Information Sciences 226 (2013) 1–22

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Learning paradigm based on jumping genes: A general framework for enhancing exploration in evolutionary multiobjective optimization

Ke Li [a], Sam Kwong [a,*], Ran Wang [a], Kit-Sang Tang [b], Kim-Fung Man [b]

[a] Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong
[b] Department of Electronic Engineering, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong

ARTICLE INFO

Article history:
Received 19 July 2011
Received in revised form 30 October 2012
Accepted 7 November 2012
Available online 28 November 2012

Keywords:
Multiobjective optimization
Evolutionary algorithms
Jumping genes
Exploration and exploitation

ABSTRACT

Exploration and exploitation are two cornerstones of evolutionary multiobjective optimization. Most of the existing works pay more attention to the exploitation, which mainly focuses on the fitness assignment and environmental selection. However, the exploration, usually realized by traditional genetic search operators, such as crossover and mutation, has not been fully addressed yet. In this paper, we propose a general learning paradigm based on *Jumping Genes* (JG) to enhance the exploration ability of multiobjective evolutionary algorithms. This paradigm adapts the JG to the continuous search space, and its activation is completely adaptive during the evolutionary process. Moreover, in order to efficiently utilize the useful information, only non-dominated solutions eliminated by the environmental selection are chosen for the secondary exploitation. Empirical studies demonstrate that the performance of a baseline algorithm can be significantly improved by the proposed paradigm.

1. Introduction

A *Multiobjective Optimization Problem* (MOP) consists of several conflicting objectives. Generally speaking, an MOP can be stated in the following form:

$$\begin{aligned} &\text{minimize} \quad F(x) = (f_1(x), f_2(x), \cdots, f_m(x))^T \\ &\text{subject to} \quad x \in \Omega \end{aligned} \tag{1}$$

where $\Omega = \prod_{i=1}^{n}[a_i, b_i]$ is the *decision space*, $x = (x_1, \ldots, x_n)^T \in \Omega$ is a potential solution to (1). $F{:}\Omega \to R^m$ constitutes m individual objective functions.

Let $u, v \in \Omega$, u is said to dominate v if and only if $f_i(u) \leqslant f_i(v)$ for each $i \in \{1, \ldots, m\}$ and $f_j(u) \leqslant f_j(v)$ for at least one index $j \in \{1, \ldots, m\}$. A solution x^* is called the Pareto optimal solution of (1) if there is no solution $x \in \Omega$ that dominates x^*. $F(x^*)$ is then called a *Pareto optimal vector*. In other words, for a Pareto optimal vector, the improvement of any objective must result in the deterioration of at least one other objective. The set of all Pareto optimal solutions is called the *Pareto Set* (PS). Accordingly, the set of all Pareto optimal vectors, $PF = \{F(x) \in R^m | x \in PS\}$, is called the *Pareto Front* (PF) [20].

Evolutionary Algorithms (EAs) are stochastic optimization algorithms inspired by the natural evolution of species. In the last two decades, great effort has been dedicated to the development of *Multiobjective Evolutionary Algorithms* (MOEAs). These methods can generate a set of approximated Pareto optimal solutions in a single run. In addition, they can be used for solving problems without good mathematical properties. Since the pioneering work of Schaffer [25], a large variety of

* Corresponding author.
E-mail address: cssamk@cityu.edu.hk (S. Kwong).

0020-0255/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2012.11.002

MOEAs have been proposed and applied to a wide range of practical optimization problems [4,5]. Srinivas et al. proposed a *Non-dominated Sorting Genetic Algorithm* (NSGA) [28] that applies the non-dominated sorting technique to divide the population into a hierarchy of sub-groups based on the Pareto domination. Then a niching strategy is applied to promote the population diversity. Later, Deb et al. proposed the state-of-the-art NSGA-II [8], which uses the fast non-dominated sorting and crowding distance schemes. Zitzler et al. presented the *Strength Pareto EA* (SPEA) [42], which maintains an external archive to store non-dominated solutions, and uses a clustering procedure to keep the size of this archive. SPEA2 [41], which is an improved version of SPEA, incorporates a fine-grained fitness assignment scheme, a density estimation technique, and an enhanced archive truncation method. Recently, Zhang et al. proposed the *MOEA based on Decomposition* (MOEA/D) [36,18] which decomposes an MOP into a number of single-objective optimization sub-problems. Then a population-based method can be applied to solve these sub-problems simultaneously. Qu et al. proposed an MOEA based on *Summation of Normalized Objective Values and Diversified Selection* (SNOV-DS) [23]. Zhao et al. proposed a *Multiobjective Particle Swarm Optimization* (MOPSO) algorithm [38], which ensembles a set of parameter values and external archives, for solving MOPs. Combining the *Design For Multi-Objective Six Sigma* (DFMOSS) and quasi-Newton method with an aging model, Ono et al. proposed a memetic algorithm [22] for robust optimization.

Exploitation and exploration are two cornerstones of evolutionary approaches. Exploitation aims at discovering useful information by using the current knowledge. It associates with the modules for selecting and archiving elite solutions in an MOEA. Exploration, usually implemented by genetic search operators, such as crossover and mutation, tends to discover new and unknown areas in the search space. Although exploitation and exploration are equally important for problem solving, most of the existing works focus on the exploitation. It is encouraging to see that several non-traditional exploration operators have been proposed in recent years. Most of them take the characteristics of evolutionary dynamics into account and show competitive performances for solving MOPs. Inspired by the Baldwin effect [34], Gong et al. [11] proposed an improved clonal selection algorithm, in which antibodies are simultaneously evolved by four operators, i.e. clonal proliferation, Baldwinian learning, hyper-mutation, and clonal selection. Wang et al. [33] proposed a MOPSO, which employs a new optimal criterion based on the preferential order to identify the best compromise among solutions. Nebro et al. [21] proposed a hybrid meta-heuristic algorithm, i.e. *Archive-Based hYbrid Scatter Search* (AbYSS), which combines the scatter search and evolutionary operators. ke et al. [16] proposed a fast algorithm to calculate the hypervolume contribution of each solution and applied it for population truncation in MOEA. ke et al. [15] suggested an effective adaptive operator selection mechanism and parameter control method in mutliobjective differential evolution.

In this paper, we propose a learning paradigm based on jumping genes (denoted as JGBL), which aims at enhancing the exploration ability of an MOEA. It is worth mentioning that there are five key differences between JGBL paradigm and our previous works on *Jumping Genes Genetic Algorithm* (JGGA) [2,24].

1. In the original JGGA [2], solutions are encoded in binary strings, which are not suitable for problems in continuous search space. Although RJGGA [24] claims to adapt *Jumping Genes* (JG) to the continuous search space, the rationale of JG is merely simulated by a special treatment of the traditional crossover [6] and mutation [7], which in fact do not fully implement the JG features. To overcome these problems, the JG operators in the proposed JGBL paradigm are well developed in the real-coded system.
2. The application of JG is well motivated in JGBL paradigm, rather than being an additive genetic operator of the traditional crossover and mutation. Specifically, the rationale of JGBL paradigm is to provide some solutions, which are eliminated by the environmental selection, another chance for the secondary exploitation. Therefore, more diversified building blocks are expected to be generated, and then the prematurity can be greatly reduced.
3. In previous works [2,24], the JG operation is activated in a random manner, while our JGBL paradigm is adaptively activated during the evolutionary process.
4. In order to give sufficient exploitation on a specific solution, all JG operators are utilized simultaneously whenever the JGBL paradigm is activated. This is different from our previous works [2,24], in which only one JG operator is randomly chosen each time.
5. Previously, solutions are randomly selected from the entire population for the JG operation. Here, some deterministic criterion is used to select appropriate solutions for this secondary exploitation, which can utilize useful information more efficiently.

The remainder of this paper is organized as follows. Backgrounds and related works are given in Section 2. The underlying mechanism of JGBL paradigm is elaborated in Section 3. The performances of our proposed JGBL paradigm are empirically studied in Section 4. Finally, conclusion of this work is given in Section 5.

2. Background and related works

Crossover and mutation, inspired by the biological mechanisms of DNA, are two conventional genetic search operators in EAs [13]. JG, also known as transposon, can be used as an alternative for genetic recombination as well. The adaptation of JG, which is a context-sensitive [30] operator to promote the intra and inter movements of genes, to the context of EA has been developed for years. Simoes et al. [1] proposed a transposition mechanism, in which the transposon of a chromosome is flanked by an identical or inverted sequence. The activation of JG operations is to identify the sequence end. By employing

the artificial transposons, Spirov et al. [27] suggested a scheme that exploits the coevolution of host chromosomes and their genetic parasites. Different from the previous two works [1,27] where JG is only partially emulated, Chan et al. [2] comprehensively studied the JG phenomenon and proposed four computational JG operators in the framework of NSGA-II. Nawaz Ripon et al. [24] developed a *Real-coded Jumping Genes Genetic Algorithm* (RJGGA) to solve continuous MOPs. Instead of using the JG operators directly, RJGGA just employs the polynomial mutation [7] and *Simulated Binary Crossover* (SBX) [6] to emulate the single and double chromosomes' JG operators, respectively. Recently, Tang et al. [31] theoretically justified the outperformance of binary-coded JG. Besides the theoretical studies, JG has also been widely applied to various real-world optimization scenarios, such as passive circuits systems [39], planar UWB monopole antenna design [35], power voltage control systems [19] and wireless local area network optimization [3].

To our best knowledge, almost all existing studies on JG operations, in the context of EA, are implemented in the binary-coded system. Although RJGGA claims to adapt JG to continuous search space, the JG operators are merely emulated by crossover and mutation as discussed before. Handling continuous problems with binary-coded system has several drawbacks: (1) it is difficult to achieve arbitrary precision by using a binary string; (2) the Hamming cliff can largely impair the performance of binary-coded GAs; (3) it is inappropriate to treat all *Holland's schemata* equally in the continuous search space [12], etc. In contrast, individuals coded in real numbers can achieve as high precision as the bit length of the machine. Hamming cliff can be eliminated in the real-coded system. Moreover, the projection from genotype to phenotype becomes straightforward. Any change on the decision variables will result in the equipotent variation in the objective function, thus it provides a faster way for local tuning. However, since the boundaries of decision variables might be different from each other, the *cut and paste* and *copy and paste* operations in the original JG paradigm cannot be directly adapted to the continuous space. In details, the upper and lower bounds of the original position of a transposon might be different from those after the JG operations. This phenomenon is also known as boundary violation, which might make the mutants generated by the JG operations become invalid. Therefore, it is preferable to re-model the JG operators in the real-coded system, while at the same time, the boundary violation can be avoided.

3. Proposed algorithm

In this section, the individual representation and the remedy technique for boundary violation are described at first, and the underlying mechanism of JGBL paradigm is elaborated later.

3.1. Individual representation

In this paper, the individual is encoded as a chromosome consists of several genes. Instead of using "0–1" binary numbers in the binary-coded system, each gene holds a real number of the corresponding decision variable. Fig. 1 gives a simple example of individual $x = (x_1, \ldots, x_8)^T$ that is composed of eight consecutive genes.

3.2. Remedy technique for boundary violation

As discussed in Section 2, JG operators, such as *cut and paste*, might generate invalid individual due to the boundary violation problem. As shown in Fig. 1, x_2 becomes invalid if it is transferred to the gene position of x_5, since x_2 violates the boundary constraint of x_5.

Definition 1 (*Variable Information*). The variable information of x, denoted as $Inf(x)$, is defined as the ratio that occupied in its own range. Specifically, it is calculated as follows:

$$Inf(x) = \frac{x - lower(x)}{upper(x) - lower(x)} \tag{2}$$

where $upper(x)$ and $lower(x)$ represent the upper and lower bounds of x, respectively.

The calculation of $Inf(x)$ is similar to the popular normalization method in the EA literature [4]. Rather than a concrete value in the normalization case, $Inf(x)$ is used as a record of the percentage maintained by x in its corresponding range. The value after being transferred to a new position is restored as follows:

$$Tran(x) = lower(y) + Inf(x) \times [upper(y) - lower(y)] \tag{3}$$

where $Tran(x)$ indicates the value of x after being transferred to the gene position of y.

Boundary Constraint	[3,4]	[1,2]	[4,5]	[0,1]	[5,6]	[7,8]	[1,2]	[8,9]
Decision variable	3.456	1.245	4.654	0.012	5.654	7.568	1.025	8.332
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8

Fig. 1. An example of solution representation.

3.3. Computational form of JG

Generally speaking, there exists two types of JG operators [2], i.e. *cut and paste* and *copy and paste*. A transposon is made of genes selected from each chromosome in a random manner. There is no restriction on the number of transposons, while the size of each transposon is decided by a parameter called *jumping percentage*. The transposon contents are transferred in a horizontal manner, which is a type of lateral movement that happens in one chromosome or between different ones. The activation of JG operations is determined by a probabilistic factor called *jumping rate*, and the settle position of a transposon is randomly chosen. Fig. 2a and b gives the intuitive illustrations on the *cut and paste* over one chromosome and two different ones, respectively. Take Fig. 2a as an example, this chromosome consists of seven genes. Let *jumping percentage* = 0.4, thus the transposon is composed of $\lceil 7 \times 0.4 \rceil = 3$ genes. Suppose that genes *b*, *d* and *f* are chosen to form the transposon, and the insert position is after gene *a*. Then the transposon *bdf* is cut from the original chromosome and inserted to the position after *a*; meanwhile, the remaining genes *c*, *e* and *g* are merged together and shifted to the position after *bdf*. Different from the *cut and paste*, the transposon is inserted in a replacement manner for *copy and paste*. Fig. 3a and b gives the intuitive illustrations on the *copy and paste* act upon one chromosome and two different ones, respectively. Take Fig. 3a as an example, instead of being cut from its original position, the transposon *bdf* is just a copy of their original genes. Assuming that the position after gene *a* is chosen as the insert position, then three genes after *a* (i.e. *b*, *c*, *d*) are replaced by *bdf* while the other genes *e*, *f* and *g* remain intact. The process of double chromosomes' *cut and past* and *copy and paste* are analogous to that of the single chromosome.

3.4. JGBL in MOEAs

Different from single-objective optimization problems, where the quality of a solution is evaluated by a definite objective function, in MOPs, non-dominated solutions are usually incommensurable due to the conflicting nature of multiple criteria. Non-dominated solutions play a crucial role in providing the desired direction towards the PF, and the evolution might stagnate if their quality is not good enough. This circumstance often happens when most solutions have been absorbed in the area which is far away from the PF, or when they have been trapped by some local optima. Fig. 4 gives two examples to illustrate the relationship between the performance of NSGA-II (measured by the *Hypervolume* (HV) indicator [42]) and the number of non-dominated solutions (denoted as |*Non_dom*|). The black fork marker indicates the stage that the number

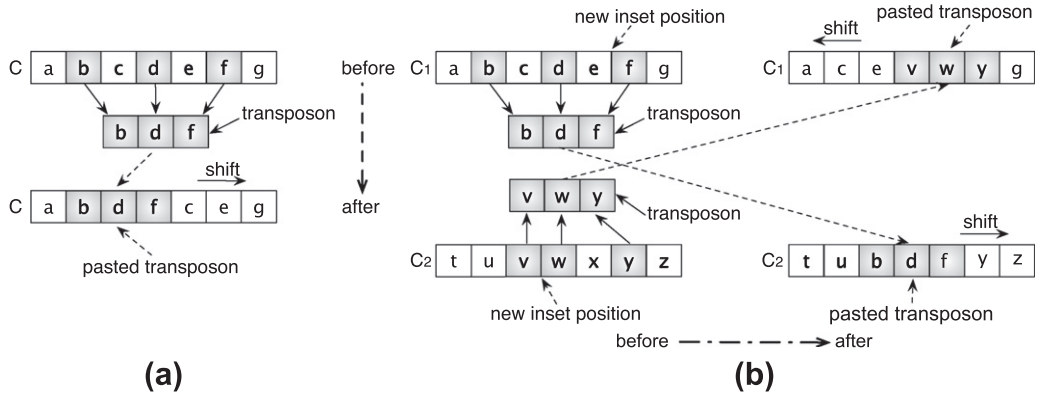


Fig. 2. *Cut and paste* transposition: (a) single and (b) double.

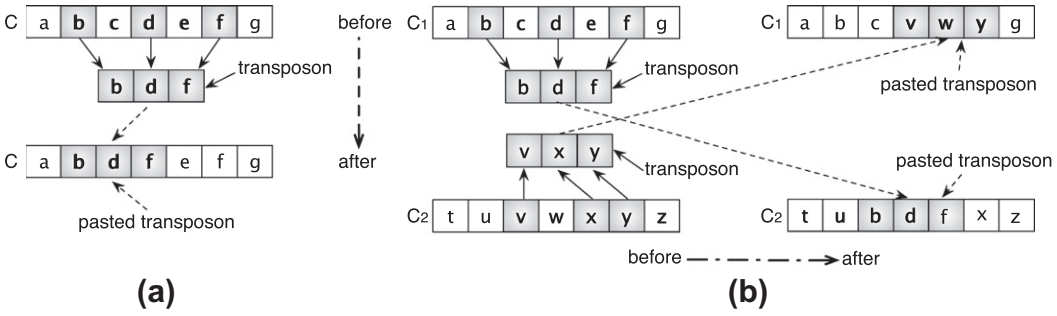


Fig. 3. *Copy and paste* transposition: (a) single and (b) double.

of non-dominated solutions in the population is smaller than or equal to a predefined threshold N (here N is the population size), while the green square marker represents the opposite situation. As shown in Fig. 4a, it is obvious that the trajectory's slope is steep at the first 20 generations, and it gradually turns to be moderate with the expansion of non-dominated solutions. This scenario can be ascribed to the polynomial bias of WFG 1, which absorbs most of the solutions in the regions far away from the PF. Moreover, WFG 1 also has flat bias property, which provides rare gradient information to the search algorithm, thus it is hard for the conventional genetic search operators to generate good solutions. DTLZ 3 [9] is a difficult problem which has many local optima that obstruct an MOEA from converging to the PF. As shown in Fig. 4b, the HV values keep being quite small during the first 130 generations. This can be ascribed to the fact that most of the solutions are mistakenly attracted by the local optima.

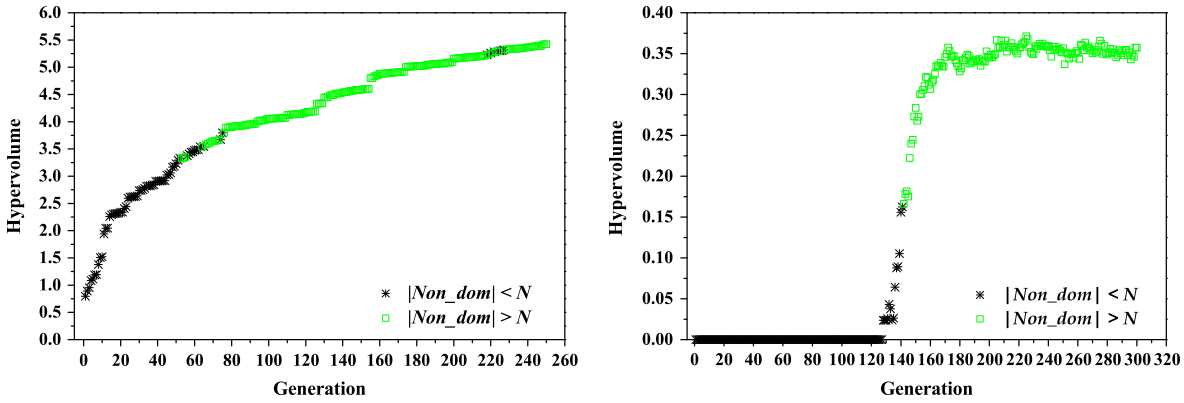
Almost all existing EAs tend to assign high survival rates to the elite solutions, but this might be insufficient for an effective search in some scenarios. For example, the fast explosion of the current best schemata usually results in a fast convergence of the algorithm, but it is also with a high risk of losing diversity and premature convergence when solving multimodal problems. Let us consider the examples discussed in Fig. 4 again, the stagnation of the evolution can be stimulated if more diversified and high quality building blocks can be generated. Rather than transmitting genes in a vertical manner, JG can be regarded as a complementation of traditional genetic search operators, which transmit genes in a horizontal way. Then, diversified building blocks can be obtained by the collaboration of the traditional genetic search operators and JG, thus the exploration ability of the baseline MOEAs is expected to be enhanced. In this paper, JG operations only take place when the number of non-dominated solutions is larger than the population size. Following the general structure of an MOEA, the basic framework which incorporates the JGBL paradigm is given in Algorithm 1, where P, Q, R, F and J denote the evolutionary population, the offspring population, the hybrid population merged by P and Q, the set of current non-dominated solutions and the mutant population generated by JG operations, respectively. The population size is denoted as *Popsize*. It is worth noting that if there are no solutions generated by JG, the *Popsize* offspring are purely generated by traditional genetic search operators (line 4 in Algorithm 1); otherwise, the offspring population would be a hybrid of solutions that are generated by JG and traditional genetic search operators together. The activation of JGBL paradigm is determined by the number of non-dominated solutions in R, while the frequency of JG operations is controlled by *jumping rate*.

Algorithm 1. Framework of the JGBL based MOEA

```
1  Initialization(P);
2  Initialization(F, J, Q, R);
3  repeat
4      Q ← Generate(P, PopSize);
5      R ← P ⋃ Q;
6      P ← EnvironmentalSelection(R);
7      F ← FilterNondominatedSolutions(R);
8      if |F| > PopSize then
9          E ← F - P;
10         J ← JG(P, E);
11         R' ← P ⋃ J;
12         P ← EnvironmentalSelection(R');
13     else
14         J ← ∅;
15     end
16 until termination condition fulfilled;
17 return P
```

Fig. 5 gives more intuitive illustration on line 5–15 of Algorithm 1. At first, a combined population $R_t = P_t \bigcup Q_t$ is formed, where P_t and Q_t are parent and offspring populations respectively at the current generation t. Then, non-dominated solutions in R_t are filtered out exclusively after the environmental selection (line 7 in Algorithm 1). The JGBL paradigm is activated when the number of non-dominated solutions is larger than *Popsize* (i.e. $|F_t| > Popsize$). E_t (line 9 in Algorithm 1) represents the non-dominated solutions that are eliminated by the environmental selection. Instead of eliminating these solutions directly, JGBL paradigm aims at exploiting useful information from them (line 10 in Algorithm 1). This is motivated by the following two advantages.

1. Since solutions in E_t possess satisfactory convergence property, conducting JG operations on them might have larger chance to generate better mutants, compared to those dominated ones (this issue will be discussed in Section 4.7).
2. In general, these solutions are eliminated by the environmental selection since they reside in more crowded areas. In this case, the JGBL paradigm can be regarded as a local refiner which might be helpful for exploring some unknown regions. Thus the overall spread of solutions might be enhanced at last, by the preferable mutants.



(a) Evolutionary trajectory of *HV* on WFG 1 (b) Evolutionary trajectory of *HV* on DTLZ 3

Fig. 4. Relationship between the performance of NSGA-II and |*Non_dom*|.

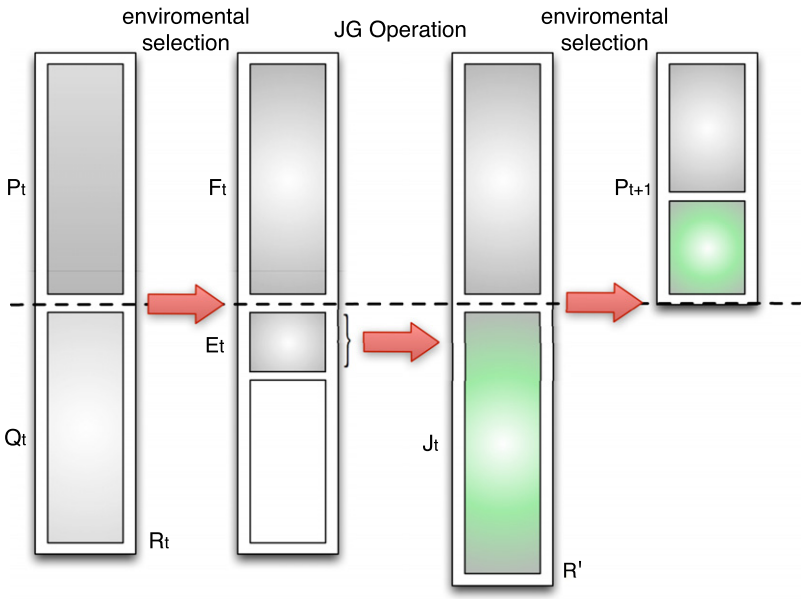


Fig. 5. The work flow of JGBL paradigm embedded in the environmental selection procedure of an MOEA.

After JG operations, the JG mutants in J_t are combined with the non-dominated solutions survived from the first round environmental selection to form a new hybrid population R' (line 11 in Algorithm 1). Afterwards, another round environmental selection is operated upon R' to filter out the parent population P_{t+1} for the next generation (line 12 in Algorithm 1).

Fig. 6 gives a possible population distribution (*Popsize* = 8 here), where star markers indicate the non-dominated solutions eliminated by the first round environmental selection. JGBL paradigm aims at exploiting these solutions, and some satisfactory mutants (indicated by square markers) might be generated accordingly. Afterwards, some previously survived non-dominated solutions might be eliminated by the second round environmental selection, since they reside in a more crowded area compared to the JG mutants. The latter two key differences of JGBL paradigm introduced in Section 1 are further explained as follows.

1. All those four different JG operators introduced in Section 3.3 are utilized simultaneously whenever the JGBL paradigm is activated. In this case, a solution is able to generate six mutants each time. Therefore, more opportunities can be provided to explore unknown regions and the population diversity has larger chance to be enhanced.
2. When the JG operator is applied onto two different solutions, one solution is selected from those survived from the first round environmental selection, while the other is chosen from the non-dominated solutions eliminated by the first round environmental selection. In this way, useful information is likely to be fully utilized and exploited.

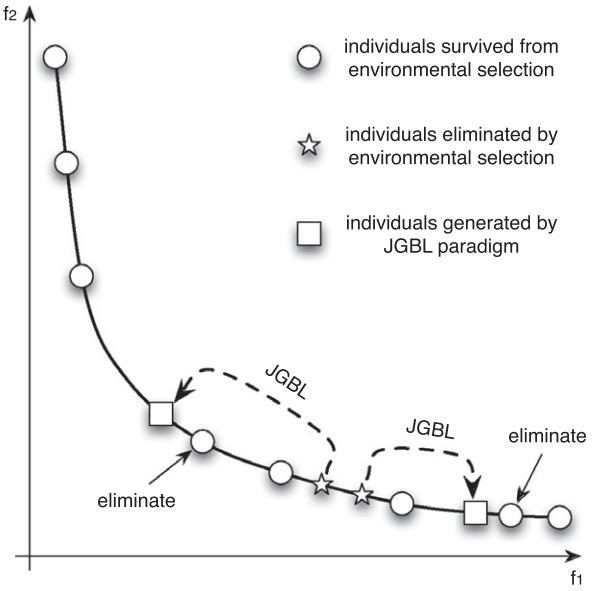


Fig. 6. The possible effect of JGBL paradigm.

4. Empirical studies

In this section, the JGBL paradigm is incorporated with two state-of-the-art MOEAs, i.e. NSGA-II and SPEA2, respectively, for validating its advantages. Generally speaking, the empirical studies include the following five aspects:

1. Performance comparisons with two baseline algorithms.
2. Sensitive studies on the two hyper-parameters, i.e. *jumping percentage* and *jumping rate*.
3. CPU-time cost study.
4. Verification on the rationale of JGBL paradigm.
5. Performance comparisons with the state-of-the-art MOEA/D.

In the following paragraphs, the benchmark problems, performance metrics and parameter settings are given at first, then the empirical studies are elaborated step by step.

4.1. Test instances

All test instances used in this paper have already been widely investigated in this literature. The bi-objective instances are taken from WFG and ZDT test suites [14,40] and the tri-objective instances consist of DTLZ test suite [9]. All these test instances are scalable to any number of problem dimensionality, while DTLZ and WFG are also scalable to arbitrary number of objectives. In addition, WFG has some distinct difficulties such as bias, deceptive and parameter dependency. The problem dimensionality (denoted as *nreal*) of each test instance is given in Table 1. Details about these test instances can be found in the corresponding Refs. [9,14,40].

4.2. Performance metrics

As discussed in [43], no unary metric can give a comprehensive measure on the performance of an MOEA. Here, we consider three widely used unary metrics and one binary indicator:

Table 1
The settings on problem dimensionality for different test instances.

Test instances	nreal
ZDT 1 to ZDT 3	30
ZDT 4 and ZDT 6	10
DTLZ 1	7
DTLZ 2 to DTLZ 6	12
DTLZ 7	22
WFG 1 to WFG 9	6

Table 2
Reference points settings for different test instances.

Test instances	Reference points
ZDT test suite	(2.0,2.0)
DTLZ 1	(1.0,1.0,1.0)
DLTZ 2 to DTLZ 6	(2.0,2.0,2.0)
DTLZ 7	(2.0,2.0,7.0)
WFG test suite	(3.0,5.0)

1. *Generation Distance* (GD) [32]: Let P^* be a set of uniformly distributed Pareto optimal vectors and P be the obtained PF approximation. The GD value of P is calculated as:

$$GD(P,P^*) = \frac{\sum_{x\in P^*} dist(x,P)}{|P|} \tag{4}$$

where $dist(x,P)$ is the minimum Euclidean distance between x and vectors in P, and $|P|$ indicates the size of P. In general, GD is able to evaluate the convergence of P if $|P^*|$ is large enough. Here, $|P^*| = 1000$ for bi-objective instances and $|P^*| = 10{,}000$ for tri-objective ones. It is clear that a lower GD value indicates a better convergence.
2. *Spacing* (SP) [26]: SP measures the uniformity of the PF approximation. Mathematically, it is formulated as:

$$\mathrm{SP} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\bar{d}-d_i)^2} \tag{5}$$

where n is the size of the PF approximation. d_i is the Euclidean distance between vector i and its nearest neighbor, and $\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i$. SP = 0 indicates that all approximated vectors are uniformly distributed along the PF.
3. *Hypervolume indicator* (HV) [42]: Let $F^* = (f_1(x^*),\ldots,f_m(x^*))^T$ be a reference point in the objective space which is dominated by all approximated vectors. The HV value of P is the volume of union regions that are dominated by P and bounded by F^*.

$$I_H(P) = volume\left(\bigcup_{x\in P}[f_1(x),f_1(x^*)]\times\cdots\times[f_m(x),f_m(x^*)]\right) \tag{6}$$

The higher HV, the better P approximates the PF. The reference points used for evaluating the HV are given in Table 2.
4. Binary ϵ-indicator (I_ϵ) [43]: A vector $F(x) = (f_1(x),\ldots,f_m(x))^T$ is said to ϵ-dominate another vector $F(y) = (f_1(y),\ldots,f_m(y))^T$, written as $F(x) \succeq_\epsilon F(y)$, if and only if $\forall 1 \leqslant i \leqslant m: f_i(x) \leqslant \epsilon \cdot f_i(y)$ for a given $\epsilon > 0$. Formally, the I_ϵ of two approximation sets A and B is defined as:

$$I_\epsilon(A,B) = \inf_{\epsilon\in R^+}\{\forall x\in B, \exists y\in A : F(x) \succeq_\epsilon F(y)\} \tag{7}$$

It can be calculated as:

$$I_\epsilon(A,B) = \max_{y\in B}\min_{x\in A}\max_{1\leqslant i\leqslant m}\frac{f_i(x)}{f_i(y)} \tag{8}$$

There are three cases as follows:
(a) $\succcurlyeq$: $I_\epsilon(A,B) \leqslant 1$ and $I_\epsilon(A,B) > 1 \Rightarrow A$ is better than B.
(b) $\preccurlyeq$: $I_\epsilon(A,B) > 1$ and $I_\epsilon(A,B) \leqslant 1 \Rightarrow B$ is better than A.
(c) $\approx$: $I_\epsilon(A,B) > 1$ and $I_\epsilon(A,B) > 1 \Rightarrow A$ and B are incomparable.

In comparison, I_ϵ requires two MOEAs each time. For each MOEA, 50 simulation runs will produce 50 approximation sets. Hence, there are 2500 pairs for comparison.

4.3. General experimental setting

All algorithms are implemented in ANSI C. Empirical studies are conducted on Intel Core 2 Duo CPU P8400@2.26 GHz, 2 GB RAM computer, with Microsoft Windows 7 operating system. The parameter settings are as follows.

- Control parameters of SBX and polynomial mutation: Following Refs. [8,41], crossover rate $p_c = 0.9$, crossover index $\eta_c = 20$, mutation rate $p_m = \frac{1}{nreal}$, and mutation index $\eta_m = 20$.
- Hyper-parameters of JGBL paradigm: *jumping rate* = 0.6 and *jumping percentage* = 0.5.
- Population size (denoted *Popsize*): *Popsize* = 100 for all test instances.

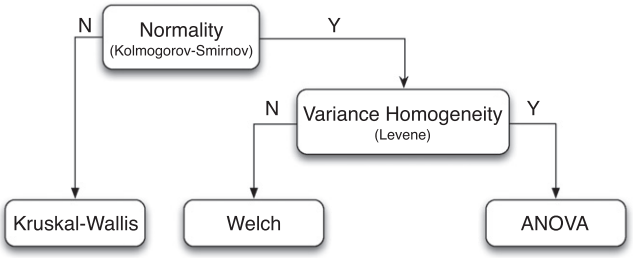


Fig. 7. The general structure of the statistical analysis used in this work.

- Number of independent runs and maximum number of generations: 50 independent runs are conducted for each algorithm on each test instance. The maximum number of generations is 250 for bi-objective instances, and 300 for tri-objective ones, respectively.

The best results are highlighted with boldface and dark background. As discussed in [10,21], all candidate MOEAs are stochastic algorithms, so that the following statistical analysis is necessary for providing confidential comparisons. The general structure of statistical analysis is given in Fig. 7. Firstly, *Kolmogorov–Smirnov* test is performed to check whether the results follow the Gaussian distribution or not. If the results are not Gaussian distributed, the non-parametric *Kruskal–Wallis* test is used to compare the median result of each algorithm; otherwise, the homogeneity of the variances is checked by *Levene* test. *ANOVA* test is invoked to verify the confidence of comparisons if the variances are equal, while *Welch* test is performed to accomplish this task if the variances are different. 95% confidence level is adopted to compare the statistical significance between two competing algorithms, with † indicating that the JGBL variant is significantly better than its competing algorithm, while ‡ representing the opposite scenario.

4.4. Performance comparisons with baseline algorithms

Tables 3 and 4 present the statistical results of GD, SP, HV, while Table 5 gives the values of I_ϵ by each algorithm for each test instance. The bottom rows of Tables 3 and 4 give the ratio of dominant performance for the corresponding metric. Specifically, 4/21 means the corresponding algorithm significantly outperform its competitor on 4 out of 21 instances for the given metric.

From Tables 3–5, we can clearly find that the baseline algorithm can be substantially improved by the incorporation of JGBL paradigm. Especially on the comprehensive metric HV, the outperformances of both JGBL variants are overwhelming

Table 3
Performance comparisons between NSGA-II and NSGA-II + JGBL.

Problems	GD		SP		HV	
	NSGA-II	NSGA-II + JGBL	NSGA-II	NSGA-II + JGBL	NSGA-II	NSGA-II + JGBL
ZDT 1	1.807E−4(6.75E−5)†	**1.357E−4(1.66E−5)**	7.206E−3(5.92E−4)†	**6.643E−3(1.14E−3)**	3.6588(3.99E−4)†	**3.6604(2.11E−4)**
ZDT 2	1.529E−4(2.99E−5)†	**1.354E−4(1.47E−5)**	7.402E−3(6.57E−4)†	**6.892E−3(8.69E−4)**	3.3244(6.07E−4)†	**3.3273(1.45E−4)**
ZDT 3	9.672E−5(4.70E−5)†	**7.852E−5(7.69E−5)**	7.911E−3(7.19E−4)†	**7.165E−3(7.57E−4)**	4.8127(4.02E−4)†	**4.8148(1.17E−4)**
ZDT 4	3.690E−4(1.79E−4)†	**2.375E−5(1.99E−5)**	7.539E−3(7.62E−4)†	**6.734E−3(5.39E−4)**	3.6514(5.52E−3)†	**3.6610(1.62E−4)**
ZDT 6	7.255E−4(7.63E−5)†	**3.186E−4(4.87E−5)**	5.614E−3(4.75E−4)†	**5.227E−3(4.81E−4)**	3.0196(2.66E−3)†	**3.0397(3.66E−3)**
DTLZ 1	6.743E−2(2.25E−1)†	**4.686E−3(2.07E−2)**	3.679E−1(1.66E+0)†	**6.847E−2(1.84E−1)**	0.9472(6.01E−2)†	**0.9671(1.20E−3)**
DTLZ 2	**1.330E−3(1.72E−4)**‡	2.599E−3(1.09E−3)	**5.769E−2(5.00E−3)**‡	6.058E−2(6.31E−3)	7.3261(2.55E−2)	**7.3300(3.15E−2)**
DTLZ 3	9.603E−1(1.09E+0)†	**8.931E−2(3.17E−1)**	3.935E+0(6.54E+0)†	**5.027E−1(1.74E+0)**	0.4963(1.29E+0)†	**7.3466(7.64E−3)**
DTLZ 4	8.362E−3(6.09E−3)†	**5.722E−3(5.14E−3)**	8.297E−2(4.01E−2)†	**6.873E−2(1.65E−2)**	6.8704(6.30E−1)†	**7.3523(1.14E−2)**
DTLZ 5	2.168E−4(7.22E−5)†	**9.198E−5(3.33E−5)**	9.809E−3(7.31E−4)†	**8.557E−3(7.35E−4)**	6.0998(1.12E−3)	**6.1012(1.73E−3)**
DTLZ 6	8.784E−2(1.24E−2)†	**1.008E−5(3.94E−5)**	9.399E−2(2.84E−2)†	**1.128E−2(7.40E−4)**	3.7386(2.98E−1)†	**6.0991(1.07E−3)**
DTLZ 7	3.702E−3(1.08E−3)†	**1.755E−3(1.08E−3)**	**7.150E−2(9.56E−3)**‡	8.949E−2(2.13E−2)	13.0627(9.65E−2)†	**13.2081(8.73E−2)**
WFG 1	3.425E−2(8.71E−3)†	**2.234E−2(1.20E−4)**	**2.109E−2(2.30E−2)**‡	2.568E−2(1.59E−2)	8.5732(6.62E−1)†	**10.1808(3.76E−3)**
WFG 2	**1.053E−3(9.07E−4)**	1.331E−3(8.84E−4)	**1.541E−2(2.12E−3)**‡	1.637E−2(1.41E−2)	11.0217(4.16E−1)†	**11.3750(2.50E−1)**
WFG 3	7.000E−4(8.48E−5)†	**5.812E−4(6.40E−5)**	2.009E−2(1.66E−3)†	**1.861E−2(1.77E−3)**	10.9324(4.76E−3)†	**10.9408(3.11E−3)**
WFG 4	1.435E−4(1.66E−4)†	**1.343E−3(1.52E−4)**	2.175E−2(1.98E−3)	**2.003E−2(2.23E−3)**	8.6620(6.46E−3)†	**8.6761(3.01E−3)**
WFG 5	6.501E−3(4.84E−5)†	**2.788E−3(1.07E−3)**	**2.156E−2(2.24E−3)**	2.191E−2(2.59E−3)	8.1523(3.00E−2)†	**8.6747(4.35E−1)**
WFG 6	2.571E−3(2.52E−3)†	**1.759E−3(2.36E−3)**	2.209E−2(2.33E−3)†	**2.023E−2(1.83E−3)**	8.5249(1.60E−1)†	**8.6234(3.67E−2)**
WFG 7	9.656E−4(6.09E−5)†	**8.677E−4(8.02E−5)**	2.210E−2(1.98E−3)†	**2.020E−2(1.73E−3)**	8.6662(2.61E−3)	**8.6693(3.70E−3)**
WFG 8	2.549E−2(7.94E−3)†	**1.245E−2(6.41E−3)**	2.233E−2(6.29E−3)†	**2.059E−2(2.52E−2)**	7.1386(4.93E−1)†	**8.1914(1.56E−2)**
WFG 9	1.033E−3(2.59E−4)†	**9.880E−4(1.71E−4)**	2.093E−2(2.01E−3)†	**2.000E−2(2.15E−3)**	8.4279(2.43E−2)†	**8.4358(1.72E−2)**
Ratio	2/21	19/21	5/21	16/21	0/21	21/21

† and ‡ denote that the performance of NSGA-II is significantly worse than and better than NSGA-II + JGBL, respectively. And the best mean values are highlighted with bold face with gray background.

Table 4
Performance comparisons between SPEA2 and SPEA2 + JGBL.

Problems	GD		SP		HV	
	SPEA2	SPEA2 + JGBL	SPEA2	SPEA2 + JGBL	SPEA2	SPEA2 + JGBL
ZDT 1	**1.627E−4(5.97E−5)**	1.635E−4(2.55E−5)	3.359E−3(3.67E−4)†	**3.006E−3(3.41E−4)**	3.6588(6.78E−4)†	**3.6603(4.51E−4)**
ZDT 2	1.569E−4(5.22E−5)†	**9.085E−5(1.81E−5)**	3.327E−3(3.40E−4)†	**3.135E−3(2.52E−4)**	3.3240(1.01E−3)†	**3.3270(1.13E−3)**
ZDT 3	9.720E−5(4.45E−5)†	**4.300E−5(2.01E−5)**	4.012E−3(4.64E−4)†	**3.576E−3(4.7E−4)**	4.7967(7.18E−2)†	**4.8140(3.55E−4)**
ZDT 4	5.802E−4(4.61E−4)†	**7.865E−5(7.12E−4)**	3.681E−3(1.15E−4)†	**2.905E−3(2.11E−3)**	3.6460(1.19E−2)†	**3.6607(3.13E−2)**
ZDT 6	1.118E−3(9.76E−4)†	**4.697E−4(1.75E−4)**	5.344E−3(9.97E−3)†	**2.646E−3(2.78E−4)**	3.0155(3.44E−3)†	**3.0287(5.11E−3)**
DTLZ 1	1.384E−1(2.47E−1)†	**2.423E−3(7.93E−4)**	1.067E−1(2.11E+0)†	**1.918E− 2(8.22E−3)**	0.9672(2.81E−2)†	**0.9735(7.83E−4)**
DTLZ 2	**1.246E−3(2.22E−4)**‡	2.284E−3(2.98E−4)	2.364E−2(2.24E−3)†	**2.192E−2(3.03E−3)**	7.3826(1.24E−2)†	**7.3924(8.78E−3)**
DTLZ 3	4.700E−1(7.85E−1)†	**2.425E−2(3.22E−1)**	1.121E+0(2.58E+0)†	**1.250E−1(1.33E+0)**	2.0476(2.67E+0)†	**7.1000(2.37E−2)**
DTLZ 4	**7.557E−3(6.18E−3)**‡	8.262E−3(6.33E−3)	6.495E−2(3.78E−2)†	**6.148E−2(2.33E−2)**	6.9638(6.18E−1)	**7.0262(5.47E−1)**
DTLZ 5	1.930E−4(5.42E−5)†	**7.702E−5(4.53E−5)**	4.684E−3(4.25E−4)†	**3.660E−3(3.15E−4)**	6.1038(2.94E−3)	**6.1042(2.77E−3)**
DTLZ 6	9.044E−2(8.83E−3)†	**4.398E−6(9.76E−5)**	7.161E−2(2.41E−2)†	**4.357E−3(3.22E−4)**	3.6928(2.61E−1)†	**6.1166(3.01E−3)**
DTLZ 7	3.874E−3(1.49E−3)†	**1.562E−3(6.59E−3)**	3.543E−2(6.15E−3)†	**3.461E−2(4.55E−3)**	13.2700(6.03E−1)†	**13.3916(6.85E−1)**
WFG 1	4.093E−2(1.26E−2)†	**2.236E−2(1.35E−2)**	1.518E−2(2.10E−2)†	**5.652E−3(2.48E−3)**	8.2083(7.31E−1)†	**8.8902(5.36E−1)**
WFG 2	7.333E−4(7.80E−4)†	**6.812E−4(6.45E−4)**	**6.942E−3(1.18E−3)**‡	7.828E−3(1.55E−3)	10.9053(4.52E−1)†	**11.4578(5.92E−1)**
WFG 3	5.543E−4(5.31E−5)†	**4.880E−4(3.81E−5)**	**9.475E−3(9.36E−4)**‡	1.094E−2(1.48E−3)	10.9432(3.69E−3)†	**10.9525(2.58E−3)**
WFG 4	1.524E−3(8.57E−5)†	**1.507E−3(5.48E−5)**	1.044E−2(1.29E−3)†	**1.012E−2(1.00E−3)**	8.6675(5.84E−3)†	**8.6822(3.93E−3)**
WFG 5	6.316E−3(2.23E−5)†	**2.084E−3(3.13E−4)**	1.048E−2(9.18E−4)	**2.084E−3(9.00E−4)**	8.1524(2.90E−2)†	**8.6704(5.36E−2)**
WFG 6	3.245E−3(2.84E−3)†	**5.080E−4(5.74E−4)**	1.047E−2(9.64E−4)†	**9.692E−3(8.44E−4)**	8.4764(1.82E−1)†	**8.6539(4.61E−3)**
WFG 7	7.884E−4(4.00E−5)†	**7.420E−4(3.59E−5)**	1.063E−2(1.04E−3)†	**1.013E−2(9.76E−4)**	8.6741(2.69E−3)	**8.6786(2.17E−3)**
WFG 8	2.986E−2(7.40E−3)†	**9.937E−3(8.48E−4)**	2.294E−2(2.69E−2)†	**1.202E−2(4.83E−3)**	6.9265(3.63E−1)†	**8.1943(4.11E−1)**
WFG 9	8.417E−4(3.50E−4)†	**6.840E−4(2.34E−4)**	1.115E−2(1.09E−3)	**1.050E−2(1.44E−3)**	8.4357(2.88E−2)†	**8.4593(2.91E−2)**
Ratio	3/21	18/21	2/21	19/21	0/21	21/21

† and ‡ denote that the performance of the SPEA2 is significantly worse than and better than SPEA2 + JGBL, respectively. And the best mean values are highlighted in bold face with gray background.

Table 5
The comparison results of NSGA-II + JGBL, SPEA2 + JGBL and their corresponding baseline algorithms on binary indicator I_ϵ.

	NSGA-II			SPEA2		
	$\succcurlyeq$	$\preccurlyeq$	$\backsimeq$	$\succcurlyeq$	$\preccurlyeq$	$\backsimeq$
ZDT1	1978	101	421	1872	118	510
ZDT2	2011	87	402	2058	67	375
ZDT3	2005	89	406	2102	77	321
ZDT4	2015	81	404	2146	63	291
ZDT6	2024	90	386	2111	82	307
DTLZ1	2075	86	339	1774	315	411
DTLZ2	1569	458	473	1602	497	401
DTLZ3	2307	10	188	2331	15	154
DTLZ4	2013	208	279	317	1977	206
DTLZ5	944	625	931	898	802	800
DTLZ6	2232	42	226	2187	51	262
DTLZ7	1798	359	343	1877	272	351
WFG1	2110	106	284	2203	93	204
WFG2	2003	205	292	2134	185	181
WFG3	1788	343	369	1702	371	427
WFG4	1769	328	403	1695	410	395
WFG5	1988	253	259	2034	197	269
WFG6	1826	304	370	2041	165	294
WFG7	928	451	1121	1386	413	701
WFG8	2205	71	224	2354	21	125
WFG9	1558	409	533	1216	517	767

$\succcurlyeq$, $\preccurlyeq$ and $\backsimeq$ record the number of times the JGBL variant performs better than, worse than and incomparable to its corresponding baseline algorithm, respectively.

without any exception. Figs. 8 and 9 show the PF approximations obtained in the run with the median HV value of each algorithm for five selected test instances. It is evident that both JGBL variants obtain better approximations than their baseline algorithms in terms of convergence and spread metrics. As shown in Figs. 8d and 9d, both NSGA-II and SPEA2 cannot converge to the PF of DTLZ 3, while their JGBL variants converge well within 300 generations. The outperformance of JGBL variants can be attributed to the additional exploitation of those eliminated non-dominated solutions, by which the exploration ability is improved. WFG 1 can test an MOEA's ability in coping with bias [14]. As shown in Figs. 8a and 9a, even though the fronts obtained by both JGBL variants cannot completely converge to the PF, they are far better than those obtained by their corresponding baseline algorithms. DTLZ 6 also has a strong bias feature [9]. Comparing to the well spreaded and converged curves found by both JGBL variants, fronts obtained by NSGA-II and SPEA2 are not only scattered, but also lack

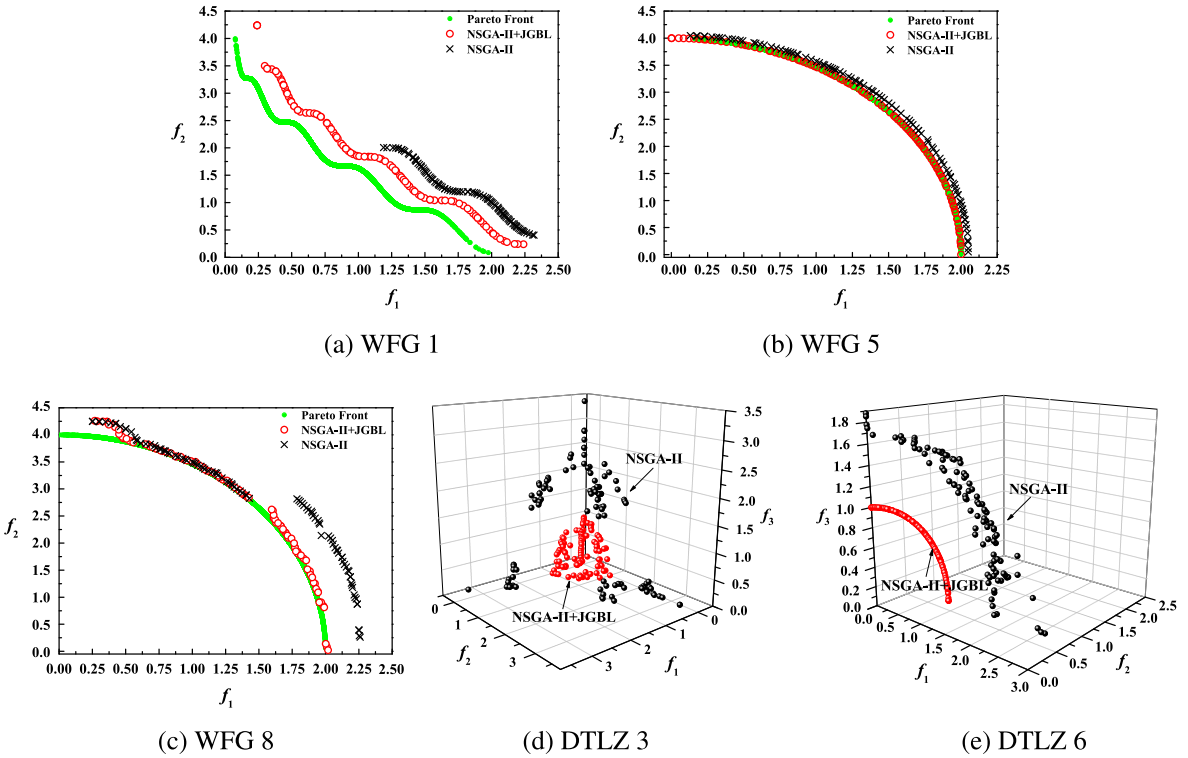


Fig. 8. Final solutions obtained by NSGA-II and NSGA-II + JGBL with median HV values.

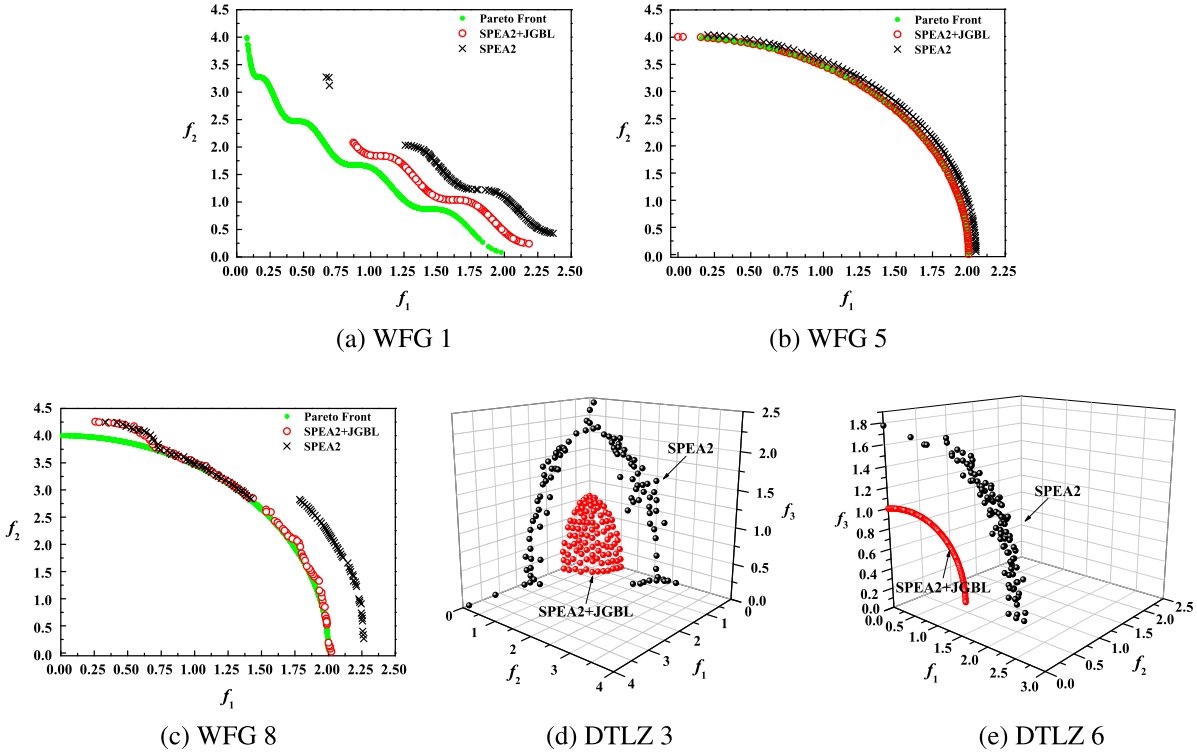


Fig. 9. Final solutions obtained by SPEA2 and SPEA2 + JGBL with median HV values.

of convergence. WFG 5 is a difficult problem with deceptive property [14]. Figs. 8b and 9b give us the clear illustration that the Pareto curves obtained by JGBL variants have wider spread and better convergence. In addition to a significant bias, the distance related parameters of WFG 8 are dependent on position related parameters. From the results shown in Figs. 8c and 9c, we can find that although solutions obtained by the JGBL variants do not cover the entire PF, they are much better than those obtained by the baseline algorithms in terms of convergence and spread metrics. The outperformance of JGBL variants in this case should be attributed to those diversified solutions generated by the JGBL paradigm. In summary, depending on the properties of the test instances, the improvements brought by the JGBL paradigm are different. If the test instance has distinct difficulties, such as multi-modality, bias or deceptive, on which the baseline algorithm cannot tackle well, the incorporation of JGBL paradigm has a larger chance to obtain substantial improvement. Otherwise, the improvement is limited, since the baseline algorithm is able to tackle those test instances well enough.

Fig. 10 presents the evolution of the median HV values versus the number of generations for the JGBL variants and their baseline algorithms. From these subfigures, we clearly find that HV values have been substantially increased by the incorporation of JGBL paradigm. In Fig. 10a, the placid trajectories of NSGA-II and SPEA2 should be ascribed to the local optima in DTLZ 3. As for JGBL variants, the increase of HV values can be explained by the diversified solutions generated by the JGBL paradigm, which helps the algorithm skip from the local optima. In Fig. 10b and c, the trajectories of JGBL variants always lay above those of their baseline algorithms. Concerning the Fig. 10d, all trajectories surge up to a high level within 50 generations, and go to a stable stage afterwards. But the ascending slopes of JGBL variants are always larger than those of their baseline algorithms.

4.5. Sensitivity study on hyper-parameters

Hereinafter, without loss of generality, only NSGA-II is considered as the baseline algorithm to further study the characteristics of JGBL paradigm. Moreover, only the comprehensive metric HV and binary indicator I_ϵ are employed to evaluate the performances of different algorithms. In JGBL paradigm, there are two hyper-parameters, i.e. *jumping percentage* and *jumping rate*, to control the length of transposon and the frequency of JG operations. The effects of these two hyper-parameters are comprehensively investigated as follows.

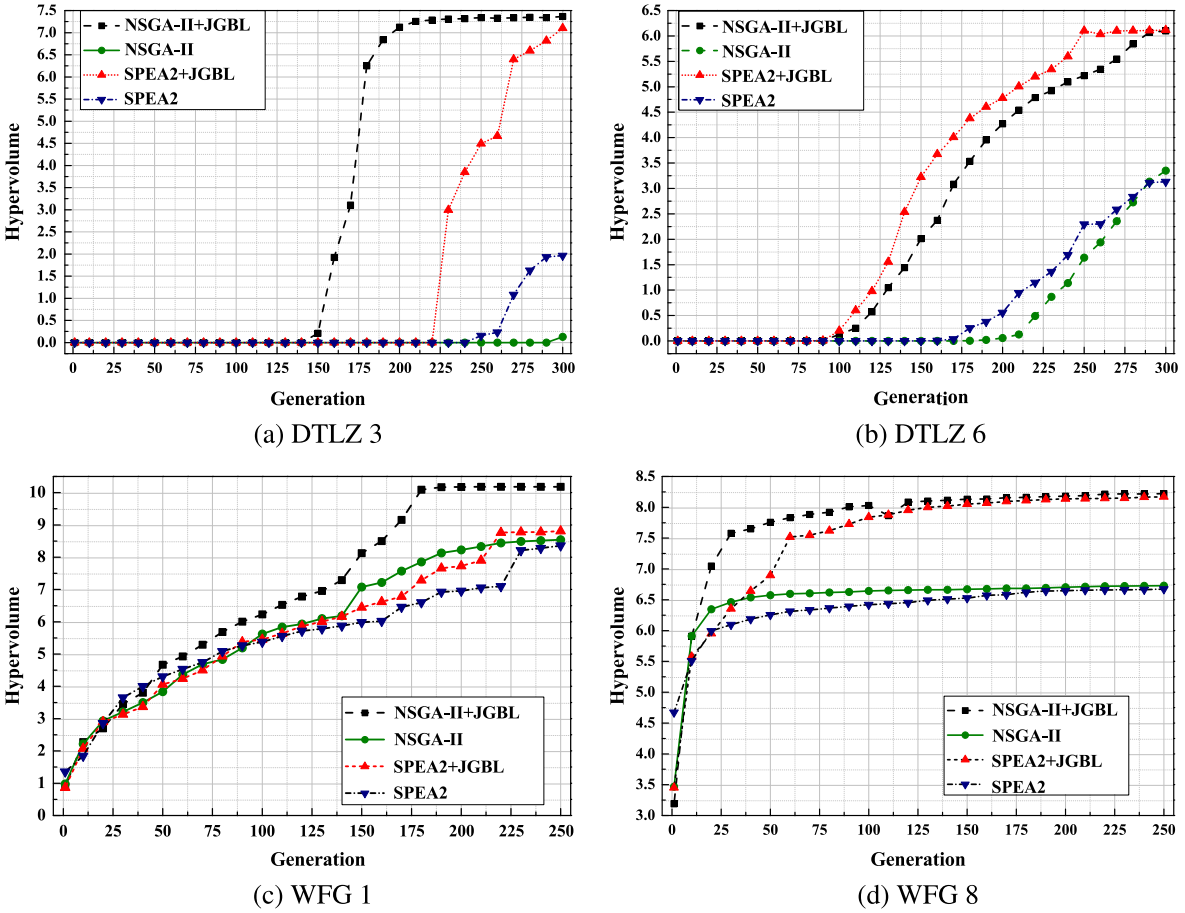


Fig. 10. Evolutionary trajectories of *HV* on four different test instances.

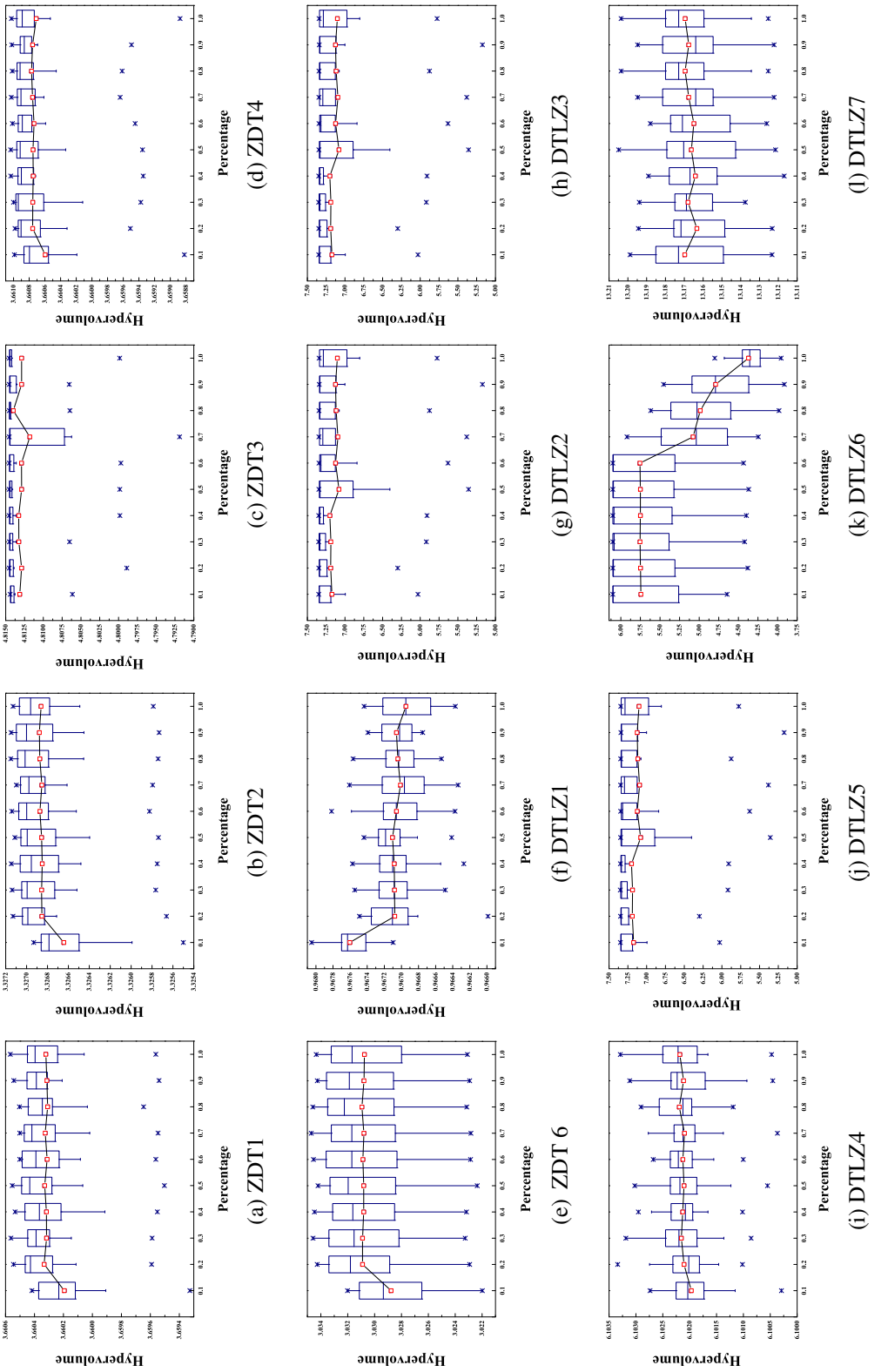


Fig. 11. Sensitivity study of *jumping percentage*.

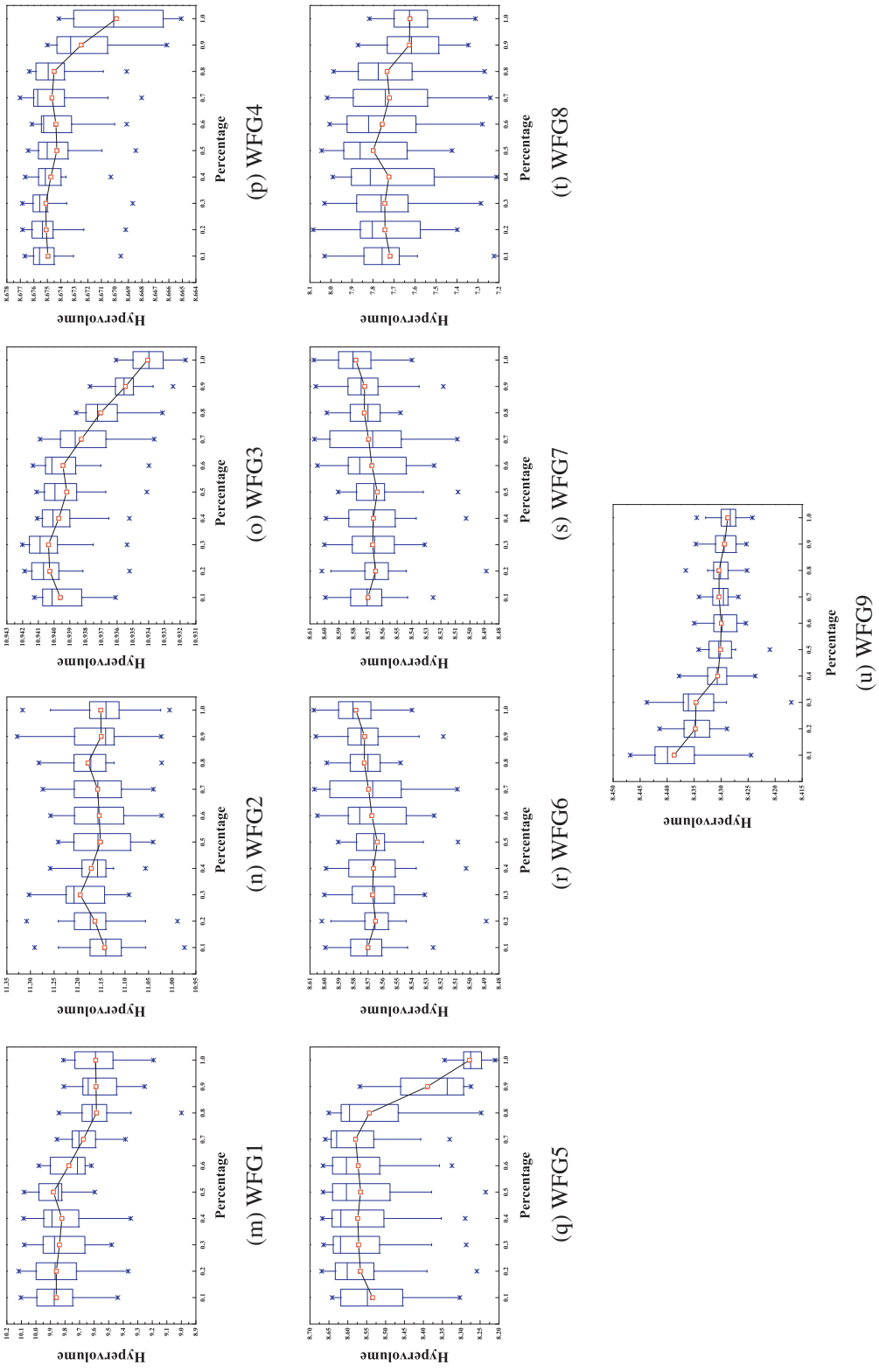


Fig. 11. (*continued*)

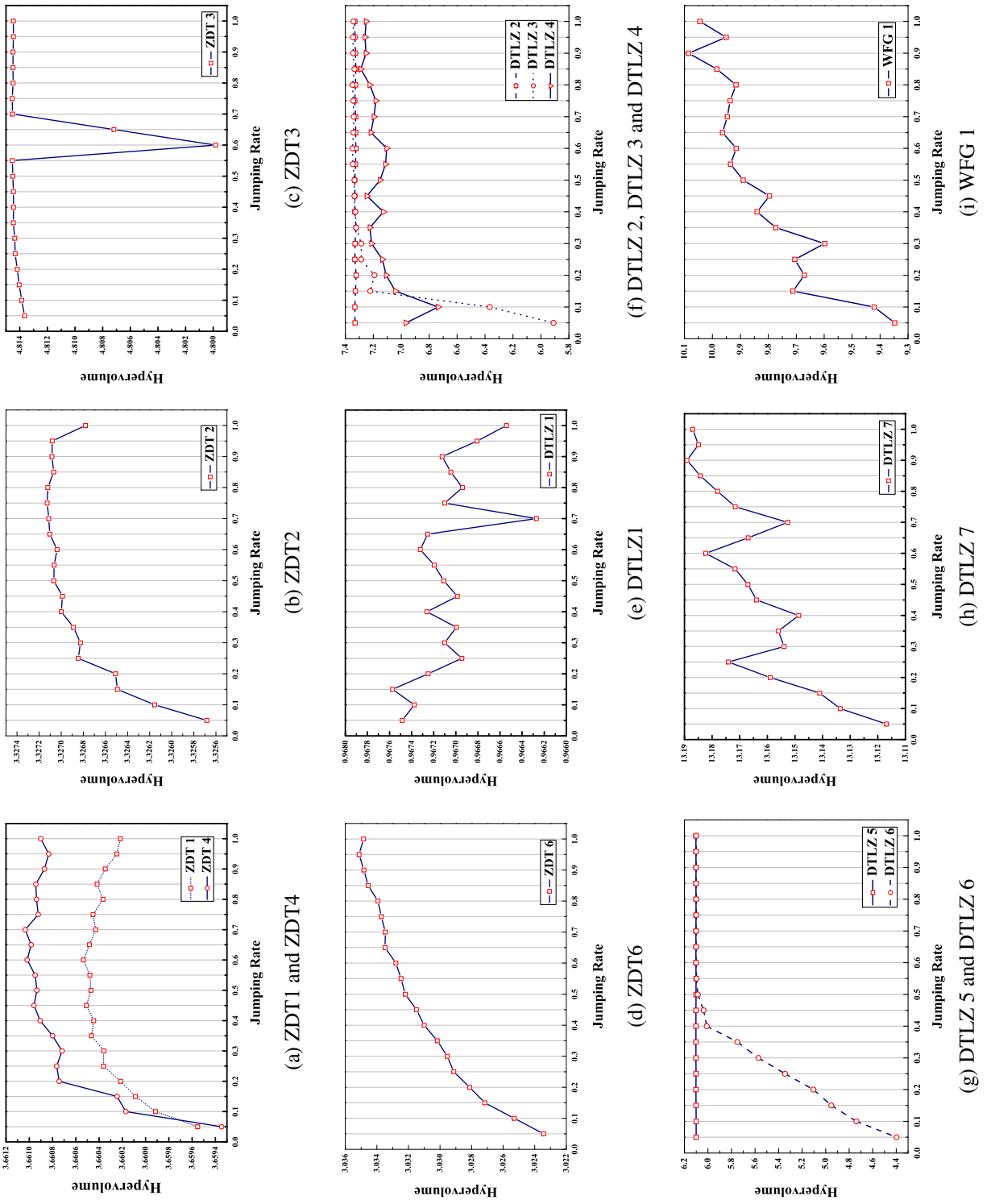


Fig. 12. Sensitivity study of *jumping rate*.

1. Sensitivity study on *jumping percentage*. The benchmark problems and parameter settings are kept the same as in Sections 4.1 and 4.3, except the *jumping percentage* and *jumping rate*. More specifically, *jumping percentage* is updated from 0.1 to 1.0 with the step size of 0.1, and the *jumping rate* is increased from 0.05 to 1.0 with the step size of 0.05 for each

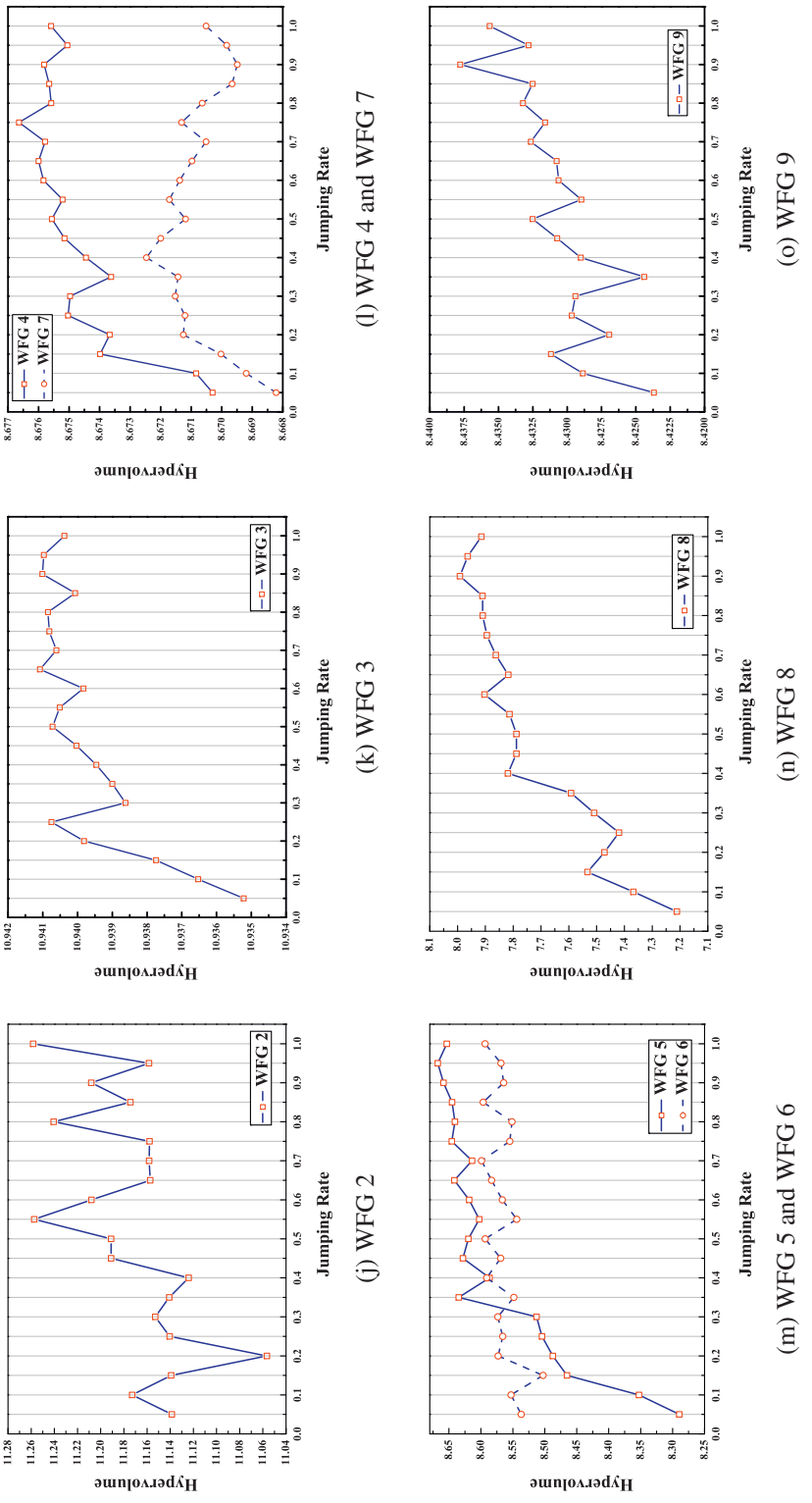


Fig. 12. (*continued*)

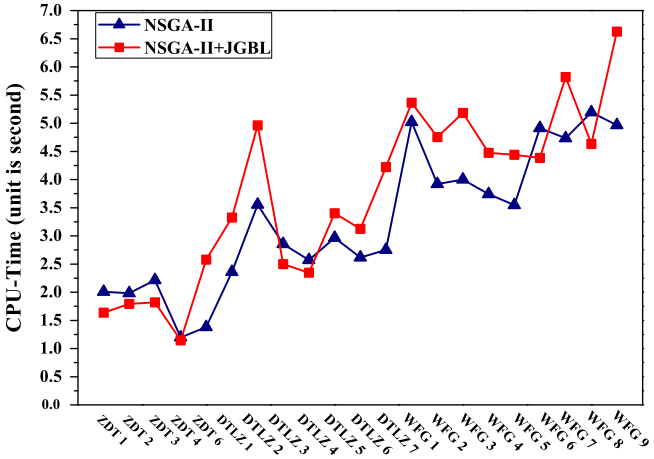


Fig. 13. The CPU-time costs on all test instances.

Table 6
CPU-time costs on three different test suites.

	NSGA-II	NSGA-II + JGBL	Increase rate (%)
ZDT test suite	8.56	10.97	28.2
DTLZ test suite	19.68	26.29	33.6
WFG test suite	34.38	45.68	32.9

investigated *jumping percentage*. In total, we have 10×20 parameter combinations for each test instance. Statistical results are plotted as boxplots in Fig. 11, with the red[1] square indicating the mean HV value obtained by a parameter combination. The vertical axis of each subfigure indicates the HV value and the horizontal axis represents the variation of *jumping percentage*. In general, these 21 subfigures can be divided into three different categories. The first one is featured by a relatively low HV value when the *jumping percentage* is 0.1, then the HV value increases to a high level and keeps stable later on. ZDT 1, ZDT 2, ZDT 4 and ZDT 6 belong to this category. The second category is characterized by a relatively stable tendency of HV trajectory. That is to say, the performance of JGBL paradigm is not sensitive to the length of transposon in this case. It consists of ZDT 3, DTLZ 2 to DTLZ 5 and DTLZ 7, WFG 2, WFG 6 to WFG 8. As for the last category, the HV value surges up to a high level when the *jumping percentage* is small, and then it gradually decreases with the increase of this parameter value. This implies that a large number of gene transpositions deteriorate the performance of JGBL paradigm when dealing with these test instances.

2. Sensitivity study on *jumping rate*. Here the *jumping percentage* is fixed as 0.5 while the *jumping rate* is updated from 0.05 to 1.0 with step size of 0.05. The HV trajectories against different settings of *jumping rate* for all test instances are plotted in Fig. 12. The vertical axis of each subfigure indicates the HV value and the horizontal axis represents the variation of *jumping rate*. These trajectories can be generally divided into five categories. The first one is characterized as that the HV value gradually decreases with the increase of *jumping rate*. Only DTLZ 1 belongs to this category, which indicates that rare occurrence of JG operations is suitable for solving DTLZ 1. The second category, which consists of ZDT 3, DTLZ 2 and DTLZ 5, is featured by a relatively stable trend. It means that the frequency of JG operations has little effect on the algorithm's performance. The HV trajectories of DTLZ 3, DTLZ 4 and DTLZ 6 are marked by the curves whose slopes are steep when the *jumping rate* is small, and then keep stable later. The characteristic of the fourth category is that the HV value increases rapidly when the *jumping rate* is small. However, the trend goes to a steady increasing state afterwards and finally drops down when the *jumping rate* is larger than about 0.75. ZDT 1, ZDT 2, ZDT 4, WFG 3 and WFG 7 can be categorized into this taxonomy. As for the last category, the HV trajectories are with some oscillations, which indicate that it is suitable to use JG operations more frequently in this case.

From the above parametric studies, we conclude that the performance of JGBL paradigm is sensitive to its two hyperparameters. In general, the ranges [0.3,0.6] for *jumping percentage* and [0.5,0.7] for *jumping rate* are recommended for common user.

[1] For interpretation of color in Fig. 11, the reader is referred to the web version of this article.

Table 7
Performance comparison among different JGBL variants, NSGA-II and RJGGA on HV.

	NSGA-II	Variant-I	Variant-II	Variant-III	Variant-IV	RJGGA	NSGA-II + JGBL
ZDT 1	3.6588(3.99E−4)†	**3.6605(1.95E−4)**	3.6595(3.28E−4)	3.6591(3.95E−4)†	3.6600(2.42E−4)	3.6593(3.11E−4)	3.6604(2.11E−4)
ZDT 2	3.3244(6.07E−4)†	3.3271(2.26E−4)	3.3264(3.63E−4)	3.3249(6.23E−4)†	3.3263(3.49E−4)†	3.3252(3.41E−4)†	**3.3273 (1.45E−4)**
ZDT 3	4.8127(4.02E−4)†	4.8145(9.86E−5)	4.8138(2.86E−4)†	4.8131(3.72E−4)†	4.8138(2.14E−4)†	4.8135(2.05E−4)†	**4.8148(1.17E−4)**
ZDT 4	3.6514(5.52E−3)†	3.6594(1.24E−4)†	3.6597(1.61E−3)†	3.6590(7.17E−3)†	3.6595(2.61E−3)†	3.6592(3.10E−3)†	**3.6610(1.62E−4)**
ZDT 6	3.0196(2.66E−3)†	3.0378(1.42E−3)†	3.0316(1.29E−3)†	3.0229(3.24E−3)†	3.0302(1.96E−3)†	3.0284(1.11E−3)†	**3.0397(3.66E−3)**
DTLZ 1	0.9472(6.01E−2)†	0.9678(1.32E−3)	0.9670(1.31E−3)	0.9478(6.23E−2)†	**0.9688(6.93E−4)**‡	0.9611(7.51E−2)†	0.9671(1.20E−3)
DTLZ 2	7.3261(2.55E−2)	7.3271(2.05E−2)	7.3240(2.16E−2)†	7.3254(2.43E−2)†	**7.3683(7.03E−3)**‡	7.3250(4.33E−2)	7.3300(3.15E−2)
DTLZ 3	0.4963(1.29E+0)†	7.3459(1.91E−2)†	7.3163(4.46E−2)†	0.9610(1.89E+0)†	4.8202(2.82E+0)†	6.4533(1.52E+0)†	**7.3466(7.64E−3)**
DTLZ 4	6.8704(6.30E−1)†	**7.3594(5.10E−1)**	7.3361(1.70E−2)†	6.8778(5.98E−2)†	7.0425(4.01E−1)†	7.3075(3.42E−2)†	7.3523(1.14E−2)
DTLZ 5	6.0998(1.12E−3)†	**6.1021(1.70E−3)**‡	6.1004(1.75E−4)	6.1000(1.47E−3)	6.1009(1.61E−3)	6.1005(1.81E−3)	6.1012(1.73E−3)
DTLZ 6	3.7386(2.98E−1)†	6.1013(3.13E−3)	3.1508(5.04E−1)†	4.0019(2.60E−1)†	**6.1019(1.59E−3)**‡	5.5438(2.47E−1)†	6.0991(1.07E−3)
DTLZ 7	13.0627(9.65E−2)†	13.1504(8.14E−2)†	13.0860(1.17E−1)†	13.0738(9.29E−2)†	13.1373(7.96E−2)†	13.1041(8.72E−2)†	**13.2081(8.73E−2)**
WFG 1	8.5732(6.62E−1)†	10.1692(3.53E−3)†	9.6220(5.43E−1)†	8.7059(7.65E−1)†	9.5592(3.83E−1)†	10.0049(5.20E−1)†	**10.1808(3.76E−3)**
WFG 2	11.0217(4.16E−1)†	11.3792(3.92E−1)	11.2211(3.73E−1)†	11.0058(4.15E−1)†	**11.4578(7.37E−2)**‡	11.2958(6.46E−1)†	11.3750(2.50E−1)
WFG 3	10.9324(4.76E−3)†	10.9368(5.24E−3)†	10.9332(4.16E−2)†	10.9331(3.16E−3)†	10.9326(3.59E−3)†	10.9352(4.02E−3)†	**10.9408(3.11E−3)**
WFG 4	8.6620(6.46E−3)†	8.6722(5.78E−3)†	8.6639(5.39E−3)†	8.6641(4.67E−3)†	8.6662(5.91E−3)†	8.6683(5.82E−3)†	**8.6761(3.01E−3)**
WFG 5	8.1523(3.00E−2)†	**8.6755(1.42E−1)**	8.1639(6.17E−2)†	8.1615(3.22E−2)†	8.1637(3.23E−2)†	8.3056(5.31E−2)†	8.6747(4.35E−1)
WFG 6	8.5249(1.60E−1)	8.6136(1.49E−1)	8.6063(9.35E−2)†	8.5137(1.61E−2)†	8.6091(1.22E−1)†	8.6022(2.01E−1)†	**8.6234(3.67E−2)**
WFG 7	8.6662(2.61E−3)†	**8.6717(2.69E−3)**‡	8.6654(3.31E−3)	8.6674(2.79E−3)	8.6678(2.12E−3)	8.6670(4.01E−3)	8.6693(3.70E−3)
WFG 8	7.1386(4.93E−1)†	**8.1942(4.25E−1)**	8.1298(2.75E−1)†	7.1684(4.81E−1)†	7.0413(4.69E−1)†	8.1423(5.72E−1)†	8.1914(1.56E−2)
WFG 9	8.4279(2.43E−2)†	8.4292(1.40E−2)†	8.4281(1.39E−2)†	8.4321(1.91E−2)	8.4308(1.79E−2)	8.4301(1.27E−2)	**8.4358(1.72E−2)**
Ratio	0/21	6/21	0/21	0/21	4/21	0/21	11/21

† and ‡ denote that the performance of the corresponding algorithm is significantly worse than and better than NSGA-II + JGBL, respectively. And the best mean values are highlighted in bold face with gray background.

Table 8
The comparison results of NSGA-II + JGBL and several other JG variants on the binary indicator I_ϵ.

Problem	Variant-I			Variant-II			Variant-III			Variant-IV			RJGGA		
	$\succcurlyeq$	$\preccurlyeq$	$\precsim$	$\succcurlyeq$	$\preccurlyeq$	$\precsim$	$\succcurlyeq$	$\preccurlyeq$	$\precsim$	$\succcurlyeq$	$\preccurlyeq$	$\precsim$	$\succcurlyeq$	$\preccurlyeq$	$\precsim$
ZDT1	881	435	1184	1270	251	979	1797	132	571	1176	283	1041	1388	197	915
ZDT2	1089	242	1169	1201	199	1100	1995	120	385	1673	170	657	1689	154	657
ZDT3	985	406	1109	1536	264	700	1873	205	422	1504	273	723	1802	211	487
ZDT4	2002	100	398	1952	127	421	1899	115	486	1876	130	494	1998	107	395
ZDT6	1823	201	476	1934	183	383	1992	180	328	1989	176	335	2015	148	337
DTLZ1	682	691	1127	723	640	1137	2143	71	286	252	2017	231	1676	302	522
DTLZ2	1581	501	418	1867	354	279	1815	361	324	276	1995	229	1807	315	378
DTLZ3	1766	299	435	1888	216	396	2294	21	185	2071	62	367	1944	188	368
DTLZ4	798	821	881	1883	317	300	1991	213	296	1912	285	303	1799	259	442
DTLZ5	311	1895	294	900	743	857	904	802	794	888	902	710	891	725	884
DTLZ6	808	814	878	2076	113	311	1992	136	372	361	1724	415	1978	295	227
DTLZ7	1696	302	502	1954	292	254	2005	301	194	1893	268	339	1875	328	297
WFG1	1878	323	299	1971	286	243	2105	152	243	1968	301	231	1899	282	319
WFG2	777	761	962	1827	289	384	1949	211	340	311	1859	330	1831	252	417
WFG3	1802	322	376	1840	284	376	1911	295	294	1975	227	298	1799	363	338
WFG4	1713	361	426	1804	297	399	1797	344	359	1755	320	425	1811	334	355
WFG5	723	876	901	2113	202	184	2072	224	204	1992	273	235	1904	318	278
WFG6	915	367	1218	1756	310	434	1882	287	331	1801	371	328	1845	219	436
WFG7	511	1419	570	1091	478	931	844	530	1126	828	493	1179	972	367	1161
WFG8	879	721	900	1751	382	367	1922	208	370	2045	197	258	1817	414	269
WFG9	1511	385	604	1604	402	494	1305	601	594	1311	529	660	1455	514	531

$\succcurlyeq$, $\preccurlyeq$ and $\precsim$ record the number of times NSGA-II + JGBL performs better than, worse than and incomparable to the corresponding algorithms, respectively.

Table 9
Performance comparisons between MOEA/D and NSGA-II + JGBL.

Problems	GD		SP		HV	
	MOEA/D	NSGA-II + JGBL	MOEA/D	NSGA-II + JGBL	MOEA/D	NSGA-II + JGBL
ZDT 1	1.650E−4(4.75E−5)†	**1.357E−4(1.66E−5)**	6.913E−3(2.54E−4)†	**6.643E−3(1.14E−3)**	3.6600(5.69E−4)	**3.6604(2.11E−4)**
ZDT 2	1.471E−4(3.53E−5)	**1.354E−4 (1.47E−5)**	**4.450E−3 (8.78E−4)**‡	6.892E−3(8.69E−4)	3.3251 (3.24E−4)	**3.3273(1.45E−4)**
ZDT 3	8.532E−5(7.29E−5)†	**7.852E− 5(7.69E−5)**	2.239E−2(1.56E−4)†	**7.165E−3(7.57E−4)**	4.8147(1.18E−4)	**4.8148(1.17E−4)**
ZDT 4	3.111E−5(2.11E−5)†	**2.375E−5 (1.99E−5)**	7.374E−3(7.81E−4)†	**6.734E−3(5.39E−4)**	3.6602 (2.19E−4)†	**3.6610(1.62E−4)**
ZDT 6	3.199E−4(3.49E−5)	**3.186E−4(4.87E−5)**	**2.869E−3(4.93E−4)**‡	5.227E −3(4.81E−4)	3.0391(3.18E−3)	**3.0397(3.66E−3)**
DTLZ 1	**4.507E−3(2.98E−2)**	4.686E−3 (2.07E−2)	7.136E−2(3.11E−2)†	**6.847E −2(1.84E−1)**	0.9563(4.99E−3)†	**0.9671(1.20E−3)**
DTLZ 2	**9.388E−4(5.91E−4)**‡	2.599E−3(1.09E−3)	7.216E−2(4.33E−2)†	**6.058E−2(6.31E−3)**	**7.3649 (4.33E−2)**‡	7.3300(3.15E−2)
DTLZ 3	**5.601E−2(1.216E−1)**‡	8.931E−2(3.17E−1)	8.418E−1(1.48E+0)†	**5.027E−1(1.74E+0)**	**7.3589(1.38E−2)**	7.3466(7.64E−3)
DTLZ 4	5.828E−3(3.92E−3)†	**5.722E−3 (5.14E−3)**	8.383E−2(3.06E−2)†	**6.873E−2**(1.65E−2)	7.3058(7.03E−2)†	**7.3523(1.14E−2)**
DTLZ 5	**4.401E−5(1.17E−5)**‡	9.198E−5(3.33E−5)	9.745E−3(1.81E−3)†	**8.557E−3(7.35E−4)**	6.0896(3.65E−3)	**6.1012(1.73E−3)**
DTLZ 6	**8.429E−6(8.78E−6)**‡	1.008E−5(3.94E−5)	2.672E−2(7.02E−4)†	**1.128E−2(7.40E−4)**	5.9896 (3.14E−3)†	**6.0991(1.07E−3)**
DTLZ 7	8.370E−3(7.98E−4)†	**1.755E−3 (1.08E−3)**	1.227E−1(8.72E−2)†	**8.949E−2(2.13E−2)**	13.1038 (8.71E−2)†	**13.2081(8.73E−2)**
WFG 1	3.939E−2(1.57E−4)†	**2.234E−2 (1.20E−4)**	2.979E−2(2.03E−2)†	**2.568E−2(1.59E−2)**	9.0600 (5.19E−3)†	**10.1808(3.76E−3)**
WFG 2	2.173E−3(8.36E−1)†	**1.331E−3 (8.84E−4)**	5.077E−2(2.67E−2)†	**1.637E−2(1.41E−2)**	11.3188 (3.09E−1)†	**11.3750(2.50E−1)**
WFG 3	**5.041E−4(5.97E−5)**‡	5.812E−4(6.40E−5)	2.470E−2(2.12E−3)†	**1.861E−2(1.77E−3)**	10.9405(4.02E−3)	**10.9408(3.11E−3)**
WFG 4	2.687E−3(1.87E−4)†	**1.343E−3 (1.52E−4)**	2.346E−2(2.81E−3)†	**2.003E−2(2.23E−3)**	8.5383 (4.14E−3)†	**8.6761(3.01E−3)**
WFG 5	6.052E−3(1.11E−3)†	**2.788E−3 (1.07E−3)**	2.797E−2(2.61E−3)†	**2.191E−2(2.59E−3)**	8.3185 (5.10E−1)†	**8.6747(4.35E−1)**
WFG 6	**9.521E−4(4.09E−4)**‡	1.759E−3(2.36E−3)	2.601E−2(1.96E−3)†	**2.023E−2(1.83E−3)**	**8.6681 (4.01E−1)**‡	8.6234(3.67E−2)
WFG 7	9.433E−4(9.05E−5)†	**8.677E− 4(8.02E−5)**	2.620E−2(1.95E−3)†	**2.020E−2(1.73E−3)**	8.6622(3.52E−3)	**8.6693(3.70E−3)**
WFG 8	1.760E−2(7.25E−3)†	**1.245E−2 (6.41E−3)**	2.550E−2(3.74E−1)†	**2.059E−2(2.52E−2)**	7.7569 (1.77E−2)†	**8.1914(1.56E−2)**
WFG 9	**8.145E−4(1.27E−2)**‡	9.880E−4(1.71E−4)	2.580E−2(1.82E−3)†	**2.000E−2(2.15E−3)**	8.4256 (2.39E−2)†	**8.4358(1.72E−2)**
UF 1	**1.023E−2(3.52E−3)**	1.108E−2 (3.14E−3)	1.161E−1(8.08E−2)†	**1.011E −1(7.52E−2)**	**3.5319(7.09E−2)**	**3.5310(7.51E−2)**
UF 2	1.169E−2(2.23E−3)†	**1.107E−2 (1.33E−3)**	**1.110E−2(4.17E −3)**‡	1.211E−2(5.87E−3)	**3.6005(4.65E−2)**	3.5847(4.22E−2)
UF 3	1.598E−3(3.49E−4)	**1.579E−3(4.18E−4)**	6.795E−3(2.77E−4)†	**6.607E−3(3.09E−4)**	3.6440(2.73E−1)	**3.6512(1.99E−1)**
UF 4	1.073E−2(5.13E−3)	**1.066E−2(5.99E−3)**	9.025E−3(8.73E−4)†	**8.869E−3 (9.01E−4)**	3.0741(3.34E−3)†	**3.0822(3.01E−3)**
UF 5	8.601E−2(2.42E−2)†	**8.455E−2 (1.89E−2)**	4.513E−2(1.55E−2)†	**4.303E−2(1.14E−2)**	1.5152 (4.16E−1)†	**1.9823(4.51E−1)**
UF 6	**2.201E−2(5.17E−3)**‡	2.281E−2(5.63E−3)	1.335E−2(5.56E−3)	**1.329E−2(5.71E−3)**	3.1490(5.11E−1)†	**3.1510(5.01E−1)**
UF 7	7.199E−3(2.35E−4)†	**7.020E −3(3.05E−4)**	**3.112E−2 (2.33E−2)**	3.117E−2(2.95E−2)	3.4199(3.68E−1)†	**3.4521(3.15E−1)**
UF 8	**2.125E−2(4.03E−3)**‡	2.305E−2(3.87E−3)	9.393E−2(1.00E−1)	**9.101E −2(1.97E−1)**	**5.3131 (1.18E+0)**‡	5.3122(1.80E+0)
UF 9	**1.086E−2(3.28E−3)**	1.103E−2 (3.10E−3)	1.609E−1(9.24E−1)†	**1.518E −1(8.01E−1)**	3.8061(1.09E+0)†	**3.8211(1.01E+0)**
UF 10	1.036E−2(3.53E−3)†	**1.001E− 1(2.99E−3)**	2.166E−1(1.01E+0)†	**2.033E−1(1.25E+0)**	4.1176(1.61E+0)†	**4.1582(1.55E+0)**
Ratio	12/31	19/31	4/31	27/31	6/31	25/31

† and ‡ denote that the performance of MOEA/D is significantly worse than and better than NSGA-II + JGBL, respectively. And the best mean values are highlighted in bold face with gray background.

4.6. CPU-time cost

The average CPU-time consumed by NSGA-II and NSGA-II + JGBL for all test instances are plotted in Fig. 13, and they are further categorized into three different groups in Table 6. The last column of Table 6 records the increase rates of CPU-time brought by the incorporation of JGBL paradigm. According to Table 6 and Fig. 13, it is observed that good performances achieved by the incorporation of JGBL paradigm do not come for free. For most of the instances, the CPU-time costs of NSGA-II + JGBL are larger than NSGA-II. More specifically, as shown in Table 6, the increase rate of CPU-time cost is 28.2% when solving ZDT instances. However, as shown in Fig. 13, NSGA-II + JGBL spends more CPU-time only in solving ZDT 6, whereas it costs even less CPU-time than NSGA-II in solving the other ZDT instances. This might be explained as NSGA-II also generates many non-dominated solutions when solving those test instances. As for the DTLZ and WFG test suites, NSGA-II + JGBL is more time-consuming than the baseline NSGA-II, except for DTLZ 3, DTLZ 4, WFG 6 and WFG 8. The extra CPU-time should come from the additional round environmental selection.

4.7. Verification of the underlying rationale of JGBL paradigm

The effectiveness of our proposed JGBL paradigm has been fully demonstrated from the previous empirical studies. However, two questions are still unclear: (1) whether the superiority of JGBL paradigm really benefits from its motivated rationale? (2) How about the performance comparison with the existing JGGAs? To this end, NSGA-II + JGBL is modified into four different variants, denoted as Variant-I to Variant-IV. In addition, our previously proposed RJGGA [24] is used as another competing algorithm to answer the second question, in view of its good performances for solving continuous MOPs. Technical details of four artificial variants are given as follows.

1. Variant-I: The JG operation is employed to operate upon the entire hybrid population (i.e. R_t in Fig. 5) after the first round environmental selection. Afterwards, the mutant population is merged with the non-dominated solutions survived from the first round environmental selection. Then another round environmental selection is raised to filter out the parent population for the next generation.
2. Variant-II: Instead of exploiting the non-dominated solutions, this variant aims at mining information from the dominated solutions in the hybrid population.
3. Variant-III: It uses traditional genetic search operators, i.e. SBX and polynomial mutation, to replace the JG operators in the original JGBL paradigm.
4. Variant-IV: It uses *Differential Evolution* (DE) [29] operator to replace the JG operators in the original JGBL paradigm. Here DE/rand/1/bin operator is chosen with $CR = 0.2$ and $F = 0.2$, as recommended in [17].

The benchmark problems and parameter settings are kept the same as in Sections 4.1 and 4.3. The performances of all algorithms are evaluated by the HV metric and binary indicator I_ϵ, and the results are tabulated in Tables 7 and 8, respectively. From these results, it is clear to see that the original JGBL paradigm is the best choice at hand, as it wins on 11 out of 21 test instances. As for the other variants, Variant-I, which performs best, wins on six out of 21 test instances with two significantly better ones. Its good performance can be explained by the secondary exploitation on the non-dominated solutions that are eliminated by the first round environmental selection. However, in case the maximum number is fixed in advance, the exploitation upon those dominated solutions can be regarded as a waste of function evaluations, since very little useful information can be extracted from them. This explains why Variant-I cannot outperform the original JGBL paradigm. From the HV values obtained

Table 10
The comparison results of NSGA-II + JGBL and MOEA/D on binary indicator I_ϵ.

Problems	$\succcurlyeq$	$\preccurlyeq$	$\backsimeq$	Problems	$\succcurlyeq$	$\preccurlyeq$	$\backsimeq$
ZDT1	1772	140	588	WFG5	1899	333	268
ZDT2	1706	135	659	WFG6	318	1784	398
ZDT3	1802	211	487	WFG7	1115	454	931
ZDT4	1998	107	395	WFG8	2013	206	281
ZDT6	2015	148	337	WFG9	1461	501	538
DTLZ1	1676	302	522	UF1	784	812	904
DTLZ2	1807	315	378	UF2	744	855	901
DTLZ3	1944	188	368	UF3	1611	388	495
DTLZ4	1799	259	442	UF4	1574	325	601
DTLZ5	891	725	884	UF5	1889	244	367
DTLZ6	1978	295	227	UF6	1396	593	511
DTLZ7	1875	328	297	UF7	1727	343	430
WFG1	1899	282	319	UF8	699	1250	551
WFG2	1831	252	417	UF9	1382	460	658
WFG3	1658	428	414	UF10	1701	391	408
WFG4	1901	259	340				

$\succcurlyeq$, $\preccurlyeq$ and $\backsimeq$ record the number of times NSGA-II + JGBL performs better than, worse than and incomparable to MOEA/D, respectively.

by Variant-II, we find that the exploitation upon the dominated solutions in JGBL paradigm is not a wise choice. This can be explained by that little useful information contained in the dominated solutions, thus elite building blocks can be rarely generated. Regarding the other two variants, which replace the JG operators with the traditional genetic search operators and DE operator, respectively, the outperformance of the original JGBL paradigm is still significant. Specifically, Variant-IV shows significantly better performance on four out of 21 test instances, while Variant-III is the worst variant. Comparing to RJGGA, NSGA-II + JGBL wins on all test instances, and shows significantly better performance on 16 of them. Anyway, all JGBL variants and RJGGA outperform the baseline NSGA-II. Especially on DTLZ 3 and WFG 8, which are featured by the multi-modality, only MOEAs with JG operators are able to obtain satisfactory results. Based on the above discussions, we conclude that the outperformance achieved by the JGBL paradigm should benefits from its underlying rationale, which is an intelligent cooperation of horizontal and vertical gene movements.

4.8. Performance comparisons with the state-of-the-art MOEA/D

In this section, the performance of NSGA-II + JGBL is further compared with the state-of-the-art MOEA/D [18]. In addition to the benchmark problems introduced in Section 4.1, 10 unconstrained instances [37], UF 1 to UF 10, are also considered in the empirical study here. The parameters of MOEA/D are set the same as in its original paper [36], while those of NSGA-II + JGBL are set the same as in Section 4.3, except that on UF 1 to UF 7 we evolve 300 generations and 500 generations are required on UF 8 to UF 10. The performance of each algorithm is validated by the four metrics given in Section 4.2. As shown in Tables 9 and 10, it is clear that NSGA-II + JGBL shows very competitive performance to MOEA/D. More specifically, both NSGA-II + JGBL and MOEA/D perform comparatively similar on the relatively simple ZDT instances. As for DTLZ instances, MOEA/D obtains better results on GD, while NSGA-II + JGBL performs better on SP, HV and I_ϵ metrics, and similar trends can be found on WFG instances. As for the UF instances, which are with complicated PS and nonlinear parameter dependencies, NSGA-II + JGBL still shows very competitive performances. In summary, the performances of NSGA-II + JGBL are better than MOEA/D in terms of convergence and diversity for most of the cases. This owes to the enhanced exploration ability brought by the JGBL paradigm, which is able to generate more elite building blocks, and thus a better grained search is guaranteed.

5. Conclusion

In this paper, we proposed the JGBL paradigm to enhance the exploration ability of MOEAs. It adapts the JG operation to the continuous search space. The underlying rationale of JGBL paradigm is exploiting the non-dominated solutions eliminated by the environmental selection. From the empirical studies, significant improvements were found by the incorporation of JGBL paradigm for the baseline MOEAs and its underlying rationale was also validated.

Acknowledgements

The authors are grateful to Mr. Biao Luo for his valuable suggestions on this paper. This work was jointly supported part by the Natural Science Foundation of China Grant No. 61272289 and City University of Hong Kong Strategic Grant No. 7002826.

References

[1] A, Borges Simoes, E. Costa, Enhancing transposition performance, in: Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99), vol. 2, 1999, pp. 1434–1441.
[2] T. Chan, K. Man, K. Tang, S. Kwong, A jumping gene paradigm for evolutionary multiobjective optimization, IEEE Transactions on Evolutionary Computation 12 (2) (2008) 143–159.
[3] T.M. Chan, K.F. Man, K.S. Tang, S. Kwong, A jumping-genes paradigm for optimizing factory wlan network, IEEE Transactions On Industrial Informatics 3 (1) (2007) 33–43.
[4] C.A. Coello, G.B. Lamont, D.A. Van Veldhuizen, Evolutionary Algorithms for Solving Multi-Objective Problems, 2nd ed., Genetic and Evolutionary Computation, Springer, New York, 2007.
[5] K. Deb, Multi-Objective Optimization Using Evolutionary Algorithms, John Wiley & Sons, 2001.
[6] K. Deb, R.B. Agrawal, Simulated binary crossover for continuous search space, Complex Systems 9 (1994) 1–34.
[7] K. Deb, M. Goyal, A combined genetic adaptive search (geneas) for engineering design, Computer Science and Informatics 26 (1996) 30–45.
[8] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.
[9] K. Deb, L. Thiele, M. Laumanns, E. Zitzler, Scalable test problems for evolutionary multiobjective optimization, in: L. Jain, X. Wu, A. Abraham, L. Jain, R. Goldberg (Eds.), Evolutionary Multiobjective Optimization, Advanced Information and Knowledge Processing, Springer, Berlin Heidelberg, 2005, pp. 105–145.
[10] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm and Evolutionary Computation 1 (1) (2011) 3–18.
[11] M. Gong, L. Jiao, L. Zhang, Baldwinian learning in clonal selection algorithm for optimization, Information Sciences 180 (8) (2010) 1218–1236.
[12] F. Herrera, M. Lozano, J.L. Verdegay, Tackling real-coded genetic algorithms: operators and tools for behavioural analysis, Artificial Intelligence Review 12 (1998) 265–319.
[13] J.H. Holland, Adaptation in Natural and Artificial Systems, MIT Press, Cambridge, MA, USA, 1975.
[14] S. Huband, P. Hingston, L. Barone, L. While, A review of multiobjective test problems and a scalable test problem toolkit, IEEE Transactions on Evolutionary Computation 10 (5) (2006) 477–506.

[15] L. Ke, F. Álvaro, K. Sam, Multi-Objective differential evolution with adaptive control of parameters and operators, in: C. Coello et al. (Eds.), LION'11: Proc. 5th International Conference on Learning and Intelligent Optimization, vol. 6683, Springer, Rome, Italy, 2011, pp. 473–487.
[16] L. Ke, K. Sam, C. Jingjing, L. Miqing, Z. Jinhua, S. Ruimin, Achieving balance between proximity and diversity in multi-objective evolutionary algorithm, Information Sciences 182 (1) (2012) 220–242.
[17] S. Kukkonen, J. Lampinen, GDE3: the third evolution step of generalized differential evolution, in: Proceedings of 2005 IEEE Congress on Evolutionary Computation, IEEE Press, 2005, pp. 443–450.
[18] H. Li, Q. Zhang, Multiobjective optimization problems with complicated pareto sets, MOEA/D and NSGA-II, IEEE Transactions on Evolutionary Computation 12 (2) (2009) 284–302.
[19] H.M. Ma, K.T. Ng, K.F. Man, Multiobjective coordinated power voltage control using jumping genes paradigm, IEEE Transactions On Industrial Electronics 55 (11) (2008) 4075–4084.
[20] K. Miettinen, Nonlinear Multiobjective Optimization, vol. 12, Kluwer Academic Publisher, 1999.
[21] A.J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J.J. Durillo, A. Beham, AbYSS: Adapting scatter search to multiobjective optimization, IEEE Transactions on Evolutionary Computation 12 (4) (2008) 439–457.
[22] S. Ono, Y. Hirotani, S. Nakayama, A memetic algorithm for robust optimal solution search-hybridization of multi-objective genetic algorithm and quasi-newton method, International Journal of Innovative Computing, Information and Control 5 (12B) (2009) 5011–5020.
[23] B.Y. Qu, P.N. Suganthan, Multi-objective evolutionary algorithms based on the summation of normalized objectives and diversified selection, Information Sciences 180 (2010) 3170–3181.
[24] K.S.N. Ripon, S. Kwong, K. Man, A real-coding jumping gene genetic algorithm (RJGGA) for multiobjective optimization, Information Sciences 177 (2) (2007) 632–654.
[25] J.D. Schaffer, Multiple objective optimization with vector evaluated genetic algorithms, in: Proceedings of the 1st International Conference on Genetic Algorithms, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1985, pp. 93–100.
[26] J.R. Schott, Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization, Master's thesis, Massachusetts Institute of Technology. Dept. of Aeronautics and Astronautics, 1995.
[27] A.V. Spirov, A.B. Kazansky, Jumping genes-mutators can rise efficacy of evolutionary search, in: Proceedings of the 2002 Genetic and Evolutionary Computation Conference (GECCO'02), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002, pp. 561–568.
[28] N. Srinivas, K. Deb, Multiobjective optimization using nondominated sorting in genetic algorithms, Evolutionary Computation 2 (3) (1994) 221–248.
[29] R. Storn, K. Price, Differential evolution:a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (1997) 341–359.
[30] Sue Wall, J.M. Kiran Ghanekar, J.W. Dale, Context-sensitive transposition of is6110 in mycobacteria, Microbiology 145 (1999) 3169–3176.
[31] K.-S. Tang, R.J. Yin, S. Kwong, K.T. Ng, K.-F. Man, A theoretical development and analysis of jumping gene genetic algorithm, IEEE Transactions on Industrial Informatics 7 (3) (2011) 408–418.
[32] D.A.V. Veldhuizen, G.B. Lamont, Evolutionary computation and convergence to a pareto front, in: Stanford University, California, Morgan Kaufmann, 1998, pp. 221–228.
[33] Y. Wang, Y. Yang, Particle swarm optimization with preference order ranking for multi-objective optimization, Information Sciences 179 (2009) 1944–1959.
[34] L.D. Whitley, V.S. Gordon, K.E. Mathias, Lamarckian evolution, the Baldwin effect and function optimization, in: PPSN, 1994, pp. 6–15.
[35] X.-S. Yang, K.T. Ng, S.H. Yeung, K.F. Man, Jumping genes multiobjective optimization scheme for planar monopole ultrawideband antenna, IEEE Transactions On Antenna and Propagation 56 (12) (2008) 3659–3666.
[36] Q. Zhang, H. Li, MOEA/D: a multiobjective evolutionary algorithm based on decomposition, IEEE Transactions on Evolutionary Computation 11 (6) (2007) 712–731.
[37] Q. Zhang, A. Zhou, S. Zhao, P.N. Suganthan, W. Liu, S. Tiwari, Multiobjective Optimization Test Instances for the CEC 2009 Special Session and Competition, University of Essex and Nanyang Technological University, Tech. Rep. CES-487, 2008.
[38] S. Zhao, P.N. Suganthan, Multi-objective evolutionary algorithm with ensemble of external archives, International Journal of Innovative Computing, Information and Control 6 (4) (2010) 1713–1726.
[39] S.-Y. Zheng, S.H. Yeung, W.S. Chan, K.F. Man, K.S. Tang, Design of broadband hybrid coupler with tight coupling using jumping gene evolutionary algorithm, IEEE Transactions On Industrial Electronics 56 (8) (2009) 2987–2991.
[40] E. Zitzler, K. Deb, L. Thiele, Comparison of multiobjective evolutionary algorithms: empirical results, Evolutionary Computation 8 (2000) 173–195.
[41] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: improving the strength pareto evolutionary algorithm for multiobjective optimization. In: K. Giannakoglou, et al. (Eds.), Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001), International Center for Numerical Methods in Engineering (CIMNE), 2002, pp. 95–100.
[42] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach, IEEE Transactions on Evolutionary Computation 3 (4) (1999) 257–271.
[43] E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, V. da Fonseca, Performance assessment of multiobjective optimizers: an analysis and review, IEEE Transactions on Evolutionary Computation 7 (2) (2003) 117–132.