do not need a referential Pareto frontier $P^*$, but is more complicated than the calculation of IGD value, especially when the number of objectives is large. In this paper, we use IGD instead of HV since the referential Pareto frontiers of the test problems are all known and evenly distributed ones can be generated using PlatEMO.

### 5.4 NSGAII with FNS and DNS

In this subsection, we compare FNS and DNS by embedded them into the same MOEA. Since FNS is originally used in NSGAII, we replace FNS in NSGAII by DNS to build a new MOEA. In the following, we call the NSGAII

with FNS NSGAII-FNS, and call the NSGAII with DNS NSGAII-DNS. Note that NSGAII-FNS and NSGAII-DNS are only different in the non-dominated sorting method.

In order to achieve fair competition and statistical performance, all the tests are run for 30 times independently, and the mean and standard deviation of two performance metrics, CPU time and IGD value, are recorded. In the following tables, the best record for a certain performance metric is marked as in grey cell. Besides, the Wilcoxon rank sum test with a significant level of 0.05 is adopted to perform statistical analysis the experimental results, where the symbols "+","-" and "=" indicate that the result by NSGAII-FNS are significantly better, significantly worse and statistically similar to that obtained by NSGAII-DNS, respectively.

As shown in Table 3, NSGAII-DNS spends significantly less CPU time than NSGAII-FNS on all test problems, which further verifies that DNS is faster than FNS. As for the IGD value, Table 3 shows that 38 out of the total 39 test problems are statistically similar. This is reasonable for that NSGAII-FNS and NSGAII-DNS are only different in the non-dominated sorting method, which affects the CPU time but not the final solutions.

In Figure 5, we demonstrate the decrease curve of IGD of the first test problem in each series. Because inside a series, the test problems have more or less the same structure, the IGD curve of one problem can represent the others. For each test problem, 10 samples of IGD value are taken evenly from 1000 to 10000 times of objective function evaluations. From Figure 5, the IGD value of NDGAII-DNS and NSGAII-FNS converge to almost the same value for problems ZDT1 and DTLZ1. Figures 5(d) and 5(e) show that NSGAII-DNS outperforms NSGAII-FNS for problems BT1 and LSMOP1. For problem UF1, Figure 5(c) shows that NSGAII-FNS outperforms NSGAII-DNS.

### 5.5 Compare DSGA with other MOEAs

In this subsection, we compare the numerical performance of the proposed method DSGA with other existing MOEAs. The referential MOEAs are MOEAD, sparseEA, PPS and LSMOF. We investigate average and standard deviation of CPU time and IGD value. Each test problem runs 30 time independently. The statistical results are shown in Table 4 and 5. Grey cell represents the best values of a certain performance metric.

Table 4 illustrates the CPU time of comparing DSGA with other MOEAs. It can be seen that DSGA outperforms the other MOEAs on all test problems except LSMOF on problem ZDT2. This indicates that DNS indeed accelerates the speed of non-dominated sorting. Table 5 illustrates the IGD value of comparing DSGA with other MOEAs. It can be observed that SparseEA performs the best on the ZDT series, PPS performs the best on most problems of the LSMOP series. The proposed DSGA outperforms the others on DTLZ6, UF1-2,UF6-7,BT1-5 and BT9. Statistically, in terms of the Wilcoxon rank sum test, the proportion of the test instances where DSGA performs significantly better

**Table 3** Numerical comparison between NSGAII-FNS and NSGAII-DNS

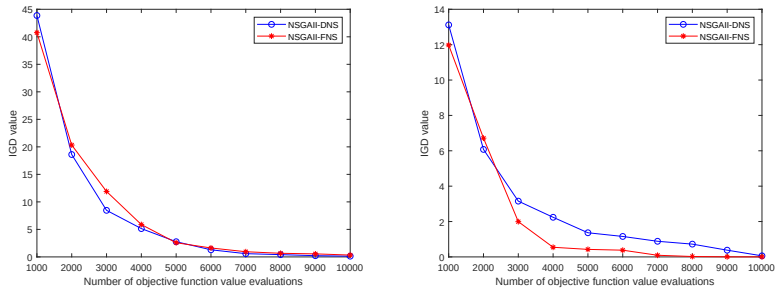
| Pro. | CPU time | | IGD | |
|---|---|---|---|---|
| | NSGAII-FNS | NSGAII-DNS | NSGAII-FNS | NSGAII-DNS |
| ZDT1 | 1.5586e+0 (9.35e-2) - | 6.1376e-1 (2.74e-2) | 2.1861e-1 (1.28e-1) = | 1.8775e-1 (1.04e-1) |
| ZDT2 | 2.2492e+0 (1.31e-1) - | 7.6010e-1 (4.65e-2) | 5.1763e-1 (1.51e-1) = | 5.3278e-1 (1.30e-1) |
| ZDT3 | 1.4290e+0 (1.08e-1) - | 6.1202e-1 (4.46e-2) | 1.6500e-1 (1.01e-1) = | 1.4404e-1 (1.05e-1) |
| ZDT4 | 2.0416e+0 (1.17e-1) - | 7.0622e-1 (4.60e-2) | 1.5368e+1 (3.24e+0) = | 1.5371e+1 (3.90e+0) |
| ZDT6 | 1.7983e+0 (7.56e-2) - | 6.7817e-1 (4.39e-2) | 3.7633e+0 (3.69e-1) = | 3.7927e+0 (3.27e-1) |
| DTLZ1 | 1.8721e+0 (6.99e-2) - | 6.0092e-1 (4.22e-2) | 5.5622e-2 (1.09e-1) = | 6.6503e-2 (1.26e-1) |
| DTLZ2 | 9.4927e-1 (6.63e-2) - | 5.5882e-1 (5.62e-2) | 5.1195e-3 (1.86e-4) = | 5.0904e-3 (1.30e-4) |
| DTLZ3 | 1.8266e+0 (9.17e-2) - | 6.3956e-1 (4.41e-2) | 5.2642e+0 (2.78e+0) = | 6.1121e+0 (3.33e+0) |
| DTLZ4 | 1.3209e+0 (5.86e-1) - | 6.0191e-1 (1.54e-1) | 1.7712e-1 (3.17e-1) = | 1.5251e-1 (3.00e-1) |
| DTLZ5 | 9.4323e-1 (6.21e-2) - | 5.2869e-1 (4.75e-2) | 5.0690e-3 (1.55e-4) = | 5.1468e-3 (1.71e-4) |
| DTLZ6 | 1.1294e+0 (3.61e-2) - | 5.0044e-1 (2.78e-2) | 5.7473e-3 (3.52e-4) = | 6.5599e-3 (5.22e-3) |
| DTLZ7 | 1.1807e+0 (1.35e-1) - | 5.3100e-1 (1.90e-2) | 7.5917e-3 (8.00e-4) = | 7.3846e-3 (8.12e-4) |
| UF1 | 1.4560e+0 (1.99e-1) - | 6.5256e-1 (1.17e-1) | 1.1816e-1 (2.58e-2) = | 1.3276e-1 (3.72e-2) |
| UF2 | 1.1261e+0 (5.45e-2) - | 5.4319e-1 (2.94e-2) | 6.4574e-2 (9.35e-3) = | 6.7811e-2 (1.81e-2) |
| UF3 | 1.0534e+0 (5.06e-2) - | 5.5701e-1 (4.76e-2) | 7.4628e-2 (3.19e-3) = | 7.4178e-2 (3.33e-3) |
| UF4 | 1.5129e+0 (1.07e-1) - | 6.0604e-1 (3.55e-2) | 3.7298e-1 (5.60e-2) = | 3.7393e-1 (4.95e-2) |
| UF5 | 1.7999e+0 (1.25e-1) - | 6.2569e-1 (4.05e-2) | 6.8505e-1 (1.75e-1) = | 6.9100e-1 (2.15e-1) |
| UF6 | 1.6990e+0 (1.28e-1) - | 6.7698e-1 (6.86e-2) | 3.6041e-1 (1.18e-1) = | 3.8024e-1 (1.34e-1) |
| UF7 | 1.4960e+0 (2.59e-1) - | 5.7436e-1 (2.11e-2) | 2.2450e-1 (1.43e-1) = | 2.0384e-1 (1.51e-1) |
| UF8 | 1.0871e+0 (8.33e-2) - | 6.2884e-1 (2.60e-2) | 3.0100e-1 (3.12e-2) = | 2.9188e-1 (2.31e-2) |
| UF9 | 1.2634e+0 (1.11e-1) - | 6.6747e-1 (3.65e-2) | 4.3603e-1 (8.85e-2) = | 4.0026e-1 (5.97e-2) |
| BT1 | 1.1270e+0 (6.22e-2) - | 5.6646e-1 (3.22e-2) | 3.0879e+0 (1.59e-1) = | 3.0515e+0 (1.05e-1) |
| BT2 | 1.2267e+0 (5.93e-2) - | 6.0572e-1 (3.66e-2) | 6.9969e-1 (4.03e-2) = | 6.8137e-1 (3.77e-2) |
| BT3 | 1.3580e+0 (1.26e-1) - | 5.9054e-1 (4.20e-2) | 2.2955e+0 (1.51e-1) = | 2.3211e+0 (1.49e-1) |
| BT4 | 1.2578e+0 (7.16e-2) - | 5.7150e-1 (3.69e-2) | 2.1949e+0 (1.69e-1) = | 2.1776e+0 (1.83e-1) |
| BT5 | 1.1971e+0 (6.07e-2) - | 5.7392e-1 (3.52e-2) | 3.0398e+0 (2.06e-1) = | 3.0689e+0 (1.68e-1) |
| BT6 | 1.8117e+0 (7.17e-2) - | 6.5935e-1 (3.27e-2) | 4.1243e-1 (2.69e-1) = | 4.9615e-1 (3.53e-1) |
| BT7 | 1.8247e+0 (1.04e-1) - | 6.2203e-1 (5.81e-2) | 4.6484e-1 (3.09e-1) = | 4.3477e-1 (2.36e-1) |
| BT8 | 2.0006e+0 (2.18e-1) - | 8.5726e-1 (3.58e-2) | 2.6431e+0 (4.86e-1) = | 2.8699e+0 (5.37e-1) |
| BT9 | 1.2621e+0 (7.13e-2) - | 8.5001e-1 (4.10e-2) | 2.5291e+0 (1.16e-1) = | 2.4768e+0 (1.14e-1) |
| LSMOP1 | 1.7803e+0 (7.96e-2) - | 7.8256e-1 (4.81e-2) | 2.9050e-1 (6.35e-2) = | 2.8214e-1 (4.96e-2) |
| LSMOP2 | 1.4015e+0 (5.97e-2) - | 7.8288e-1 (6.04e-2) | 1.6666e-1 (1.13e-2) = | 1.6229e-1 (1.28e-2) |
| LSMOP3 | 1.8770e+0 (8.89e-2) - | 8.0653e-1 (5.16e-2) | 2.7901e+0 (1.51e+0) = | 3.1558e+0 (2.00e+0) |
| LSMOP4 | 1.5460e+0 (5.81e-2) - | 7.6719e-1 (4.12e-2) | 2.1587e-1 (5.48e-2) = | 2.1142e-1 (5.21e-2) |
| LSMOP5 | 1.8922e+0 (1.45e-1) - | 7.9718e-1 (3.78e-2) | 6.4383e-1 (1.25e-1) = | 6.0490e-1 (1.68e-1) |
| LSMOP6 | 1.4741e+0 (7.61e-2) - | 8.1068e-1 (1.07e-1) | 3.4260e-1 (3.36e-2) = | 3.4841e-1 (2.96e-2) |
| LSMOP7 | 2.3373e+0 (1.87e-1) - | 9.7543e-1 (5.06e-2) | 1.6921e+1 (1.64e+1) = | 1.8387e+1 (1.74e+1) |
| LSMOP8 | 1.8467e+0 (1.05e-1) - | 8.2289e-1 (4.58e-2) | 5.4191e-1 (1.44e-1) = | 5.5080e-1 (1.14e-1) |
| LSMOP9 | 1.8192e+0 (1.39e-1) - | 8.3844e-1 (4.11e-2) | 2.0432e+0 (7.00e-1) + | 2.4735e+0 (7.29e-1) |
| +/-/= | 0/39/0 | | 1/0/38 | |

than MOEAD, SparseEA, PPS and LSMOF is 25/39, 25/39, 26/39 and 19/39, respectively.

For further observation, Figure 6 depicts the decrease trajectory of IGD value on the first case of each test problem series. Ten samples are recorded after every 1000 times of objective function evaluation. It can be seen from the figures that the IGD value of DSGA may start at a high value, but can dramatically decrease to a promising zone. For all the test problems, IGD value obtained by DSGA can converge to a competitively small value. For problem BT1, DSGA outperforms the other MOEAs.

### 5.6 Scalability with respect to the number of variables

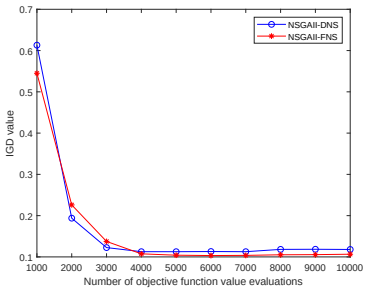
In this subsection, we investigate the performance of DSGA with respect to the scaling of decision variables. In this experiment, only SparseEA and LSMOF are selected as referential algorithms for they are originally designed for large-scale multi-objective optimization, test problems are also restricted in the LSMOP series [1]. There are 9 problems in the LSMOP series, the number of subcomponent in each variable group $n_k$ is set to be 5, the number of objectives is set to be 2 and 3, and the number of decision variables is set to be 10, 30, 50, 100, 300, 500, and 1000. Test problems in the LSMOP series can be categories



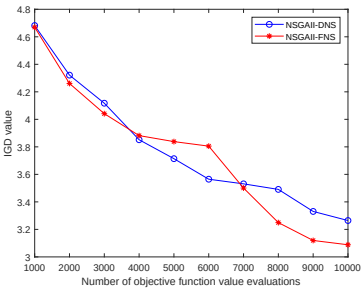

(a) IGD of ZDT1



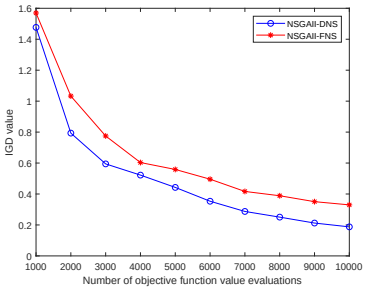

(b) IGD of DTLZ1




(c) IGD of UF1




(d) IGD of BT1




(e) IGD of LSMOP1

**Fig. 5** The decreasing trend of IGD.

into three groups (LSMOP 1-4, LSMOP5-8 and LSMOP9) according to their structure of Pareto frontiers. So for the sake of saving space, we demonstrate the performance of LSMOP1, LSMOP5 and LSMOP9 for 2 objectives and the performance of LSMOP2, LSMOP6 and LSMOP9 for 3 objectives.

Table 6 illustrate the CPU time of SparseEA, LSMOP and DSGA on solving 2-objective LSMOP1, LSMOP5 and LSMOP9. From the table, DSGA outperforms SparseEA on all cases except LSMOP9 with 10 decision variables. Comparing betwween LSMOF and DSGA, for LSMOP1 and LSMOP9,

**Table 4** CPU time of comparing DSGA with other MOEAs

| Pro. | MOEAD | SparseEA | PPS | LSMOF | DSGA |
|---|---|---|---|---|---|
| ZDT1 | 2.4248e+0 (3.71e-1) - | 1.2606e+0 (3.17e-1) - | 2.4444e+0 (2.40e-1) - | 1.0326e+0 (1.60e-1) - | 8.0745e-1 (1.69e-1) |
| ZDT2 | 2.2201e+0 (6.36e-2) - | 1.1076e+0 (6.24e-2) - | 2.1664e+0 (8.21e-2) - | 7.9564e-1 (5.71e-2) + | 9.2181e-1 (1.23e-1) |
| ZDT3 | 2.1949e+0 (6.33e-2) - | 1.1977e+0 (1.66e-1) - | 2.2556e+0 (6.10e-2) - | 7.6536e-1 (6.36e-2) - | 7.5482e-1 (3.12e-1) |
| ZDT4 | 2.2348e+0 (5.48e-2) - | 9.8576e-1 (4.85e-2) - | 2.2587e+0 (4.46e-2) - | 7.8026e-1 (8.39e-2) - | 6.9213e-1 (4.21e-2) |
| ZDT6 | 2.2169e+0 (5.67e-2) - | 1.0164e+0 (4.65e-2) - | 2.1928e+0 (5.80e-2) - | 7.4497e-1 (9.11e-2) - | 6.4645e-1 (3.29e-2) |
| DTLZ1 | 2.3653e+0 (2.29e-1) - | 8.0046e-1 (4.37e-2) - | 2.4402e+0 (2.16e-1) - | 9.8658e-1 (7.33e-2) - | 5.8128e-1 (4.75e-2) |
| DTLZ2 | 2.2210e+0 (5.94e-2) - | 6.9656e-1 (3.76e-2) - | 2.2947e+0 (6.68e-2) - | 1.0473e+0 (7.99e-2) - | 5.4538e-1 (8.15e-2) |
| DTLZ3 | 2.2914e+0 (1.06e-1) - | 1.0659e+0 (1.26e-1) - | 2.2898e+0 (6.21e-2) - | 9.2339e-1 (6.89e-2) - | 6.2179e-1 (3.90e-2) |
| DTLZ4 | 2.2767e+0 (9.81e-2) - | 7.9015e-1 (1.08e-1) - | 2.2770e+0 (6.11e-2) - | 9.6959e-1 (1.86e-1) - | 5.4853e-1 (1.29e-1) |
| DTLZ5 | 2.2900e+0 (5.60e-2) - | 6.9504e-1 (3.71e-2) - | 2.3332e+0 (5.65e-2) - | 1.0034e+0 (1.39e-1) - | 5.0713e-1 (4.08e-2) |
| DTLZ6 | 2.3338e+0 (7.34e-2) - | 9.2624e-1 (5.13e-2) - | 2.3897e+0 (7.50e-2) - | 7.5826e-1 (1.00e-1) - | 4.7041e-1 (2.94e-2) |
| DTLZ7 | 2.2749e+0 (6.05e-2) - | 9.6761e-1 (4.27e-2) - | 2.3028e+0 (6.75e-2) - | 9.0487e-1 (5.70e-2) - | 5.2903e-1 (2.75e-2) |
| UF1 | 2.3682e+0 (6.64e-2) - | 1.3797e+0 (9.50e-2) - | 2.3612e+0 (5.59e-2) - | 8.1320e-1 (1.05e-1) - | 5.6727e-1 (3.76e-2) |
| UF2 | 2.5270e+0 (8.58e-2) - | 1.3113e+0 (3.16e-2) - | 2.5416e+0 (5.10e-2) - | 6.8092e-1 (7.65e-2) - | 5.8093e-1 (3.54e-2) |
| UF3 | 2.3783e+0 (4.95e-2) - | 1.2820e+0 (4.32e-2) - | 2.4021e+0 (4.78e-2) - | 9.7097e-1 (5.29e-2) - | 5.2081e-1 (2.12e-2) |
| UF4 | 2.4370e+0 (8.26e-2) - | 1.4366e+0 (7.26e-2) - | 2.4918e+0 (1.02e-1) - | 8.7897e-1 (6.44e-2) - | 6.1693e-1 (4.22e-2) |
| UF5 | 2.4928e+0 (2.00e-1) - | 1.5469e+0 (9.40e-2) - | 2.5222e+0 (1.44e-1) - | 9.2592e-1 (1.14e-1) - | 6.7507e-1 (5.90e-2) |
| UF6 | 2.3963e+0 (7.68e-2) - | 1.4399e+0 (9.12e-2) - | 2.3826e+0 (5.41e-2) - | 8.4192e-1 (1.49e-1) - | 6.9199e-1 (1.36e-1) |
| UF7 | 2.3751e+0 (6.08e-2) - | 1.4488e+0 (1.36e-1) - | 2.3881e+0 (5.76e-2) - | 8.6274e-1 (1.32e-1) - | 5.7545e-1 (2.55e-2) |
| UF8 | 2.4417e+0 (6.02e-2) - | 1.3527e+0 (5.02e-2) - | 2.4761e+0 (6.29e-2) - | 2.3228e+0 (6.06e-1) - | 6.6578e-1 (4.44e-2) |
| UF9 | 2.5273e+0 (7.70e-2) - | 1.4667e+0 (5.66e-2) - | 2.4647e+0 (5.99e-2) - | 2.8312e+0 (9.41e-1) - | 7.7431e-1 (6.52e-2) |
| BT1 | 2.4554e+0 (1.67e-1) - | 1.0635e+0 (1.19e-1) - | 2.3387e+0 (5.65e-2) - | 6.8618e-1 (5.94e-2) - | 5.8412e-1 (3.87e-2) |
| BT2 | 2.5640e+0 (1.33e-1) - | 1.1309e+0 (1.24e-1) - | 2.5124e+0 (8.97e-2) - | 7.2481e-1 (6.58e-2) - | 6.0053e-1 (2.49e-2) |
| BT3 | 2.3823e+0 (5.54e-2) - | 1.2020e+0 (1.24e-1) - | 2.4568e+0 (9.43e-2) - | 7.8520e-1 (7.97e-2) - | 6.4199e-1 (4.50e-2) |
| BT4 | 2.6359e+0 (9.40e-2) - | 1.1343e+0 (1.24e-1) - | 2.5962e+0 (4.94e-2) - | 7.6298e-1 (8.04e-2) - | 6.2017e-1 (4.03e-2) |
| BT5 | 2.3708e+0 (6.03e-2) - | 1.0990e+0 (1.19e-1) - | 2.3474e+0 (5.99e-2) - | 7.1219e-1 (6.56e-2) - | 5.8593e-1 (3.93e-2) |
| BT6 | 2.5323e+0 (8.65e-2) - | 1.4419e+0 (4.86e-2) - | 2.4347e+0 (6.82e-2) - | 1.2308e+0 (4.58e-2) - | 6.4198e-1 (3.07e-2) |
| BT7 | 2.3762e+0 (8.83e-2) - | 1.4630e+0 (5.38e-2) - | 2.3837e+0 (9.31e-2) - | 9.1521e-1 (6.91e-2) - | 6.3349e-1 (3.48e-2) |
| BT8 | 2.4196e+0 (7.83e-2) - | 1.4312e+0 (5.03e-2) - | 2.3987e+0 (5.67e-2) - | 1.2199e+0 (5.56e-2) - | 6.5302e-1 (3.27e-2) |
| BT9 | 2.4513e+0 (6.79e-2) - | 1.3206e+0 (8.96e-2) - | 2.4379e+0 (5.72e-2) - | 2.3621e+0 (4.10e-1) - | 6.4692e-1 (3.95e-2) |
| LSMOP1 | 2.7658e+0 (1.09e-1) - | 1.5813e+0 (2.12e-1) - | 2.7176e+0 (7.99e-2) - | 7.2389e-1 (8.10e-2) - | 6.1531e-1 (4.20e-2) |
| LSMOP2 | 2.8977e+0 (1.07e-1) - | 1.4151e+0 (6.91e-2) - | 2.9639e+0 (2.16e-1) - | 1.1036e+0 (7.51e-2) - | 6.0759e-1 (3.80e-2) |
| LSMOP3 | 2.9789e+0 (1.08e-1) - | 1.7869e+0 (9.27e-2) - | 2.7987e+0 (7.02e-2) - | 1.2356e+0 (1.09e-1) - | 6.3825e-1 (4.92e-2) |
| LSMOP4 | 2.8890e+0 (1.32e-1) - | 1.4609e+0 (6.74e-2) - | 2.8595e+0 (7.33e-2) - | 1.1099e+0 (7.52e-2) - | 5.8999e-1 (4.00e-2) |
| LSMOP5 | 2.6880e+0 (6.46e-2) - | 1.5153e+0 (2.77e-1) - | 2.7329e+0 (8.26e-2) - | 7.5487e-1 (9.45e-2) - | 6.2768e-1 (4.36e-2) |
| LSMOP6 | 3.0901e+0 (8.43e-2) - | 1.1073e+0 (7.86e-2) - | 3.2046e+0 (9.89e-2) - | 9.2110e-1 (7.99e-2) - | 6.5120e-1 (6.72e-2) |
| LSMOP7 | 3.1335e+0 (2.19e-1) - | 1.6541e+0 (8.46e-2) - | 2.8912e+0 (9.70e-2) - | 1.3253e+0 (7.17e-2) - | 7.1027e-1 (3.82e-2) |
| LSMOP8 | 2.8519e+0 (1.03e-1) - | 1.5242e+0 (2.49e-1) - | 2.8418e+0 (6.71e-2) - | 7.5233e-1 (8.46e-2) - | 6.4570e-1 (3.56e-2) |
| LSMOP9 | 2.7927e+0 (7.63e-2) - | 1.6972e+0 (9.73e-2) - | 2.7483e+0 (8.01e-2) - | 1.2963e+0 (7.09e-2) - | 6.4567e-1 (3.94e-2) |
| +/-/= | 0/39/0 | 0/39/0 | 0/39/0 | 1/38/0 | |

DSGA is faster when dimension is small and LSMOF is faster when dimension becomes large. However, for LSMOP5, DSGA is always faster than LSMOF except when dimension is 10. Statistically, the proportion of test instances where DSGA performs significantly faster than SparseEA and LSMOF is 10/21 and 14/21, respectively. Table 8 illustrates the CPU time of the three compared MOEAs on solving 3-objective LSMOP2, LSMOP6 and LSMOP9. From the table, DSGA is still faster than SparseEA for all cases except when dimension is 10. Except LSMOP9 with dimension equals 100, 500 and 1000, DSGA is also faster than LSMOF. In terms of the Wilcoxon rank test, the proportion of test cases where DSGA performs significant faster than SparseEA and LSMOF are both 18/21. Table 6 and 8 verify that DSGA outperforms SparseEA and LSMOP in terms of CPU time.

Table 7 and 9 illustrate the IGD value of solving 2-objective LSMOP1, LSMOP5, LSMOP9 and 3-objective LSMOP2, LSMOP5, LSMOP9, respectively. From Table 7, DSGA outperforms the other two MOEAs on problem LSMOP1 with dimension of 30, 50 and problem LSMOP5 with dimensions of 30, 50, 100. From Table 9, DSGA outperforms the other two MOEAs only on problem LSMOP2. Table 7 and 9 reveals that, for solving large-scale MOPs, the solution obtained by DSGA is not as good as LSMOF and SparseEA. This is reasonable for LSMOF and SparseEA are originally designed for large-scale

**Table 5** IGD value of comparing DSGA with other MOEAs

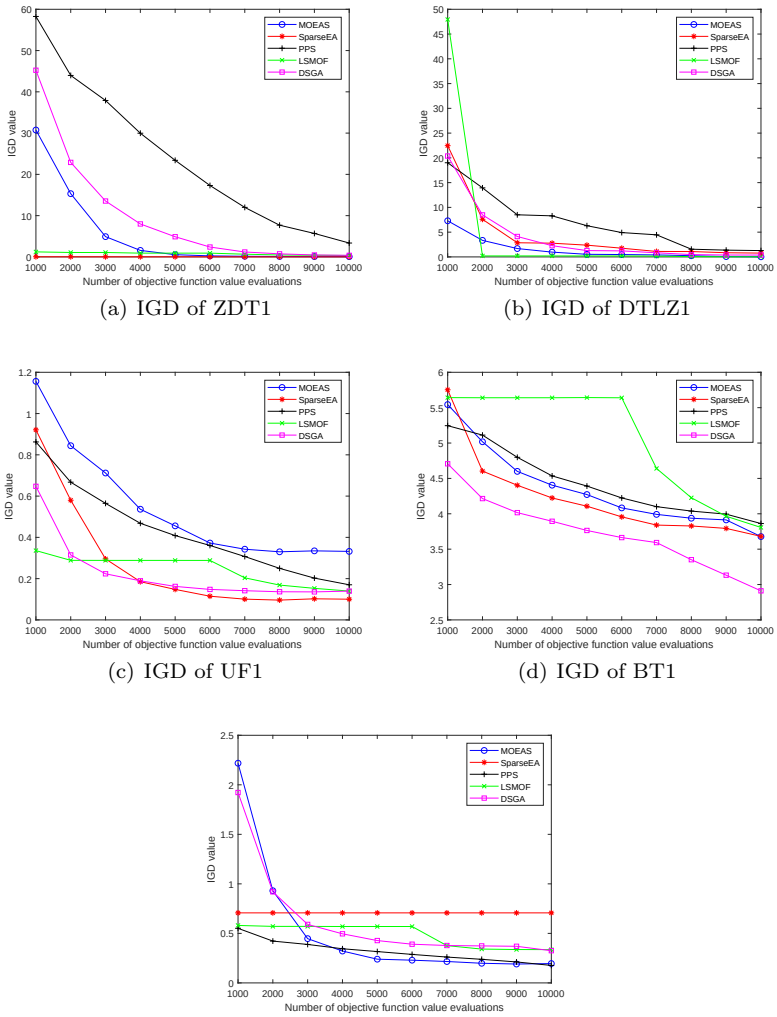
| Pro. | MOEAD | SparseEA | PPS | LSMOF | DSGA |
|---|---|---|---|---|---|
| ZDT1 | 1.6139e-1 (5.96e-2) = | 9.5040e-3 (3.26e-3) + | 3.8789e+0 (2.40e+0) - | 2.3086e-1 (1.19e-1) = | 1.8561e-1 (1.08e-1) |
| ZDT2 | 3.3019e-1 (1.73e-1) + | 1.0662e-2 (3.83e-3) + | 5.5348e+0 (2.98e+0) - | 3.2781e-1 (9.80e-2) + | 5.7252e-1 (9.72e-2) |
| ZDT3 | 2.4446e-1 (1.19e-1) - | 2.7550e-2 (2.21e-2) + | 5.8357e+0 (3.64e+0) - | 3.1681e-1 (1.48e-1) - | 1.7268e-1 (8.69e-2) |
| ZDT4 | 1.3334e+1 (3.53e+0) + | 1.7467e-2 (5.51e-3) + | 5.4397e+1 (1.07e+1) - | 4.0493e+0 (9.90e+0) + | 1.6125e+1 (3.86e+0) |
| ZDT6 | 2.8477e+0 (6.44e-1) + | 5.5024e-3 (1.72e-3) + | 9.0731e+0 (1.36e+0) - | 1.5615e+0 (4.07e-1) + | 3.9638e+0 (3.12e-1) |
| DTLZ1 | 2.0906e-1 (2.57e-1) - | 7.3819e-1 (4.62e-1) - | 1.1416e+0 (1.70e+0) - | 1.2500e-2 (9.36e-3) = | 9.5671e-2 (1.96e-1) |
| DTLZ2 | 4.3493e-3 (1.77e-4) + | 5.5388e-3 (4.08e-4) - | 8.0584e-3 (7.30e-4) - | 5.6133e-3 (4.54e-4) - | 5.1100e-3 (1.65e-4) |
| DTLZ3 | 1.1337e+1 (6.10e+0) - | 1.8369e+1 (7.82e+0) - | 3.0684e+1 (2.76e+1) - | 1.1889e-1 (9.17e-2) + | 6.2962e+0 (3.63e+0) |
| DTLZ4 | 2.5036e-1 (3.54e-1) - | 2.2649e-1 (3.43e-1) - | 3.4281e-2 (1.34e-1) + | 5.2231e-3 (1.91e-4) = | 1.0337e-1 (2.55e-1) |
| DTLZ5 | 4.3631e-3 (1.96e-4) + | 5.4735e-3 (2.62e-4) - | 7.9785e-3 (8.32e-4) - | 5.6403e-3 (4.03e-4) - | 5.0887e-3 (1.71e-4) |
| DTLZ6 | 1.3788e-2 (4.71e-2) - | 9.5649e-3 (5.34e-3) - | 7.3630e-3 (8.17e-4) - | 2.6455e-2 (2.49e-2) - | 5.5924e-3 (2.27e-4) |
| DTLZ7 | 1.7615e-1 (1.94e-1) - | 5.5259e-3 (6.91e-4) + | 4.0818e-1 (1.32e-1) - | 2.1081e-1 (2.23e-1) = | 7.3761e-3 (7.75e-4) |
| UF1 | 3.2405e-1 (1.15e-1) - | 1.5632e-1 (5.37e-2) - | 1.5424e-1 (4.20e-2) - | 1.4925e-1 (1.43e-2) - | 1.2148e-1 (2.69e-2) |
| UF2 | 1.6433e-1 (5.37e-2) - | 8.8892e-2 (5.65e-3) - | 7.4596e-2 (2.80e-2) = | 7.5469e-2 (6.13e-3) - | 6.7764e-2 (1.41e-2) |
| UF3 | 1.2548e-1 (5.52e-3) - | 8.9363e-2 (4.55e-3) - | 9.6603e-2 (7.59e-3) - | 6.6813e-2 (2.27e-3) + | 7.5017e-2 (3.37e-3) |
| UF4 | 3.2637e-1 (1.80e-2) + | 8.4030e-1 (5.65e-16)- | 2.8188e-1 (2.90e-2) + | 3.3030e-1 (6.38e-3) + | 3.8324e-1 (4.53e-2) |
| UF5 | 1.2835e+0 (2.19e-1) - | 1.0131e+0 (3.83e-1) - | 1.7282e+0 (2.74e-1) - | 1.0684e+0 (2.01e-1) - | 6.8080e-1 (1.88e-1) |
| UF6 | 6.8674e-1 (3.10e-1) - | 8.1865e-1 (2.92e-1) - | 6.4707e-1 (1.29e-1) - | 7.6842e-1 (1.55e-1) - | 3.5387e-1 (9.99e-2) |
| UF7 | 4.8710e-1 (1.43e-1) - | 1.5321e-1 (5.74e-2) = | 2.0617e-1 (1.15e-1) = | 3.5833e-1 (1.54e-2) - | 1.7953e-1 (1.30e-1) |
| UF8 | 4.7423e-1 (2.04e-1) - | 2.6922e-1 (4.89e-3) + | 3.2978e-1 (3.08e-2) - | 4.5787e-1 (3.55e-2) - | 2.9341e-1 (2.42e-2) |
| UF9 | 5.3522e-1 (8.75e-2) - | 6.0158e-1 (7.97e-2) - | 4.1953e-1 (5.47e-2) = | 4.7384e-1 (3.67e-2) - | 4.2090e-1 (6.69e-2) |
| BT1 | 3.8155e+0 (1.30e-1) - | 3.5959e+0 (1.90e-1) - | 3.8677e+0 (5.83e-2) - | 3.9117e+0 (1.11e-1) - | 3.0033e+0 (1.55e-1) |
| BT2 | 1.2234e+0 (1.05e-1) - | 1.1831e+0 (1.54e-1) - | 1.7806e+0 (1.16e-1) - | 1.6437e+0 (1.19e-1) - | 6.9230e-1 (3.92e-2) |
| BT3 | 3.3778e+0 (2.33e-1) - | 3.0974e+0 (1.96e-1) - | 3.9199e+0 (1.18e-1) - | 3.7213e+0 (1.12e-1) - | 2.2765e+0 (1.57e-1) |
| BT4 | 3.4445e+0 (1.63e-1) - | 2.9880e+0 (3.10e-1) - | 3.7974e+0 (1.13e-1) - | 3.5029e+0 (1.07e-1) - | 2.2187e+0 (1.71e-1) |
| BT5 | 3.8404e+0 (1.25e-1) - | 3.5216e+0 (1.58e-1) - | 3.9131e+0 (8.48e-2) - | 3.8525e+0 (9.47e-2) - | 3.0848e+0 (1.73e-1) |
| BT6 | 9.1604e-1 (2.77e-1) - | 8.4030e-1 (5.65e-16)- | 1.7267e+0 (3.05e-1) - | 3.7600e-1 (1.18e-2) = | 4.3709e-1 (2.06e-1) |
| BT7 | 5.1784e-1 (1.21e-1) - | 7.8220e-1 (1.81e-1) - | 9.5664e-1 (1.90e-1) - | 1.6763e-1 (4.30e-2) + | 4.2670e-1 (2.72e-1) |
| BT8 | 4.8502e+0 (9.65e-1) - | 8.4030e-1 (5.65e-16)+ | 4.3488e+0 (4.48e-1) - | 3.9141e-1 (2.79e-2) + | 2.6872e+0 (4.11e-1) |
| BT9 | 2.6762e+0 (2.92e-1) - | 2.7672e+0 (1.14e-1) - | 3.4269e+0 (1.40e-1) - | 2.9656e+0 (7.87e-2) - | 2.5105e+0 (9.46e-2) |
| LSMOP1 | 3.1136e-1 (9.02e-2) = | 5.7431e-1 (1.83e-1) - | 1.4748e-1 (6.03e-2) + | 3.3215e-1 (4.04e-2) = | 3.1627e-1 (5.39e-2) |
| LSMOP2 | 1.8787e-1 (2.68e-2) - | 1.9961e-1 (1.54e-2) - | 1.9379e-1 (2.73e-2) - | 1.1689e-1 (7.29e-3) + | 1.7049e-1 (1.11e-2) |
| LSMOP3 | 8.1304e-1 (2.65e-1) + | 1.2385e+0 (1.97e-1) + | 7.1523e-1 (1.13e-1) + | 1.1367e+0 (1.70e-1) + | 3.4084e+0 (2.01e+0) |
| LSMOP4 | 2.5493e-1 (5.52e-2) - | 2.2218e-1 (6.12e-2) = | 7.3970e-2 (2.09e-2) + | 2.2175e-1 (3.96e-2) = | 2.1624e-1 (3.69e-2) |
| LSMOP5 | 5.9936e-1 (1.74e-1) = | 6.3529e-1 (1.64e-1) = | 2.4316e-1 (1.31e-1) + | 7.2199e-1 (6.32e-2) - | 5.9972e-1 (1.36e-1) |
| LSMOP6 | 2.3545e-1 (3.86e-2) + | 3.8226e-1 (3.26e-2) - | 6.3629e-2 (2.82e-2) + | 3.2875e-1 (3.76e-2) + | 3.5123e-1 (3.57e-2) |
| LSMOP7 | 1.0588e+1 (1.56e+1) + | 1.2863e+0 (9.13e-3) + | 1.8919e+0 (1.93e+0) + | 1.2353e+0 (1.18e-2) + | 1.4844e+1 (1.41e+1) |
| LSMOP8 | 4.5822e-1 (2.23e-1) + | 6.5750e-1 (1.56e-1) - | 1.2183e-1 (5.38e-2) + | 7.2358e-1 (6.02e-2) - | 5.7027e-1 (1.48e-1) |
| LSMOP9 | 1.4042e+0 (9.58e-1) + | 8.1004e-1 (3.39e-16)+ | 6.3753e-1 (1.55e-1) + | 8.1071e-1 (6.82e-4) + | 2.3517e+0 (9.10e-1) |
| +/-/= | 11/25/3 | 11/25/3 | 10/26/3 | 13/19/7 | |

**Table 6** CPU time for scaling the number of decision variables (2 objectives)

| Pro. | $n$ | SparseEA | LSMOF | DSGA |
|---|---|---|---|---|
| LSMOP1 | 10 | 1.6756e+0 (1.13e-1) - | 1.4891e+0 (6.49e-2) - | 1.3700e+0 (2.57e-1) |
| | 30 | 1.3630e+0 (2.09e-1) - | 6.5839e-1 (4.37e-2) - | 5.6764e-1 (2.77e-2) |
| | 50 | 3.1239e+0 (4.29e-1) - | 1.3927e+0 (8.89e-2) - | 1.1716e+0 (2.72e-2) |
| | 100 | 7.1544e+0 (6.94e-1) - | 3.0502e+0 (1.20e-1) - | 2.6010e+0 (4.95e-2) |
| | 300 | 2.0307e+1 (7.52e-1) - | 8.9744e+0 (3.17e-1) - | 8.4156e+0 (2.77e-1) |
| | 500 | 3.2891e+1 (1.40e+0) - | 1.3682e+1 (3.54e-1) = | 1.3755e+1 (1.06e-1) |
| | 1000 | 1.1773e+2 (4.11e+0) - | 5.1296e+1 (3.72e+0) = | 5.2163e+1 (2.80e+0) |
| LSMOP5 | 10 | 1.7186e+0 (5.08e-2) - | 1.3452e+0 (7.12e-2) = | 1.3478e+0 (3.57e-2) |
| | 30 | 1.5262e+0 (1.48e-1) - | 6.9653e-1 (5.28e-2) - | 6.3113e-1 (2.35e-2) |
| | 50 | 3.1625e+0 (2.69e-1) - | 1.4638e+0 (8.09e-2) - | 1.2823e+0 (3.32e-2) |
| | 100 | 7.0126e+0 (2.71e-1) - | 3.4452e+0 (1.50e-1) - | 2.8021e+0 (4.37e-2) |
| | 300 | 2.0736e+1 (4.92e-1) - | 1.0508e+1 (6.20e-1) - | 8.4338e+0 (7.92e-2) |
| | 500 | 3.4321e+1 (6.17e-1) - | 1.7277e+1 (1.20e+0) - | 1.5228e+1 (4.31e-1) |
| | 1000 | 1.2604e+2 (5.53e+0) - | 5.9962e+1 (2.31e+0) - | 5.1496e+1 (5.08e-1) |
| LSMOP9 | 10 | 1.1714e+0 (6.09e-2) + | 1.2573e+0 (6.23e-2) = | 1.2950e+0 (9.69e-2) |
| | 30 | 1.5531e+0 (6.53e-2) - | 1.2077e+0 (6.33e-2) - | 6.6127e-1 (8.41e-2) |
| | 50 | 3.2256e+0 (1.28e-1) - | 2.4059e+0 (1.95e-1) - | 1.2758e+0 (4.76e-2) |
| | 100 | 6.9683e+0 (1.72e-1) - | 3.3972e+0 (1.04e+0) - | 2.8765e+0 (6.90e-2) |
| | 300 | 1.9441e+1 (5.01e-1) - | 5.2587e+0 (4.74e-1) + | 8.5197e+0 (8.48e-2) |
| | 500 | 3.2716e+1 (5.57e-1) - | 8.4073e+0 (1.95e-1) + | 1.4727e+1 (1.06e-1) |
| | 1000 | 1.1948e+2 (1.71e+0) - | 3.0912e+1 (7.62e-1) + | 5.1153e+1 (3.33e-1) |
| +/-/= | | 1/20/0 | 3/14/4 | |

MOPs. So some modification need to be made before NSGA can be used to efficiently solve large-scale MOPs.

Finally, Figure 7 depicts the trend of CPU time and IGD value for DSGA solving LSMOP9 with different numbers of decision variables. It is obvious that both CPU time and IGD value increase as the rising of dimensions. The CPU time spent on 2-objective problems and 3-objective problems are vary




(a) IGD of ZDT1

(b) IGD of DTLZ1

(c) IGD of UF1

(d) IGD of BT1

(e) IGD of LSMOP1

**Fig. 6** The decreasing trajectory of IGD.

close, while the IGD values are dramatically different, and the gap increase as the number of dimension rises. This means that, for scaling the number of objectives, DSGA is not sensitivity on CPU time but on IGD value.

**Table 7** IGD value for scaling the number of decision variables (2 objectives)

| Pro. | $n$ | SparseEA | LSMOF | DSGA |
|---|---|---|---|---|
| LSMOP1 | 10 | - | - | - |
| | 30 | 5.0793e-1 (1.93e-1) - | 3.2681e-1 (2.22e-2) = | 3.1610e-1 (6.66e-2) |
| | 50 | 6.2709e-1 (1.63e-1) - | 3.7718e-1 (3.62e-2) = | 3.7305e-1 (8.21e-2) |
| | 100 | 6.3945e-1 (1.51e-1) - | 4.9600e-1 (5.52e-2) = | 5.0090e-1 (1.08e-1) |
| | 300 | 6.7988e-1 (1.04e-1) + | 5.9195e-1 (3.99e-2) + | 1.0629e+0 (1.63e-1) |
| | 500 | 6.8761e-1 (7.95e-2) + | 6.0422e-1 (2.80e-2) + | 1.8285e+0 (2.87e-1) |
| | 1000 | 6.2940e-1 (1.57e-1) + | 6.2921e-1 (1.82e-2) + | 2.5851e+0 (2.15e-1) |
| LSMOP5 | 10 | - | - | - |
| | 30 | 2.3685e-1 (6.47e-2) = | 2.2017e-1 (4.31e-2) = | 2.0672e-1 (3.49e-2) |
| | 50 | 2.9452e-1 (2.62e-2) = | 2.2484e-1 (2.32e-2) + | 2.9841e-1 (2.76e-2) |
| | 100 | 2.3733e-1 (6.28e-3) - | 1.6264e-1 (7.73e-3) + | 2.3089e-1 (7.32e-3) |
| | 300 | 1.2592e-1 (3.70e-3) - | 7.4986e-2 (1.64e-3) + | 1.2223e-1 (2.66e-3) |
| | 500 | 9.5754e-2 (1.23e-3) - | 5.1547e-2 (1.01e-3) + | 9.3354e-2 (2.05e-3) |
| | 1000 | 5.6494e-2 (7.98e-4) + | 3.5781e-2 (1.58e-3) + | 5.9110e-2 (6.76e-4) |
| LSMOP9 | 10 | - | - | - |
| | 30 | 8.1004e-1 (3.39e-16) + | 8.1069e-1 (5.62e-4) + | 2.3030e+0 (7.65e-1) |
| | 50 | 8.1004e-1 (3.39e-16) + | 8.1004e-1 (3.39e-16) + | 1.8156e+0 (5.52e-1) |
| | 100 | 8.1004e-1 (3.39e-16) + | 8.1004e-1 (3.39e-16) + | 1.9278e+0 (3.20e-1) |
| | 300 | 8.1004e-1 (3.39e-16) + | 8.1004e-1 (3.39e-16) + | 1.4746e+0 (1.90e-2) |
| | 500 | 8.1004e-1 (3.39e-16) + | 8.0953e-1 (6.62e-4) + | 1.2331e+0 (8.49e-3) |
| | 1000 | 8.1004e-1 (3.39e-16) + | 8.0736e-1 (1.28e-3) + | 1.0204e+0 (3.75e-3) |
| +/-/= | | 10/6/2 | 14/0/4 | |

**Table 8** CPU time for scaling the number of decision variables (3 objectives)

| Pro. | $n$ | SparseEA | LSMOF | DSGA |
|---|---|---|---|---|
| LSMOP2 | 10 | 1.1971e+0 (4.30e-2) + | 3.5560e+1 (1.03e+0) - | 1.3906e+0 (3.96e-2) |
| | 30 | 1.3138e+0 (5.05e-2) - | 2.0486e+0 (1.94e-1) - | 6.4753e-1 (2.37e-2) |
| | 50 | 2.9326e+0 (7.24e-2) - | 5.9248e+0 (2.53e-1) - | 1.3618e+0 (3.70e-2) |
| | 100 | 6.8740e+0 (1.03e-1) - | 1.5550e+1 (3.64e-1) - | 2.9988e+0 (5.07e-2) |
| | 300 | 1.9992e+1 (2.69e-1) - | 4.0133e+1 (6.56e-1) - | 9.3960e+0 (2.88e-1) |
| | 500 | 3.3210e+1 (3.77e-1) - | 5.9893e+1 (1.20e+0) - | 1.5660e+1 (1.63e-1) |
| | 1000 | 1.1838e+2 (1.01e+0) - | 1.7627e+2 (2.10e+0) - | 5.3182e+1 (2.62e-1) |
| LSMOP6 | 10 | 1.3651e+0 (5.26e-2) + | 3.6867e+1 (8.88e-1) - | 1.5633e+0 (3.87e-2) |
| | 30 | 1.3643e+0 (5.54e-2) - | 2.1508e+0 (8.22e-2) - | 7.1495e-1 (1.90e-2) |
| | 50 | 2.7061e+0 (1.25e-1) - | 3.2657e+0 (1.70e-1) - | 1.5332e+0 (3.53e-2) |
| | 100 | 6.0081e+0 (1.76e-1) - | 6.8277e+0 (6.44e-1) - | 3.3545e+0 (4.84e-2) |
| | 300 | 1.8781e+1 (3.89e-1) - | 1.3478e+1 (7.01e-1) - | 9.6922e+0 (8.30e-2) |
| | 500 | 3.1980e+1 (7.24e-1) - | 2.0348e+1 (6.81e-1) - | 1.6426e+1 (8.75e-2) |
| | 1000 | 1.1935e+2 (1.01e+0) - | 6.2599e+1 (2.15e+0) - | 5.5697e+1 (2.52e-1) |
| LSMOP9 | 10 | 1.1299e+0 (4.82e-2) + | 7.6870e+0 (1.27e+0) - | 1.3464e+0 (4.91e-2) |
| | 30 | 1.2272e+0 (4.47e-2) - | 5.3878e+0 (3.70e-1) - | 6.7267e-1 (1.83e-2) |
| | 50 | 2.5041e+0 (5.21e-2) - | 1.4876e+1 (1.34e+0) - | 1.3816e+0 (3.96e-2) |
| | 100 | 5.5634e+0 (1.12e-1) - | 1.6984e+1 (1.03e+1) - | 3.0713e+0 (6.39e-2) |
| | 300 | 1.7522e+1 (1.53e-1) - | 8.0936e+0 (2.01e+0) + | 9.9385e+0 (8.66e-1) |
| | 500 | 2.9876e+1 (5.23e-1) - | 1.2140e+1 (7.50e-1) + | 1.6102e+1 (4.65e-1) |
| | 1000 | 1.1291e+2 (6.44e-1) - | 3.7536e+1 (1.10e+0) + | 5.4566e+1 (3.29e-1) |
| +/-/= | | 3/18/0 | 3/18/0 | |

## 6 Conclusion

In this work, we have proposed a novel non-dominated sorting method, and based on it, a novel multi-objective genetic algorithm. The computational complexity of the proposed non-dominated sorting method is $O(mN \log N)$, which is the same as the current best. The numerical comparison between the proposed non-dominated sorting method and some other existing ones still shows its efficiency and outperformance. The novel multi-objective genetic algorithm is obtained by embedding the proposed non-dominated sorting method into the framework of NSGAII. Numerical experiments show that the proposed multi-objective genetic algorithm is efficient and promising.

For large-scale multi-objective optimization problem, the proposed method has no obvious advantage comparing with some methods specially designed for large-scale multi-objective optimization problem. The future work of the research is to extend the proposed method to large-scale multi-objective optimization problems.

**Table 9** IGD value for scaling the number of decision variables (3 objectives)

| Pro. | $n$ | SparseEA | LSMOF | DSGA |
|---|---|---|---|---|
| LSMOP2 | 10 | - | - | - |
| | 30 | 2.6467e-1 (1.28e-2) - | 2.8086e-1 (1.62e-2) - | 2.4747e-1 (1.32e-2) |
| | 50 | 2.7427e-1 (1.30e-2) - | 2.7517e-1 (1.64e-2) - | 2.6095e-1 (1.32e-2) |
| | 100 | 2.1629e-1 (9.56e-3) - | 2.3041e-1 (3.75e-3) - | 2.0319e-1 (4.24e-3) |
| | 300 | 1.1165e-1 (4.09e-3) - | 1.1495e-1 (4.12e-3) - | 1.0554e-1 (3.51e-3) |
| | 500 | 8.6117e-2 (4.06e-3) - | 8.9909e-2 (5.31e-3) - | 8.3681e-2 (4.01e-3) |
| | 1000 | 6.8583e-2 (4.05e-3) = | 7.3931e-2 (3.98e-3) - | 6.7881e-2 (3.40e-3) |
| LSMOP6 | 10 | - | - | - |
| | 30 | 2.3194e-1 (1.35e-1) + | 2.9160e-1 (4.61e-2) = | 2.8342e-1 (2.56e-2) |
| | 50 | 1.0317e+0 (1.60e-1) + | 6.2895e-1 (2.62e-2) + | 1.3517e+0 (3.97e-1) |
| | 100 | 1.2679e+0 (1.37e-1) + | 6.4945e-1 (1.73e-2) + | 4.4071e+0 (5.91e+0) |
| | 300 | 1.5235e+0 (1.45e-1) + | 7.2615e-1 (2.51e-2) + | 9.2594e+1 (1.46e+2) |
| | 500 | 1.5157e+0 (1.39e-1) + | 7.2690e-1 (1.58e-2) + | 1.4445e+3 (7.52e+2) |
| | 1000 | 1.6073e+0 (1.25e-1) + | 7.4857e-1 (3.04e-2) + | 5.7988e+3 (1.18e+3) |
| LSMOP9 | 10 | - | - | - |
| | 30 | 1.5368e+0 (2.52e-3) + | 1.0931e+0 (1.70e-1) + | 4.6722e+0 (1.94e+0) |
| | 50 | 1.5379e+0 (8.92e-5) + | 1.4455e+0 (1.69e-1) + | 3.8189e+0 (1.45e+0) |
| | 100 | 1.5379e+0 (1.81e-5) + | 1.5379e+0 (6.78e-16)+ | 4.6685e+0 (5.89e-1) |
| | 300 | 1.5379e+0 (3.69e-6) + | 1.5379e+0 (6.78e-16)+ | 3.0360e+0 (2.67e-1) |
| | 500 | 1.5375e+0 (1.55e-3) + | 1.5117e+0 (9.97e-2) + | 3.4895e+0 (3.94e-1) |
| | 1000 | 1.5377e+0 (7.76e-4) + | 1.2736e+0 (1.85e-1) + | 5.4596e+0 (1.21e+0) |
| +/-/= | | 12/5/1 | 11/6/1 | |

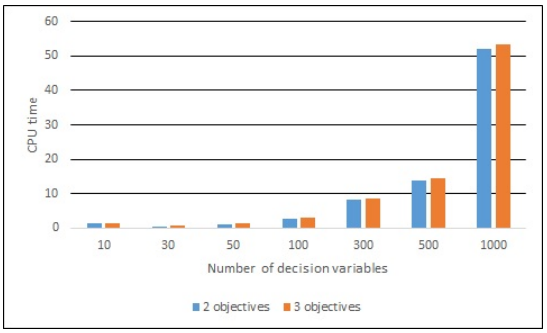
**Acknowledgements** This work is supported by the National Natural Science Foundation of China (Grants No. 11871128) and the open project of Key Laboratory, Mathematical College, Chongqing Normal University, Chongqing China (Grants No. CSSXKFKTZ201804). The authors of this paper would like to thank editors and reviews for their constructive comments and suggestions.

## Compliance with Ethical Standards

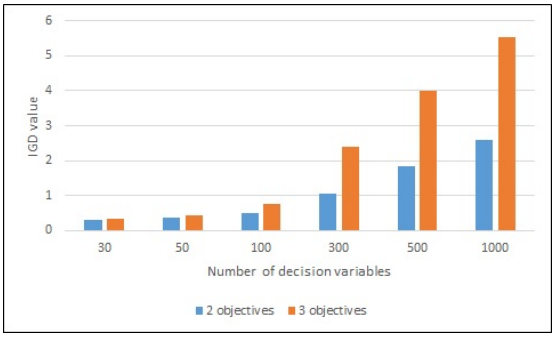
**Conflict of interest** The authors declare that they have no conflict of interest.
**Ethical approval** This article does not contain any studies with human participants or animals performed by any of the authors.
**Informed consent** Informed consent was obtained from all individual participants included in the study.
**Authorship contributions** All authors contributed to the study conception and design. Material preparation and data collection were performed by Qiang Long and Guoquan Li. The analysis and code were done by Qiang Long. The first draft of the manuscript was written by Lin Jiang, and then polished by Qiang Long and Guoquan li. All authors read and approved the final manuscript.

## References

1. Cheng, R., Jin, Y., Olhofer, M., et al.: Test problems for large-scale multiobjective and many-objective optimization. IEEE transactions on cybernetics **47**(12), 4108–4121 (2016)
2. Deb, K.: Multi-objective genetic algorithms: problem difficulties and construction of test problems. Evolutionary computation **7**(3), 205–230 (1999)
3. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation **6**(2), 182–197 (2002)
4. Deb, K., Sindhya, K., Okabe, T.: Self-adaptive simulated binary crossover for real-parameter optimization pp. 1187–1194 (2007)




(a) CPU time comparison for scaling decision variables




(b) IGD value comparison for scaling decision variables

**Fig. 7** CPU time and IGD value for scaling decision variables.


5. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable test problems for evolutionary multiobjective optimization. In: Evolutionary multiobjective optimization, pp. 105–145. Springer (2005)
6. Deep, K., Thakur, M.: A new mutation operator for real coded genetic algorithms. Applied Mathematics and Computation **193**(1), 211–230 (2007)
7. Du, Y., Xie, L., Liu, J., Wang, Y., Xu, Y., Wang, S.: Multi-objective optimization of reverse osmosis networks by lexicographic optimization and augmented epsilon constraint method. Desalination **333**(1), 66–81 (2014)
8. Fan, Z., Li, W., Cai, X., Li, H., Wei, C., Zhang, Q., Deb, K., Goodman, E.: Push and pull search for solving constrained multi-objective optimization problems. Swarm and evolutionary computation **44**, 665–679 (2019)
9. Goldberg, D.: Genetic algorithms in search, optimization, and machine learning. NN Schraudolph and J **3**(1) (1989)
10. Guo, Y., He, J., Xu, L., Liu, W.: A novel multi-objective particle swarm optimization for comprehensible credit scoring. Soft Computing **23**(18), 9009–9023 (2019)
11. Gutjahr, W.J., Pichler, A.: Stochastic multi-objective optimization: a survey on non-scalarizing methods. Annals of Operations Research **236**(2), 475–499 (2016)
12. Harik, G.R., Lobo, F.G., Goldberg, D.E.: The compact genetic algorithm. IEEE transactions on evolutionary computation **3**(4), 287–297 (1999)
13. Hei, Y., Zhang, C., Song, W., Kou, Y.: Energy and spectral efficiency tradeoff in massive mimo systems with multi-objective adaptive genetic algorithm. Soft Computing **23**(16), 7163–7179 (2019)
14. Jiang, S., Yang, S.: Evolutionary dynamic multiobjective optimization: Benchmarks and algorithm comparisons. IEEE transactions on cybernetics **47**(1), 198–211 (2016)

15. Li, H., Zhang, Q., Deng, J.: Biased multiobjective optimization and decomposition algorithm. IEEE transactions on cybernetics **47**(1), 52–66 (2016)
16. Long, Q.: A constraint handling technique for constrained multi-objective genetic algorithm. Swarm and Evolutionary Computation **15**, 66–79 (2014)
17. Long, Q., Wu, C., Huang, T., Wang, X.: A genetic algorithm for unconstrained multi-objective optimization. Swarm and Evolutionary Computation **22**, 1–14 (2015)
18. Long, Q., Wu, X., Wu, C.: Non-dominated sorting methods for multi-objective optimization: review and numerical comparison. Journal of Management Optimization **21**(1), 34–51 (2020)
19. Ma, X., Liu, F., Qi, Y., Wang, X., Li, L., Jiao, L., Yin, M., Gong, M.: A multiobjective evolutionary algorithm based on decision variable analyses for multiobjective optimization problems with large-scale variables. IEEE Trans. Evolutionary Computation **20**(2), 275–298 (2016)
20. McClymont, K., Keedwell, E.: Deductive sort and climbing sort: new methods for non-dominated sorting. Evolutionary computation **20**(1), 1–26 (2012)
21. Mirjalili, S.: Genetic algorithm. In: Evolutionary algorithms and neural networks, pp. 43–55. Springer (2019)
22. Mlakar, M., Petelin, D., Tušar, T., Filipič, B.: Gp-demo: differential evolution for multiobjective optimization based on gaussian process models. European Journal of Operational Research **243**(2), 347–361 (2015)
23. Passos, F., González-Echevarría, R., Roca, E., Castro-López, R., Fernández, F.: A two-step surrogate modeling strategy for single-objective and multi-objective optimization of radiofrequency circuits. Soft Computing **23**(13), 4911–4925 (2019)
24. Qiu, X., Xu, J.X., Tan, K.C., Abbass, H.A.: Adaptive cross-generation differential evolution operators for multiobjective optimization. IEEE Transactions on Evolutionary Computation **20**(2), 232–244 (2016)
25. Ruiz, A.B., Saborido, R., Luque, M.: A preference-based evolutionary algorithm for multiobjective optimization: the weighting achievement scalarizing function genetic algorithm. Journal of Global Optimization **62**(1), 101–129 (2015)
26. Srinivas, N., Deb, K.: Muiltiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary Computation **2**(3), 221–248 (1994)
27. Tian, Y., Cheng, R., Zhang, X., Jin, Y.: PlatEMO: A MATLAB platform for evolutionary multi-objective optimization. IEEE Computational Intelligence Magazine **12**(4), 73–87 (2017)
28. Tian, Y., Zhang, X., Wang, C., Jin, Y.: An evolutionary algorithm for large-scale sparse multi-objective optimization problems. IEEE Transactions on Evolutionary Computation (2019)
29. Tian, Y., Zheng, X., Zhang, X., Jin, Y.: Efficient large-scale multiobjective optimization based on a competitive swarm optimizer. IEEE Transactions on Cybernetics (2019)
30. Wang, H., Yao, X.: Corner sort for pareto-based many-objective optimization. IEEE transactions on cybernetics **44**(1), 92–102 (2013)
31. Wang, R., Zhou, Z., Ishibuchi, H., Liao, T., Zhang, T.: Localized weighted sum method for many-objective optimization. IEEE Transactions on Evolutionary Computation (2016)
32. While, L., Hingston, P., Barone, L., Huband, S.: A faster algorithm for calculating hypervolume. IEEE transactions on evolutionary computation **10**(1), 29–38 (2006)
33. Whitley, D.: A genetic algorithm tutorial. Statistics and computing **4**(2), 65–85 (1994)
34. Yang, M.D., Lin, M.D., Lin, Y.H., Tsai, K.T.: Multiobjective optimization design of green building envelope material using a non-dominated sorting genetic algorithm. Applied Thermal Engineering **111**, 1255–1264 (2017)
35. Ye Zhang Guowei Yang, D.G.Z.S., Chen, D.: A novel cacor-svr multi-objective optimization approach and its application in aerodynamic shape optimization of high-speed train. Soft Computing **23**(13), 5035–5051 (2019)
36. Zhang, Q., Li, H.: Moea/d: A multiobjective evolutionary algorithm based on decomposition. IEEE Transactions on evolutionary computation **11**(6), 712–731 (2007)
37. Zhang, Q., Zhou, A., Zhao, S., Suganthan, P.N., Liu, W., Tiwari, S.: Multiobjective optimization test instances for the CEC 2009 special session and competition. University of Essex, Colchester, UK and Nanyang technological University, Singapore, special

session on performance assessment of multi-objective optimization algorithms, technical report **264** (2008)
38. Zhang, X., Tian, Y., Cheng, R., Jin, Y.: An efficient approach to nondominated sorting for evolutionary multiobjective optimization. IEEE Transactions on Evolutionary Computation **19**(2), 201–213 (2014)
39. Zhang, X., Tian, Y., Jin, Y.: A knee point-driven evolutionary algorithm for many-objective optimization. IEEE Transactions on Evolutionary Computation **19**(6), 761–776 (2015)
40. Zhou, A., Qu, B.Y., Li, H., Zhao, S.Z., Suganthan, P.N., Zhang, Q.: Multiobjective evolutionary algorithms: a survey of the state of the art. Swarm and Evolutionary Computation **1**(1), 32–49 (2011)
41. Zhou, Y., Chen, Z., Zhang, J.: Ranking vectors by means of the dominance degree matrix. IEEE Transactions on Evolutionary Computation **21**(1), 34–51 (2017)
42. Zitzler, E., Deb, K., Thiele, L.: Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary computation **8**(2), 173–195 (2000)

# Figures

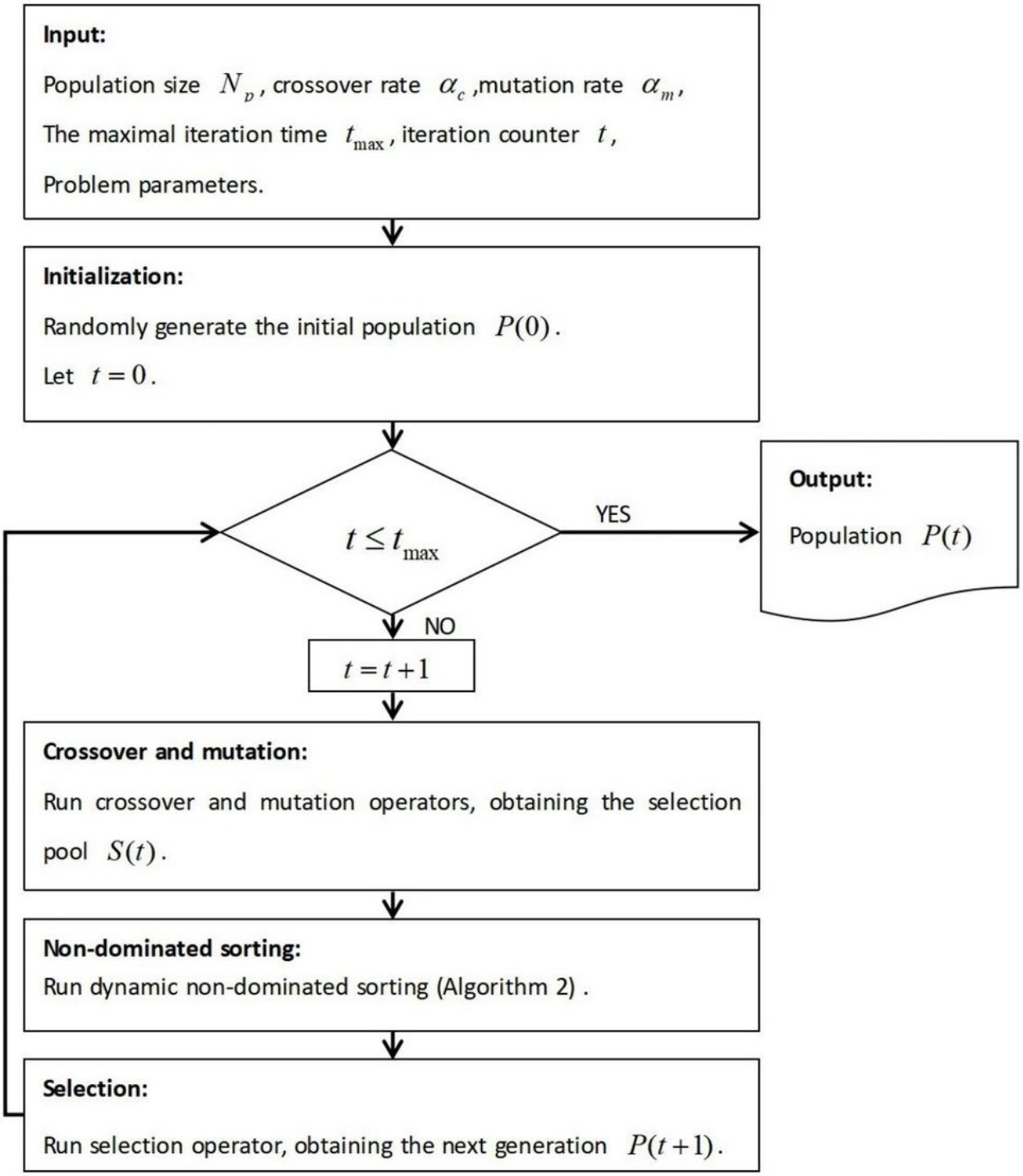



**Figure 1**

DSGA algorithm.

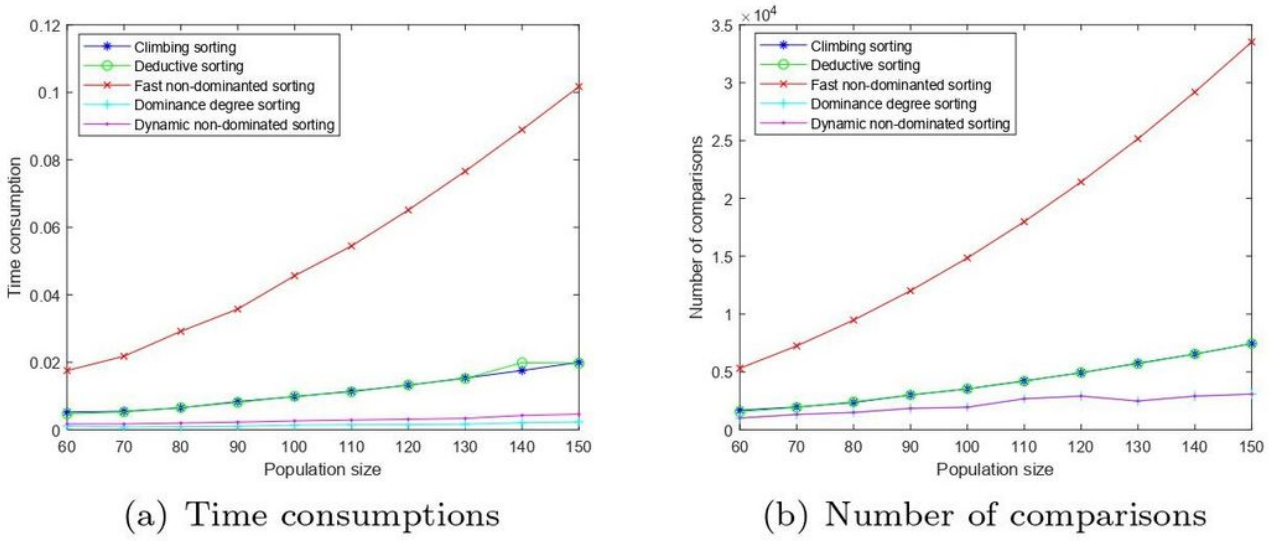



**Figure 2**

Numerical performance with respect to the variation of population size. In this test, the number of objectives m = 3, the number of Pareto frontiers k = 5, the population size is from 60 to 150 with increment 10, each Pareto frontier has the same number of solutions

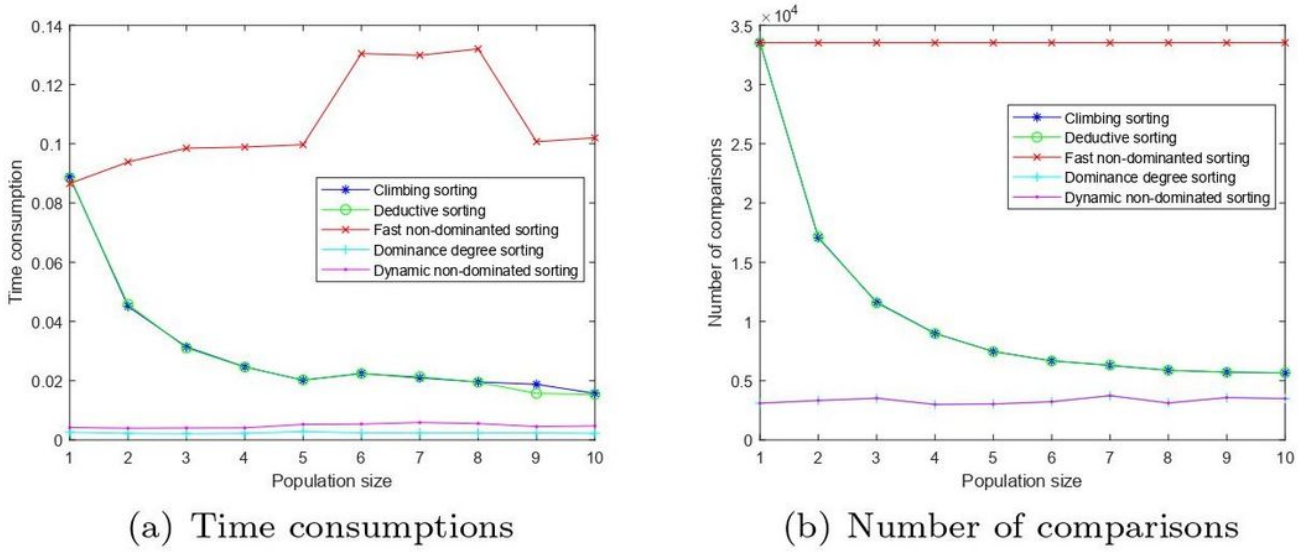



**Figure 3**

Numerical performance with respect to the variation of Pareto frontiers. In this test, the number of objectives m = 3, the number of Pareto frontiers increase from 1 to 10, the population size N = 150, each Pareto frontier has almost the same number of solutions.

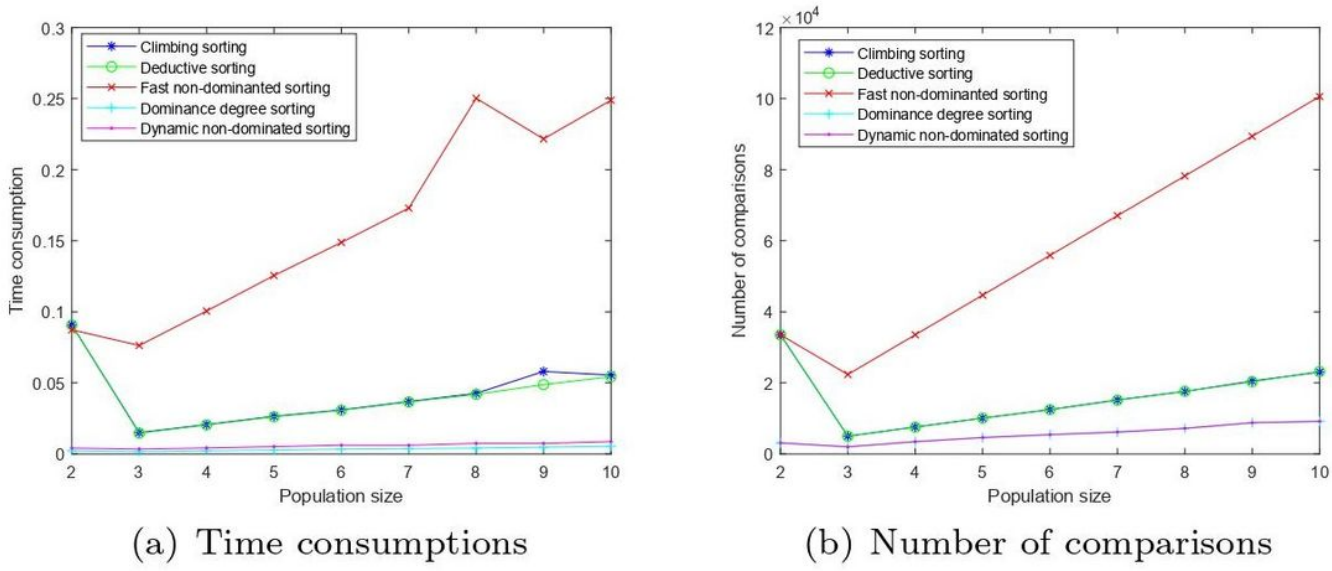



**Figure 4**

Numerical performance with respect to the variation of of objectives. In this test, the number of objectives increase from 2 to 10, the number of Pareto frontiers k = 5, the population size is N = 150, and each Pareto frontier has the same number of solutions.

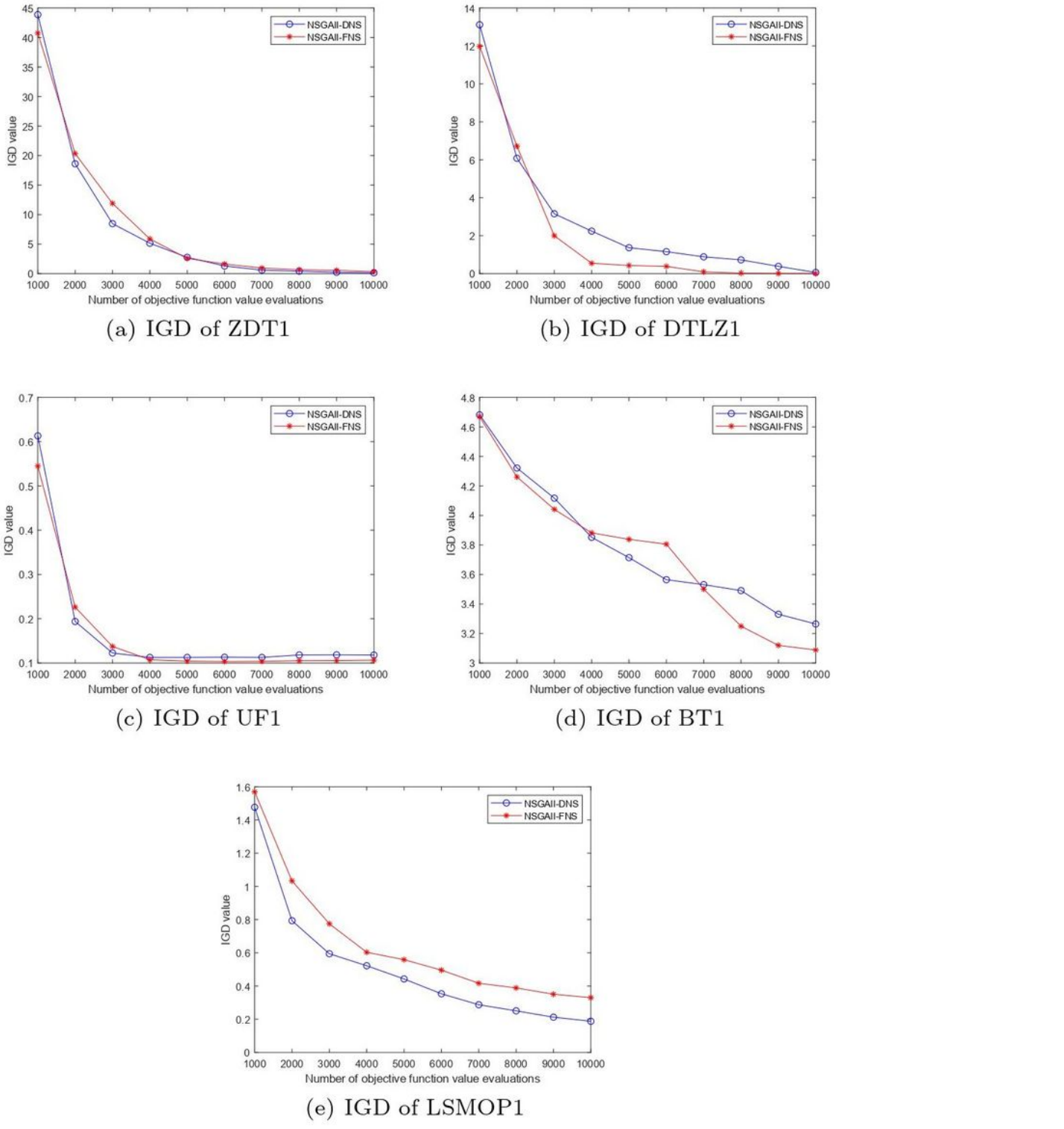



**Figure 5**

The decreasing trend of IGD.

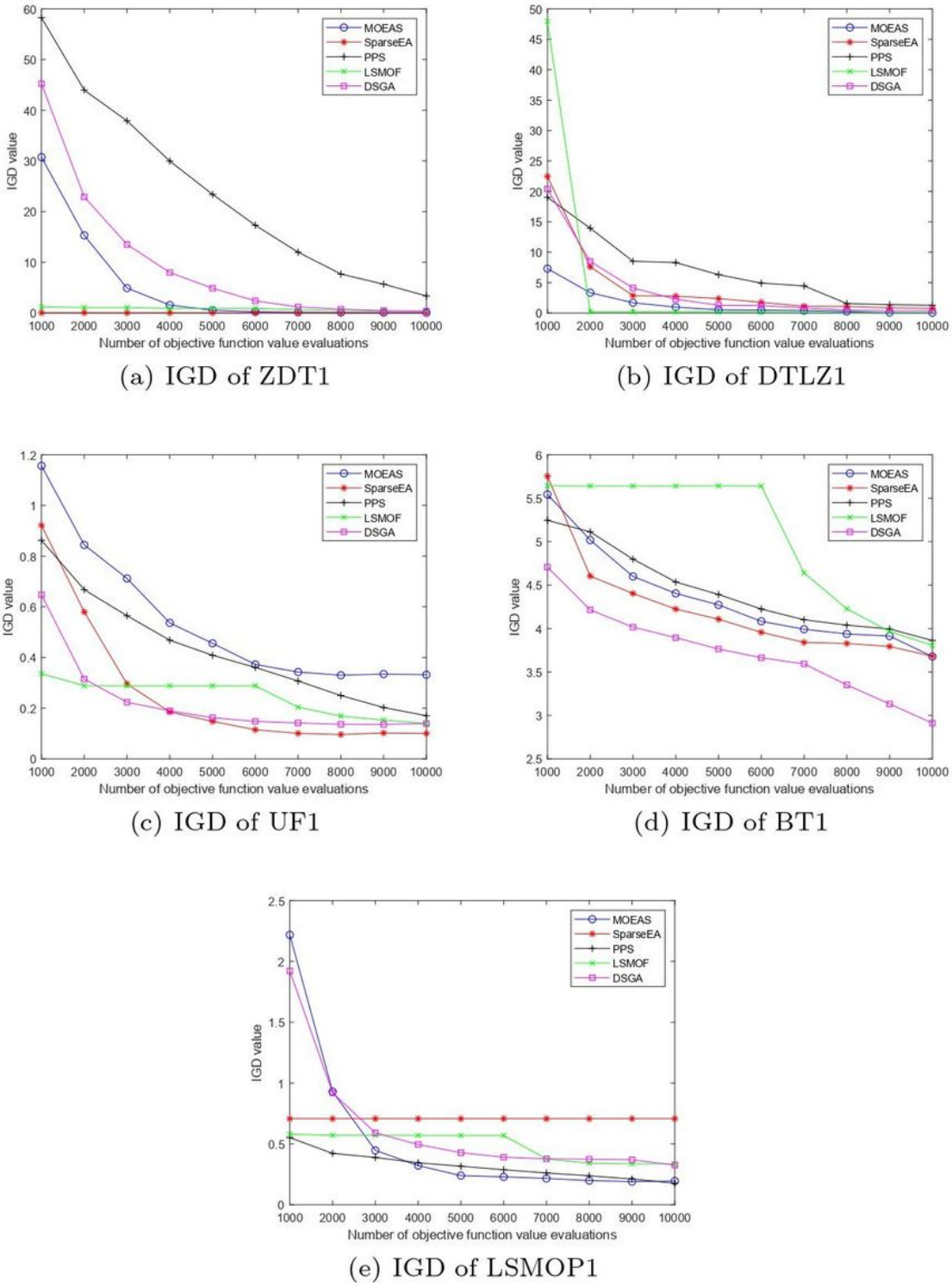
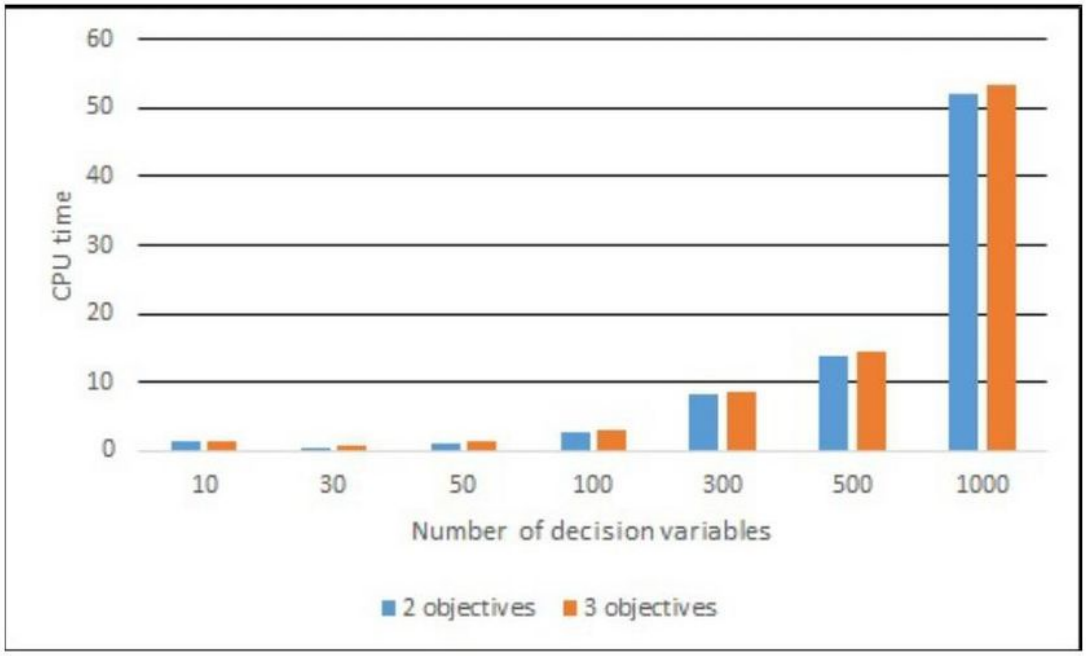
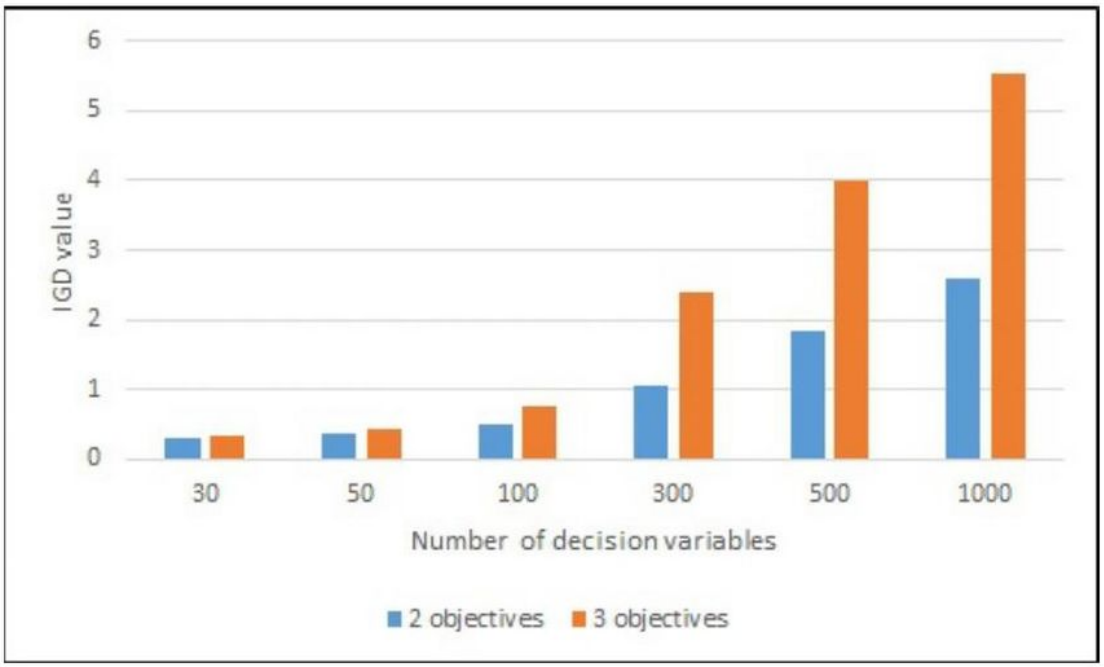



**Figure 6**

The decreasing trajectory of IGD.




(a) CPU time comparison for scaling decision variables




(b) IGD value comparison for scaling decision variables

**Figure 7**

CPU time and IGD value for scaling decision variables.