Neural Comput & Applic (2018) 30:1925–1938
https://doi.org/10.1007/s00521-016-2789-3

ORIGINAL ARTICLE

A simple yet effective grouping evolutionary strategy (*GES*) algorithm for scheduling parallel machines

Ali Husseinzadeh Kashan[1] · **Marziehsadat Keshmiry**[2] · **Jalil Heidary Dahooie**[3] · **Amin Abbasi-Pooya**[1]

Received: 12 August 2015 / Accepted: 17 December 2016 / Published online: 27 December 2016

Abstract As a novel evolutionary technique, grouping evolutionary strategy (*GES*) has proved efficient and effective on grouping problems in which the task is to partition a set of items into disjoint groups. This paper investigates the first application of *GES* to tackle the parallel-machines scheduling problem as a well-known grouping problem in which machines can be treated as groups and jobs can be regarded as the items and the task is to partition a set of jobs into disjoint groups (and process all jobs in a same group by the same machines) to minimize makespan ($C_{\max}$) criterion. The main features of *GES* algorithm that make it different from the typical evolutionary approaches proposed for the parallel-machines scheduling problem, lie in exploiting a suitable chromosomal representation and a well-designed mutation operator that works with the set of jobs assigned to each machine instead of jobs isolatedly, and uses a two-phase procedure to generate the new schedules more effectively. In addition, we hybridized *GES* with an efficient local search heuristic and proved that it has an important descent property. To verify the performance of our proposed algorithm, comparisons are made using available methodologies in the literature. Computational results signify that the proposed approach is fast and competitive in providing high quality results.

✉ Ali Husseinzadeh Kashan
A.kashan@modares.ac.ir

[1] Department of Industrial and Systems Engineering, Tarbiat Modares University, Tehran, Iran

[2] Department of Industrial Engineering, Kowsar University, Bojnourd, Iran

[3] Faculty of Management, University of Tehran, Tehran, Iran

Keywords Scheduling · Parallel-machines · Makespan · Grouping evolutionary strategy

1 Introduction

Identical parallel-machines scheduling problem is formally described as follows: a set of n independent jobs, $J = \{J_1, J_2, \ldots, J_n\}$, each having an associated processing time p_i, $i = 1,\ldots,n$, are to be processed on a set $M = \{M_1, M_2, \ldots, M_m\}$ of m machines. Each job should be processed on one of the machines, and preemption is not allowed during processing. The subset of jobs assigned to machine M_j in a schedule is denoted by S_{M_j}. Furthermore, each machine can only process one job at a time, and there is no precedence relation between jobs. The paper investigates the problem of optimal assignment of jobs to machines in order to minimize the completion time of the last job, i.e., the makespan criterion ($C_{\max}$). Due to the fact that this problem is *NP*-hard [13], it is unlikely to obtain the optimal schedule through polynomial time-bounded algorithms. Over the years extensive research has been carried out to develop efficient approaches for the problem. As a member of a family of algorithms known as list-scheduling algorithms, the well-known longest processing time (*LPT*) rule of Graham [15] has received considerable attention because it tends to perform better in terms of performance guarantee. According to LPT rule, we start with an empty schedule and iteratively assign a non-scheduled job with the longest processing time of all remaining jobs to the machine with currently minimal workload. This method generates a schedule that is no worse than $\frac{C_{\max}(LPT)}{C^*_{\max}} \le \frac{4}{3} - \frac{1}{3m}$, where $C_{\max}(LPT)$ denotes the makespan obtained based on the *LPT* rule. Coffman et al. [7] developed an algorithm entitled *MULTIFIT* that establishes the relation between bin-packing and makespan problems. Although the performance guarantee for *MULTIFIT* algorithm is tighter than that of *LPT* algorithm, it does not imply that *MULTIFIT* algorithm will yield better makespan than *LPT* algorithm for any given problem.

Min and Cheng [37] proposed a modified version of genetic algorithm based on machine code for minimizing the makespan on identical machines scheduling problem. Fatemi Ghomi and Jolai Ghazvini [12] developed an algorithm for the problem that is also applicable to schedule nonidentical parallel-machines and also the case of non-simultaneous job arrivals. With the idea that the variance of the last job's completion times on each machine in the presence of job preemption is zero, they tried to minimize sum of ranges of machine finish times instead of the makespan. Gupta and Ruiz-Torres [16] proposed a heuristic named *LISTFIT* based on bin-packing problem and list scheduling that its worst-case performance bound is no worse than that of *MULTIFIT* algorithm. Their computational results demonstrated that the heuristic outperforms the *LPT* algorithm, the *MULTIFIT* algorithm and the *COMBINE* methods of Lee and Massey [33] that utilizes the output of *LPT* algorithm as an initial solution for the *MULTIFIT* algorithm. Mokotoff [38] proposed an exact cutting plane algorithm for minimizing the makespan in parallel-machines scheduling problem. Dell'Amico and Martello [8] provided a note, demonstrating that their proposed exact algorithm in Dell'Amico and Martello [9] outperforms the exact algorithm proposed by Mokotoff [38] in terms of time and quality of solutions for minimizing makespan. Gharbi and Haouari [14] proposed an approximate decomposition algorithm that calls for iteratively solving a sequence of two-machine problems. Iori and Martello [28] reviewed, evaluated and compared scatter search algorithms for two generalizations of identical parallel machine scheduling problem. Mellouli et al. [36] studied the identical parallel-machines scheduling problem with a planned maintenance period on each machine with the objective of minimizing the sum of completion times. They proposed three exact methods to solve the problem at hand: mixed integer–linear programming method, a dynamic programming-based method and a branch-and-bound method. Lee et al. [34] presented a simulated annealing (*SA*) algorithm for the problem and evaluated its performance in comparison to *LISTFIT* and *PI* algorithms. They asserted that their approach outperforms comparator algorithms for all experimental frameworks. Husseinzadeh Kashan et al. [23] introduced a local search heuristic for enhancing the performance of the genetic algorithm for scheduling parallel batch-processing machines. Husseinzadeh Kashan and Karimi [20] proposed a discrete particle swarm optimization algorithm and its hybridized version for the problem of scheduling identical parallel machines and evaluated their performance in comparison to the *SA* algorithm. Jing et al. [29] proposed a discrete harmony search (*HS*) algorithm to solve identical parallel-machines scheduling problem. They embedded a local search in the *HS* algorithm to enhance its performance. Balin [3] proposed a new crossover operator and a new optimality criterion to adapt genetic algorithm to nonidentical machines scheduling problem. A dynamic harmony search (*DHS*) was presented by Chen et al. [6] to minimize makespan in a parallel-machines system. The proposed algorithm was also incorporated with a *VNS*-based local search to improve its effectiveness. Bathrinath et al. [4] proposed a genetic algorithm and a simulated annealing algorithm to solve parallel-machines scheduling problem with the objectives of simultaneous minimization of makespan and number of tardy jobs. The effectiveness of the proposed algorithms was compared by solving benchmark problems. Zarandi and Kayvanfar [42] compared *NSGAII* and *NRGA* in scheduling identical parallel-machines considering controllability of processing times and just-in-time (*JIT*) philosophy, when minimizing total cost and makespan. Hashemian et al. [17] considered the existence of non-availability periods of machines in their proposed model and utilized constructive and backtracking heuristics to solve the model. Bathrinath et al. [5] compared a *VNS*-based heuristic with an *SA*-based heuristic in parallel-machines scheduling problem with weighted objective function of makespan and number of tardy jobs. Kuruvilla and Paletta [32] proposed and evaluated a heuristic algorithm combining *LPT* and *MULTIFIT* heuristics to solve the parallel-machines scheduling problem with the objective of minimizing the makespan. Diana et al. [10] proposed an immune-inspired algorithm incorporating greedy randomized adaptive search (*GRASP*) and variable neighborhood descent (*VND*) algorithms for minimizing the makespan and compared it to three recently proposed genetic algorithm (*GA*), ant colony optimization (*ACO*) algorithm and simulated annealing (*SA*) algorithm for the parallel-machines scheduling problem. Pakzad-Moghaddam [39] presented a mixed integer programming formulation with the objectives of makespan and total hiring cost for the problem and proposed a particle swarm optimization (*PSO*) algorithm, embedded with Lévy flights in replacement for random walks, to solve large-sized problems. Kowalczyk and Leus [31] proposed a branch-and-price algorithm for the problem of parallel-machines scheduling with conflicts by combining bin-packing, scheduling and graph coloring methods. They demonstrated the algorithm's efficiency for problems with and without conflicting jobs. Low and Wu [35] proposed two *ACO* algorithms for the problem where parallel machines are unrelated and each job can be processed on a subset of machines and compared the algorithms on different sizes of the problem.

2 Motivation

Since its introduction in 1994, grouping genetic algorithm (*GGA*) is the most predominant algorithm for grouping problems in which the aim is to group a set of items in disjoint groups [11]. Many *NP*-hard combinatorial optimization problems such as graph coloring problem, bin-packing problem, batch-processing machine scheduling problem, line-balancing problem, timetabling problem, parallel-machines scheduling problem, cell formation problem and pickup and delivery problem are well-used examples of grouping problems.

Grouping evolution strategies (*GES*) which has been introduced in 2009 [24] is one of the latest evolutionary algorithms introduced just for grouping problems. *GES* is the grouping version of the well-known evolution strategies which have been heavily modified to suit the structure of grouping problems. *GES* has been successfully applied to bin-packing problem [24, 26], batch-processing machine scheduling problem [26] and fuzzy data clustering [25]. Results showed that on bin-packing problem and batch-processing machine scheduling problem, *GGA* is inferior to *GES*. Moreover, the design of *GES* gives it some advantages over *GGA*.

Given the fact that parallel-machines scheduling problem is a grouping problem, the aim of this paper is to propose an algorithm which takes into account the structural knowledge along with the grouping nature of this problem. The aim of the paper is to propose a *GES*-based algorithm for the parallel-machines scheduling problem. We believe that it is not straightforward to propose a *GGA* with its classic operators for this problem, because:

- When the structure of the grouping problem is in such a way that the number of groups is very small, the use of *GGA* to form the groups may be irrelevant. Short chromosome lengths imposed by the very small number of groups make the *GGA* operators less applicable. For example, in the parallel-machines scheduling problem with two machines, every solution (i.e., grouping of jobs) has inevitably two groups. Here, the *GGA* crossover and mutation operators as those described in [11] are completely inapplicable because there are only three positions for crossing sections (given that the operators work with groups). The performance of *GGA* is worsened due to the fact that the chromosomes become proportionally shorter in length, thus placing possible limitations on the search capabilities of the *GGA* operators. Hopefully, *GES* does not suffer from this deficiency.
- While we can simply apply *GES* on grouping problems with nonidentical groups, this is not true for *GGA*. Let us consider the parallel-machines scheduling problem with nonidentical machines and the objective of minimizing makespan. Since the classic *GGA* performs under whole-group sharing rationale, it cannot be applied to this scheduling problem. However, under subgroup sharing rationale which is followed by *GES*, there is no matter whether groups are identical or nonidentical [26] .

The remainder of the paper is organized as follows. In the following section, we give a brief introduction to evolution strategies (*ES*) and its source of inspiration. The grouping version of evolution strategies (*GES*) for parallel-machines scheduling problems is presented in Sect. 4. Section 5 generalizes the proposed methodology for job scheduling on nonidentical machines. Section 6 investigates the effectiveness of *GES* through computational experiments. The paper will be concluded in Sect. 7.

3 An introduction to evolution strategies (*ES*)

According to Darwin's theory about the development of species, the most important characteristics of the evolution process are inheritance, mutation and selection. Only these properties of the biological evolution had to be translated into mathematical terms to develop a most general and effective optimization technique. Evolution strategies (*ES*) of Rechenberg [40] are methods which translate the properties of the Darwinian biological evolution into mathematical terms to formulate a general optimization method.

The family of evolution strategies is introduced by $(\mu/\rho \overset{+}{,} \lambda) - ES$ notation. All members operate with a population Π^t of μ candidate solutions. In every time step t, a set Q^t of μ candidate solutions is generated from Π^t by employing the recombination and mutation operators. The symbol ρ indicates the number of parental solutions involved in the generation of every single offspring solution. When $\rho = 1$, it will be omitted. To form Π^{t+1}, the candidate solutions are selected on the basis of their fitness. *Selection* which is the goal-directed element of the evolutionary search is represented by "$\overset{+}{,}$", denoting the two mutually exclusive selection types. Employing "+" selection, the μ best of $\mu + \lambda$ candidates in $\Pi^t \cup Q^t$ are selected to form Π^{t+1}. Using "," selection, it is the μ best of λ candidate solutions in Q^t that form Π^{t+1}.

Generation of an offspring candidate solution is supported via mutation. *Mutation* consists of adding to centroid, a random noise drawn from an isotropic normal distribution. For a population $\Pi^t = \left\{X_1^t, X_2^t, \ldots, X_\mu^t\right\}$ in which $X_k^t = \left(x_{k1}^t, x_{k2}^t, \ldots, x_{\mathrm{km}}^t\right) \forall k = 1, \ldots, \mu$, is a m-dimensional solution vector in a continuous search space, the set Q^t will be composed of offspring candidate solutions $Y_i^t = \left(y_{i1}^t, y_{i2}^t, \ldots, y_{im}^t\right) \forall i = 1, \ldots, \lambda$, where:

$$y_{ij}^t = x_{i_k j}^t + z_j^t,\ \forall j = 1, \ldots, m,\ \forall i = 1, \ldots, \lambda \tag{1}$$

with $z_j^t = \sigma^t N_j(0,1)$, where $N_j(0,1)$ is a random number associated to dimension j generated from a standardized normal distribution (i.e., the variation source). The positive scalar variable σ^t, which is learnt during the evolution process is referred to as the strategy parameter or mutation strength and determines the expected distance (variation) of an offspring candidate solution from the centroid of its parents. Index i_k is drawn with replacement from the set $\{1, \ldots, \mu\}$.

Too low values for mutation strength can slow down the progress while too high values may lead to divergence. Therefore, the mutation strength σ^t should be adapted in course of search. The first mutation strength adaptation strategy was proposed for $(1+1)-ES$ by Rechenberg [41]. Rechenberg's recommendation was to monitor success probabilities (i.e., the probability that an offspring candidate solution is superior to its parent) by averaging over the number of time steps. The mutation strength is increased if the observed estimate of the success probability is >0.2 and it is decreased if the success probability is <0.2. This mutation strength adaptation scheme is known as 1/5-success rule.

4 Grouping evolution strategies (*GES*) for parallel-machines scheduling problems

In this section our aim is to propose an efficient and effective algorithm for parallel-machines scheduling problem based on *GES*. The algorithm employs an encoding scheme which is called *assignments encoding*. A particular mutation operator is used which works based on the composition of job assignments to machines and not the jobs isolatedly. The mutation strategy is implemented via a two-phase heuristic.

4.1 The structure of assignments encoding

When optimizing a continuous function by *ES*, each solution is represented by a chromosome of length m of real numbers (m is the problem dimension, i.e., the number of variables). Similarly, for parallel-machines scheduling problem, one can represent a solution as a structure whose length is equal to the number of machines. The content of each element in the structure demonstrate all jobs assigned to the relevant machine (see the left part in Fig. 1). Figure 1 demonstrates a schedule for a five jobs and two machines problem. Any feasible schedule must inevitably partition the set of jobs into two groups. In the exampled schedule, jobs J_1, J_2 and J_5 are assigned to machine M_1 and form the first group. Jobs J_3, J_4 are assigned to machine M_2 which forms the second group. Thus, machines play the role of groups. Adopting the structure of assignments encoding, *GES* works with the sets $\{J_1, J_2, J_5\}$, $\{J_3, J_4\}$ as a chromosomal structure with only two genes (one gene for each group). Hence, the *GES* operations are designed based on the set/groups of jobs rather than jobs isolatedly. The rationale is that in a grouping problem like parallel-machines scheduling problem, these are the job groups that are the innate building blocks of the problem, which can convey information on the expected quality of the schedule they are part of, and not the particular positions of any one job on its own.

4.2 The *GES* mutation operator

Given the structure of assignments encoding, the aim of this section is to adapt mutation (1) to obtain the one which works with the whole jobs assigned to a machine instead of scalars. Reconstructing (1) to work with whole jobs assigned to a machine, the major idea would be to use suitable operators as a substitute for arithmetic operators. In particular, we substitute "–" operator with a dissimilarity measure. Similar to "–" operator that measures the magnitude of difference between two scalars, a dissimilarity measure quantifies the distance/dissimilarity between two pattern of job assignments. We use "*Distance*" to address such a measure.

Let A and A' be two different subset of jobs assigned to a given machine (say M_1) in two different solutions of the problem, respectively. Let $|A|$ denotes the cardinality of A. Measuring the degree of similarity between A and A' helps us to determine how similar the assignments are to each other or how far apart they are from each other. One of the most commonly used measures to determine the degree of dissimilarity between A and A' is the Jaccard's coefficient of dissimilarity defined as follows:

$$\begin{array}{ccccccccc} \text{Machine } M_1 & \text{Machine } M_2 & & J_1 & J_2 & J_3 & J_4 & J_5 \\ \{\{J_1, J_2, J_5\}, & \{J_3, J_4\}\} & \equiv & M_1 & M_1 & M_2 & M_2 & M_1 \end{array}$$

Fig. 1 Structure of assignments encoding

$$Dis(A, A') = 1 - \frac{|A \cap A'|}{|A \cup A'|} \tag{2}$$

It is obvious that $0 \leq Dis(A, A') \leq 1$. To develop an analogous Gaussian mutation let us reshape (1) in form of $y_{ij}^t - x_{i_k j}^t = z_j^t$ and substitute "–" by "*Dis*." The Gaussian mutation can be introduced as follows:

$$Dis(y_{ij}^t, x_{i_k j}^t) \approx z_j^t \tag{3}$$

where $j = 1, \ldots, m$, $i = 1, \ldots, \lambda$ and $z_j^t = \sigma^t N_j(0, 1)$. While in (1), $x_{i_k j}^t$ and y_{ij}^t are scalars, in (3) $x_{i_k j}^t$ and y_{ij}^t denote the subset of jobs assigned to a machine M_j in schedules $X_{i_k}^t$ and Y_i^t, respectively.

In (4), "$\approx$" implies "approximately equal to." We use this symbol instead of "=" because it may not be possible to form the new group of jobs (y_{ij}^t) assigned to machine M_j in the offspring schedule Y_i^t in such a way that the value of $Dis(y_{ij}^t, x_{i_k j}^t)$ becomes exactly equal to z_j^t.

In place of Jaccard coefficient of dissimilarity, it may be used the Kulczynski's coefficient or Sørensen–Dices' coefficient. However, presumably the most widely used dissimilarity measure is the Jaccard's coefficient of dissimilarity. Moreover, both of these measures can perform equivalent to the Jaccard's coefficient of dissimilarity in our case.

The component z_j^t makes use of variations depending on the selected mutation strength σ^t. If σ^t increases, there is more likelihood for z_j^t to go far away beyond the origin, and if σ^t decreases, there is more likelihood for z_j^t to fall around the origin. While z_j^t can get any arbitrary real value unrestricted in sign (i.e., $z_j^t \in (-\infty, \infty)$), the range of $Dis(.,.)$ is only real values in [0,1]. This is evidence that z_j^t may not be an appropriate source of variation in *GES*.

Hence, we should devise a different type of random variable as an alternative source of variations. It is highly desirable that the candidate random variable take values just in [0,1]. Moreover, similar to the normal *PDF* in which changing the value of scale parameter σ^t changes the chance of getting a random normal value in a specific range, it is of interest to devise a flexible *PDF* that provides different chances for getting a value in a specific sub-range in [0,1] by means of the *PDF* input parameter(s). The Beta distribution which is denoted by $B(\alpha, \beta)$ is an intended *PDF* which is defined in [0,1] and models skew quite well. We are therefore at the point to use Beta distribution in place of isotropic normal distribution typically used in ES, as the source of variation in *GES*. Beta distribution is flexible in shape and takes all forms of *J* shape, humped shape and *U* shape. But we need only *J*-shaped and humped-shaped Beta distributions. We can achieve these shapes via considering the value of one of the shape parameters, say β, greater than one. From (3), we finally obtain the mutation relation in *GES* as follows:

$$Dis(y_{ij}^t, x_{i_k j}^t) \approx Beta_j(\alpha^t, \beta),\ \forall j = 1, \ldots, m, \forall i = 1, \ldots, \lambda,\quad i_k \in \{1, \ldots, \mu\} \tag{4}$$

where $Beta_j(\alpha^t, \beta)$ is a Beta random number associated to machine M_j with shape parameters α^t and β. Keeping the value of β constant, we can only consider α^t as the endogenous strategy parameter just similar to the classic *ES*.

4.3 Generating a new schedule

Generation of the new schedule in *GES* requires two phases. The first phase, the *inheritance phase*, includes deciding about those parts of the parent that the offspring inherits. During this phase a number of jobs may remain unassigned. Therefore, the second phase is the *post-assignment* or *reinsertion phase* in which the unassigned jobs are reassigned to machines.

The inheritance phase is handled through (4). By (4) it is implied that the construction of the new assignment of jobs to machine M_j (i.e., y_{ij}^t) in the offspring schedule during the inheritance phase should be in such a way that its degree of dissimilarity with its counterpart in the parent schedule (i.e., $x_{i_k j}^t$) be around the value $Beta_j(\alpha^t, \beta)$. This means that the degree of similarity between two assignments should be approximately equal to $1 - Beta_j(\alpha^t, \beta)$.. In other words, we seek the number of jobs shared between $x_{i_k j}^t$ and y_{ij}^t (i.e.,$n_{id}^t = |y_{ij}^t \cap x_{i_k j}^t|$) in such a way that the value of $Dis(y_{ij}^t, x_{i_k j}^t)$ gets close to the value of $Beta_j(\alpha^t, \beta)$. Indeed, the shared jobs between $x_{i_k j}^t$ and y_{ij}^t are one of the parts that offspring schedule Y_i^t inherits from parent schedule $X_{i_k}^t$. During the inheritance phase, it is reasonable to assume that $y_{ij}^t \subseteq x_{i_k j}^t$ (because, y_{ij}^t can inherit up to all jobs of $x_{i_k j}^t$). Starting from (4) we have:

$$Dis(y_{ij}^t, x_{i_k j}^t) = 1 - \frac{n_{ij}^t}{|x_{i_k j}^t|} \approx Beta_j(\alpha^t, \beta) \Rightarrow n_{ij}^t = \left\lfloor (1 - Beta_j(\alpha^t, \beta))|x_{i_k j}^t| \right\rfloor \tag{5}$$

The following algorithm describes the steps of generating the offspring schedule Y_i^t based on the parent schedule $X_{i_k}^t$.

New schedule generator (*NSG*) algorithm

```
Step 1 (the inheritance phase)
For j=1 to m
    Let $x^t_{i_k j}$ be the set of jobs assigned to machine $M_j$,
    in the parent schedule $X^t_{i_k}$ in iteration t;
    $n^t_{ij} \leftarrow \lfloor (1-Beta_j(\alpha^t,\beta)) \, | x^t_{i_k j} | \rfloor$;
    $y^t_{ij} \leftarrow \emptyset$;
    Select $n^t_{ij}$ number of jobs from $x^t_{i_k j}$ and assign them to subset $y^t_{ij}$;
End for
Step 2 (the post-assignment phase)
Allocate a machine to each of the remaining jobs that
have not been selected during Step 1 to update
the relevant $y^t_{ij}$s to obtain the complete schedule $Y^t_i$;
```

Problem-dependent heuristics can be employed at both steps of *NSG* algorithm. We select jobs in Step 1 of *NSG* algorithm in a greedy fashion based on their attribute values (e.g., processing times). Selecting n^t_{ij} number of jobs with the longest processing times helps us to use the structural knowledge of the problem to generate possibly better schedules. Such a selection strategy is inspired from *LPT* rule. However, we apply such a selection strategy in a probabilistic manner. That is, everytime for each machine M_j a random number is drawn from [0, 1]. If it becomes less than 0.1, selection is done based on job processing times. Otherwise, job selection is done randomly.

At step 2 of *NSG* algorithm any constructive heuristic can be employed. Again we use *LPT* rule to iteratively assign a remaining job with the longest processing time of all remaining jobs, which has not been selected during Step 1, to the machine with currently minimal workload.

The selection process in *GES* is deterministic and similar to what was explained in Sect. 3. Similar to the notation employed to introduce the multi-member *ES*, we can use $(\mu \overset{+}{,} \lambda) - GES$ notation to introduce the family of *GES* algorithms. In a manner similar to "1/5-success rule," starting from α^0, we increase α^t if the observed estimate of the success probability exceeds a given threshold (P_s) during G successive iterations and decrease it if the success probability gets below the threshold.

To improve the performance of *GES*, we hybridize it with a pairwise interchange algorithm (*PIA*). For a given schedule, *PIA* tries to reduce the makespan through suitable pairwise interchange of jobs between machines. The structure of *PIA* is in such a way that it either terminates without any improvement gained over the input schedule or it will decrease the makespan value of the input schedule via job interchanges. In this way it performs as a descent algorithm.

Pairwise Interchange Algorithm (*PIA*)

```
Given an input schedule, level the number of jobs assigned
to each machine by adding dummy jobs of
zero processing times.
→ Reindex machines based on decreasing order of their
own load time (i.e., $C_{max_j} = \sum_{J_r \in S_{M_j}} p_r$);

If all machines have the same load
    Terminate (the optimal schedule has been obtained);
Else
    $i \leftarrow 1$;
    $j \leftarrow m$;
    $k \leftarrow 0$;
End If
⇒ Find a job $J_a \in S_{M_i}$ and a job (or dummy job) $J_b \in S_{M_j}$
such that: $0 < p_a - p_b < C_{\max_i} - C_{\max_j}$;
If there is no job $J_a$ and $J_b$
    $j \leftarrow j-1$;
    $k \leftarrow k+1$;
    If $k = m - i$
        $i \leftarrow i+1$;
        $j \leftarrow m$;
        $k \leftarrow 0$;
        If $i = m$
            Terminate (no more improvement is possible);
        Else
            Go to ⇒;
        End If
    Else
        Go to ⇒;
    End If
Else
    Interchange job $J_a$ and $J_b$ between $M_i$ and $M_j$, and Go to →;
End If
```

In our implementation, we use $(1+1) - GES$ family which is elaborated as follows. Since $\mu = 1$, we omit index i_k.

(1+1)-*GES* Algorithm

```
t ← 0;
G_s ← 0;
α ← α^0;
Create an initial random schedule X^t;
While stopping criteria are not true
| | If $C_{\max}(X^t) = \max\left\{\sum_{i=1}^{n} p_i/n, \max_{i=1,\ldots,n}\{p_i\}\right\}$
| | | | Terminate (the optimal schedule has been obtained);
| | End if
| | Given the parent schedule $X^t$, apply NSG algorithm to obtain the offspring schedule $Y^t$;
| | If Rand(0,1)<R
| | | | Apply the pairwise interchange algorithm (PIA) given $Y^t$ as its input schedule to obtain the
        enhanced offspring schedule $E^t$;
| | | | $Y^t \leftarrow E^t$;
| | End
| | If $C_{\max}(Y^t) < C_{\max}(X^t)$
| | | | $X^{t+1} \leftarrow Y^t$;
| | | | $G_s \leftarrow G_s + 1$;
| | Else
| | | | $X^{t+1} \leftarrow X^t$;
| | End if
| | If $(t \mod G) = 0$
| | | | $\alpha \leftarrow \begin{cases} \alpha/a & \text{if } G_s/G \ge P_s \\ a\times\alpha & \text{if } G_s/G < P_s \end{cases}$
| | | | $G_s \leftarrow 0$;
| | End if
| | $t \leftarrow t+1$;
| | $\alpha^t \leftarrow \alpha$;
End while
```

5 Generalization to the case of nonidentical machines

In the nonidentical parallel-machines scheduling problem there are a set of n independent jobs $J = \{J_1, J_2, \ldots, J_n\}$, each of which has to be scheduled on one of m machines $M_1, M_2, \ldots, M_m$. A job can run on only one machine at a time, and a machine can process at most one job at a time. If a job J_i is processed on a machine M_j, it will take a positive processing time p_{ij}. When $p_{ij} = p_i$ for all i and j, the machines have the same speed and the machines are called identical. When $p_{ij} = p_i/s_j$ where p_i is the processing time of job J_i and s_j is the speed of machine M_j, then the machines are called uniform. If the p_{ij}'s are arbitrary then the machines are called unrelated. Both of the uniform and unrelated cases belong to nonidentical parallel-machines scheduling problem.

The structure of *GES* allows it to simply work for scheduling jobs on nonidentical parallel-machines. We need only

- adapt the *LPT* rule properly for the case of nonidentical machines, just to use it in the body of *NSG* algorithm. For example, for the case of uniform machines the following two rules can be referred to as the *LPT* rule. In the case of identical processors these methods give the same schedule.
    1. *As a machine finishes a job it chooses from the queue of waiting jobs the one with the longest processing time.*
    2. *Assign jobs, in the order of non-increasing processing times, to the machine on which they will have the earliest finishing time.*
- adapt the pairwise interchange algorithm to work for the case of scheduling jobs on nonidentical machines. For this purpose we should only modify the job selection step of *PIA* which says "*Find a job* $J_a \in S_{M_i}$ *and a job* (*or dummy job*) $J_b \in S_{M_j}$ *such that* $0 < p_b - p_b < C_{\max_i} - C_{\max_j}$." The following lemmas help us to determine suitable conditions for job selection.

Lemma 1 *Suppose p_{aj} and p_{bj} be the processing time of job J_a and J_b on machine M_j, respectively; let also $C_{\max_j} = \sum_{J_r \in S_{M_j}} p_r$, be the load time of machine M_j. If the conditions $0 < p_{ai} - p_{bi} < C_{\max_i} - C_{\max_j}$ and $0 < p_{aj} - p_{bj} < C_{\max_i} - C_{\max_j}$ are held for any jobs $J_a \in S_{M_i}$ and $J_b \in S_{M_j}$, the following inequalities hold after interchanging job J_a and J_b between M_i and M_j ($C'_{\max_j}$ is the load time of machine M_j after interchange.*

1. $\left|C'_{\max_i} - C'_{\max_j}\right| \le C_{\max_i} - C_{\max_j}$

2. $\max(C'_{\max_i}, C'_{\max_j}) < C_{\max_i}$

Proof By interchanging job J_a and J_b between M_i and M_j we will have $C'_{\max_i} = C_{\max_i} - p_{ai} + p_{bi}$ and $C'_{\max_j} = C_{\max_j} + p_{aj} - p_{bj}$. As a result we have $C'_{\max_i} - C'_{\max_j} = C_{\max_i} - C_{\max_j} - (p_{ai} - p_{bi}) - (p_{aj} - p_{bj})$. Following the assumptions we have $-(p_{ai} - p_{bi}) - (p_{aj} - p_{bj}) > -2(C_{\max_i} - C_{\max_j})$. We therefore arrive at $C'_{\max_i} - C'_{\max_j} > -(C_{\max_i} - C_{\max_j})$. On the other hand, from our assumptions we have $-(p_{ai} - p_{bi}) - (p_{aj} - p_{bj}) < 0$ which forces that $C'_{\max_i} - C'_{\max_j} < (C_{\max_i} - C_{\max_j})$. We conclude that $\left|C'_{\max_i} - C'_{\max_j}\right|$

$\leq C_{\max_i} - C_{\max_j}$. To prove the second inequality, we have $C'_{\max_i} = C_{\max_i} - (p_{ai} - p_{bi}) < C_{\max_i}$ and $C'_{\max_j} = C_{\max_j} + (p_{aj} - p_{bj}) < C_{\max_j} + C_{\max_i} - C_{\max_j} = C_{\max_i}$. We finally get $\max(C'_{\max_i}, C'_{\max_j}) < C_{\max_i}$.

Remark 1 When doing a job interchange during execution of *PIA*, if we have $J_a \in S_{M_i}$, then the value of makespan will be decreased.

Remark 2 When the scheduling criterion is the minimization of total flow time, any job interchange during execution of *PIA* will decrease the value of total flow time.

To adapt the pairwise interchange algorithm to work for the problem of scheduling jobs on nonidentical machines, we only modify the job selection step of *PIA* as follows: "*Find a job* $J_a \in S_{M_i}$ *and a job (or dummy job)* $J_b \in S_{M_j}$ *such that:* $0 < p_{ai} - p_{bi} < C_{\max_i} - C_{\max_j}$ *and* $0 < p_{aj} - p_{bj} < C_{\max_i} - C_{\max_j}$."

6 Computational experiments

In this section, we investigate a comparison study on the effectiveness of the proposed *GES* algorithm on a large number of test problem instances. We benchmark the experimental frameworks used in Lee et al. [34] that have originally been designed by Gupta and Ruiz-Torres [16], Lee and Massey [33] and Kedia [30]. Four experimental frameworks, namely E_1, E_2, E_3 and E_4 are considered each of them having three influencing variables: the number of machines (m); the number of jobs (n); and the type of discrete uniform distribution used to generate job processing times (p).

Table 1 presents a summary of the experimental framework which is composed of 120 problems. Given the problem parameters of Table 1, 50 test instances are generated for each problem. This yields 6000 test problem instances in total.

To implement our *GES* and its hybridized version, *GES* + *PIA*, the following set of parameters are used. The maximum number of iterations is 1000. Only 1000 schedules are generated and evaluated. G is 6, P_s is 6^{-1}, a is 0.98, β is 5 and α^0 is 10. Every generated offspring in *GES* + *PIA* is selected to be improved by *PIA* with probability of 0.1. We found these values suitable through preliminary experiments. We have done the value selection process in a greedy fashion, though *GES* is relaxed to the exact tuning of parameters.

Tables 2, 3, 4, 5, 6 and 7 report the results acquired from the computations. The column with the title "Mean" reports the mean performance obtained by the corresponding algorithm. The ratio of the makespan over the lower bound is averaged for 50 replications to obtain the mean performance of an algorithm for an experiment. It should be mentioned that the mean performance values are given by four digits precision without rounding. The column with the title "Time (s)" reports the average running time (in seconds) to solve each of 50 test instances of a problem. The last three columns demonstrate the relative performance of the algorithms. Each cell in these three columns indicate the number of times, out of 50, a given algorithm reports a better makespan compared to the other algorithm. For instance, a value of *c versus d* in *GES versus SA* column means that, out of 50 replications, there are c test instances for which *GES* performs better than *SA*, d problems for which *SA* finds a better schedule, and 50-c-d test instances for which *GES* and *SA* yield the same makespan.

We have implemented both of *GES* and *GES* + *PIA* algorithms in MATLAB and run them on a laptop computer with 2.67 GHz of CPU speed and 4 GB of RAM. The results of *GES* and *GES* + *PIA* are compared with the *SA* algorithm of Lee et al. [34].

The results for the experiment E_1 are summarized in Table 2. In terms of the mean performance there is a negligible difference between *GES* and *GES* + *PIA* algorithms. Only on two problems (e.g., $m = 4$, $n = 8$, $p \sim U(20,50)$ and $m = 5$, $n = 15$, $p \sim U(20,50)$) *GES* + *PIA* performs slightly better than *GES*. *SA*'s mean performance is inferior to *GES* and *GES* + *PIA* algorithms on seven problems, but superior on three problems. As the number of jobs increases, at a fixed level of m, the mean performance of all algorithms improves. Here, *SA* performs faster than others on average.

Table 1 Summary of the computational experiments

Experiments	*m*	*n*	*P*
E_1	3, 4, 5	2 m, 3 m, 5 m	U(1,20), U(20,50)
E_2	2, 3, 4, 6, 8, 10	10, 30, 50, 100	U(100,800)
E_3	3, 5, 8, 10	3 m + 1, 3 m + 2, 4 m + 1, 4 m + 2, 5 m + 1, 5 m + 2	U(1,100), U(100,200), U(100,800)
E_4	2	9	U(1,20), U(20,50), U(50,100), U(100,200), U(100,800)
	3	10	

The bold rows are the average values over all rows

Table 2 Results for experiment E_1

m	*n*	*p*	*SA*		*GES*		*GES* + *PIA*		*GES* versus *SA*	*GES* + *PIA* versus *SA*	*GES* + *PIA* versus *GES*
			Mean	Time (s)	Mean	Time (s)	Mean	Time (s)			
3	6	U(1,20)	1.0592	0.0653	1.0592	0.3032	1.0592	0.3120	0 versus 0	0 versus 0	0 versus 0
	9		1.0089	0.0546	1.0089	0.1560	1.0089	0.1482	0 versus 0	0 versus 0	0 versus 0
	15		1.0000	0.0187	1.0000	0.0093	1.0000	0.0043	0 versus 0	0 versus 0	0 versus 0
	6	U(20,50)	1.0567	0.0771	1.0577	0.5312	1.0577	0.3171	0 versus 1	0 versus 1	0 versus 0
	9		1.0084	0.0906	1.0084	0.3260	1.0084	0.3232	0 versus 0	0 versus 0	0 versus 0
	15		1.0007	0.1003	1.0001	0.0280	1.0001	0.0240	5 versus 0	5 versus 0	0 versus 0
4	8	U(1,20)	1.0687	0.0868	1.0687	0.4330	1.0687	0.4099	0 versus 0	0 versus 0	0 versus 0
	12		1.0036	0.0487	1.0022	0.0502	1.0022	0.0349	2 versus 0	2 versus 0	0 versus 0
	20		1.0000	0.0690	1.0000	0.0081	1.0000	0.0062	0 versus 0	0 versus 0	0 versus 0
	8	U(20,50)	1.0597	0.1047	1.0621	0.5403	1.0601	0.3989	0 versus 2	0 versus 1	1 versus 0
	12		1.0110	0.1834	1.0088	0.3943	1.0088	0.3890	12 versus 0	12 versus 0	0 versus 0
	20		1.0010	0.2071	1.0000	0.0402	1.0000	0.0109	9 versus 0	9 versus 0	0 versus 0
5	10	U(1,20)	1.0587	0.1015	1.0587	0.4517	1.0587	0.4885	0 versus 0	0 versus 0	0 versus 0
	15		1.0101	0.1109	1.0079	0.1279	1.0079	0.1244	4 versus 0	4 versus 0	0 versus 0
	25		1.0000	0.1094	1.0000	0.0093	1.0000	0.0074	0 versus 0	0 versus 0	0 versus 0
	10	U(20,50)	1.0589	0.1337	1.0602	0.7909	1.0602	0.5881	0 versus 1	0 versus 1	0 versus 0
	15		1.0091	0.2528	1.0071	0.3756	1.0069	0.3818	8 versus 0	9 versus 0	1 versus 0
	25		1.0011	0.3056	1.0000	0.0852	1.0000	0.0119	10 versus 0	10 versus 0	0 versus 0
Average			**1.0231**	**0.1177**	**1.0227**	**0.2589**	**1.0226**	**0.2211**	**2.77 versus 0.22**	**2.83 versus 0.16**	**0.11 versus 0.00**

The bold rows are the average values over all rows

Table 3 Results for experiment E_2

m	*n*	*SA*		*GES*		*GES* + *PIA*		*GES* versus *SA*	*GES* + *PIA* versus *SA*	*GES* + *PIA* versus *GES*
		Mean	Time (s)	Mean	Time (s)	Mean	Time (s)			
2	10	1.0009	0.1568	1.0009	0.3123	1.0008	0.3129	3 versus 4	3 versus 2	4 versus 2
3		1.0071	0.2197	1.0068	0.5684	1.0063	0.4042	8 versus 6	11 versus 0	8 versus 0
2	30	1.0000	0.2115	1.0000	0.0636	1.0000	0.0149	0 versus 0	0 versus 0	0 versus 0
3		1.0002	1.5393	1.0001	0.5035	1.0000	0.0711	23 versus 9	38 versus 0	27 versus 0
4		1.0009	2.3543	1.0004	0.6664	1.0000	0.2414	35 versus 6	47 versus 0	42 versus 0
6		1.0027	2.1303	1.0018	0.8096	1.0004	1.0352	29 versus 13	49 versus 0	47 versus 0
8		1.0058	2.4909	1.0041	0.9731	1.0013	1.4741	33 versus 14	49 versus 0	48 versus 1
10		1.0109	2.0803	1.0075	1.1113	1.0034	1.7572	36 versus 9	48 versus 0	46 versus 0
2	50	1.0000	0.2528	1.0000	0.0312	1.0000	0.0152	0 versus 0	0 versus 0	0 versus 0
3		1.0001	4.1131	1.0000	0.4564	1.0000	0.0268	26 versus 4	37 versus 0	17 versus 0
4		1.0004	6.8368	1.0002	0.7147	1.0000	0.0558	25 versus 9	43 versus 0	44 versus 0
6		1.0014	7.8922	1.0008	1.0654	1.0000	0.4158	28 versus 14	48 versus 0	49 versus 0
8		1.0027	8.9484	1.0018	1.3060	1.0001	0.9871	34 versus 15	50 versus 0	50 versus 0
10		1.0031	7.9190	1.0029	1.1943	1.0004	2.0985	18 versus 21	49 versus 0	50 versus 0
2	100	1.0000	0.8615	1.0000	0.0315	1.0000	0.0196	0 versus 0	0 versus 0	0 versus 0
3		1.0000	12.1987	1.0000	0.3603	1.0000	0.0227	5 versus 2	9 versus 0	7 versus 0
4		1.0000	16.3625	1.0000	0.9132	1.0000	0.0302	4 versus 21	23 versus 0	37 versus 0
6		1.0002	32.1006	1.0001	0.9906	1.0000	0.0648	26 versus 12	46 versus 0	41 versus 0
8		1.0005	38.6393	1.0003	1.1696	1.0000	0.1531	33 versus 11	48 versus 0	46 versus 0
10		1.0006	40.0168	1.0004	1.3534	1.0000	0.4399	24 versus 10	46 versus 0	45 versus 0
Average		**1.0018**	**9.3662**	**1.0014**	**0.7297**	**1.0006**	**0.4820**	**19.50 versus 9.00**	**32.20 versus 0.10**	**30.40 vs 0.15**

The bold rows are the average values over all rows

Table 4 Results for experiment E_3: $p \sim U(1,100)$

m	*n*	*SA*		*GES*		*GES* + *PIA*		*GES* versus *SA*	*GES* + *PIA* versus *SA*	*GES* + *PIA* versus *GES*
		Mean	Time (s)	Mean	Time (s)	Mean	Time (s)			
3	10	1.0115	0.1481	1.0110	0.4327	1.0107	0.3789	3 versus 1	5 versus 0	3 versus 0
	11	1.0045	0.1450	1.0038	0.3020	1.0037	0.2973	5 versus 0	6 versus 0	1 versus 0
	13	1.0019	0.1503	1.0008	0.1687	1.0005	0.4394	12 versus 1	15 versus 1	3 versus 0
	14	1.0009	0.1543	1.0001	0.0698	1.0000	0.0393	9 versus 0	10 versus 0	1 versus 0
	16	1.0004	0.1662	1.0000	0.0430	1.0000	0.0177	6 versus 0	6 versus 0	0 versus 0
	17	1.0005	0.1481	1.0000	0.0174	1.0000	0.0137	8 versus 0	8 versus 0	0 versus 0
5	16	1.0096	0.3937	1.0043	0.4449	1.0042	0.4308	28 versus 0	28 versus 0	2 versus 1
	17	1.0086	0.5000	1.0038	0.4505	1.0026	0.3616	26 versus 0	34 versus 0	11 versus 1
	21	1.0031	0.5228	1.0004	0.2430	1.0000	0.0627	26 versus 1	29 versus 0	5 versus 0
	22	1.0030	0.5059	1.0005	0.2489	1.0000	0.0624	25 versus 1	28 versus 0	7 versus 0
	26	1.0020	0.6053	1.0003	0.2361	1.0000	0.0499	22 versus 1	25 versus 0	4 versus 0
	27	1.0010	0.5718	1.0001	0.1675	1.0000	0.0240	11 versus 0	13 versus 0	2 versus 0
8	25	1.0127	1.0012	1.0067	0.8286	1.0041	0.5041	32 versus 0	40 versus 0	19 versus 0
	26	1.0095	0.9859	1.0039	0.7940	1.0014	0.4742	31 versus 5	39 versus 0	19 versus 1
	33	1.0030	1.2787	1.0014	0.6642	1.0000	0.1123	19 versus 3	30 versus 0	15 versus 0
	34	1.0033	1.4606	1.0015	0.6739	1.0000	0.1341	21 versus 3	32 versus 0	16 versus 0
	41	1.0021	1.6290	1.0009	0.6080	1.0000	0.0661	14 versus 1	24 versus 0	12 versus 0
	42	1.0018	1.6400	1.0012	0.5793	1.0000	0.0870	8 versus 2	22 versus 0	17 versus 0
10	31	1.0123	1.4212	1.0076	1.1044	1.0045	0.6870	25 versus 1	37 versus 0	23 versus 0
	32	1.0094	1.3821	1.0038	0.9977	1.0006	0.4380	29 versus 0	42 versus 0	25 versus 0
	41	1.0039	2.1531	1.0019	0.9590	1.0002	0.2134	20 versus 3	30 versus 0	18 versus 0
	42	1.0045	2.5850	1.0022	0.9487	1.0002	0.2570	20 versus 1	36 versus 0	22 versus 0
	51	1.0030	3.1415	1.0021	1.0311	1.0000	0.0836	13 versus 4	35 versus 0	27 versus 0
	52	1.0026	3.0818	1.0013	0.9569	1.0000	0.1160	16 versus 2	30 versus 0	17 versus 0
Average		**1.0048**	**1.0738**	**1.0024**	**0.5404**	**1.0013**	**0.2229**	**17.87 versus 1.25**	**25.16 versus 0.04**	**11.20 versus 0.12**

The bold rows are the average values over all rows

Table 3 reports the results on experimental framework E_2. Here the superiority of *GES* + *PIA* over *SA* and *GES* algorithms is more pronounced in terms of the mean performance. Comparing *SA* with *GES* algorithm, there are many problems on which *GES* performs better than *SA*. In terms of the mean performance, there is no problem in which *SA* could dominate either *GES* or *GES* + *PIA*. However, there are some problems for which all algorithms perform the same. Among 20 problems in this experiment, *SA* finds the optimal makespan of all 50 test instances of five problems. While *GES* performs optimal on six problems, *GES* + *PIA* performs optimal on 13 problems. However, on the optimality of performance on the rest of problems we cannot make any judgment since the lower bound values may not be optimal and the optimal solutions are also unknown. One can find that as the number of jobs and the number of machines increase, *GES* + *PIA* can report the optimal makespan for all of test problems. Results indicate that *GES* + *PIA* performs better than *SA* on average on 32.20 out of 50 test instances of each of 20 problems. While this record is only 0.1 out of 50 for *SA* over *GES* + *PIA*. Such a trend is also visible for *GES* + *PIA* versus *GES*. *GES* + *PIA* performs better than *GES* on average on 30.40 out of 50 test instances of each problem. This record is only 0.15 out of 50 for *GES* over *GES* + *PIA*. Finally *GES* performs better than *SA* on average in 19.50 out of 50 test instances of each problem. The reverse scenario occurs only on average on nine test instances of each problem. Tracking the trends in the mean performance of the algorithms reveals that for a given number of jobs, as the number of machines increases, the mean performance decreases. Comparing the execution times, as the number of jobs increases, the time taken by the algorithms becomes longer. Also, increasing the number of machines adds to the running times.

The point is that the rate of increment is very smooth and much less than that of *SA*. *SA* times may even exceed from 40 s but for *GES* and *GES* + *PIA* all times are <2 s. This is because that *GES* and *GES* + *PIA* are able to find the optimum schedules faster and get stop sooner before reaching the maximum allowed number of schedule evaluations. But, *SA* fails to get the optimal schedules more

Table 5 Results for experiment E_3: $p \sim U(100{,}200)$

m	*n*	*SA*		*GES*		*GES + PIA*		*GES* versus *SA*	*GES + PIA* versus *SA*	*GES + PIA* versus *GES*
		Mean	Time (s)	Mean	Time (s)	Mean	Time (s)			
3	10	1.0122	0.2206	1.0119	0.4801	1.0119	0.5556	3 versus 0	3 versus 0	0 versus 0
	11	1.0137	0.2859	1.0134	0.4932	1.0125	0.5176	7 versus 7	10 versus 0	11 versus 0
	13	1.0024	0.3459	1.0017	0.4040	1.0010	0.2836	21 versus 8	28 versus 0	17 versus 0
	14	1.0021	0.3837	1.0016	0.3859	1.0007	0.1740	19 versus 9	31 versus 0	24 versus 0
	16	1.0010	0.5012	1.0003	0.2471	1.0000	0.0614	21 versus 3	28 versus 0	12 versus 0
	17	1.0011	0.5071	1.0002	0.3226	1.0000	0.0461	29 versus 0	33 versus 0	10 versus 0
5	16	1.0130	0.6381	1.0096	0.7104	1.0087	0.7684	30 versus 2	35 versus 0	13 versus 0
	17	1.0141	0.8053	1.0107	0.6895	1.0096	0.7903	28 versus 3	37 versus 0	17 versus 0
	21	1.0043	1.0784	1.0011	0.5809	1.0001	0.1769	38 versus 3	45 versus 0	27 versus 0
	22	1.0042	1.2265	1.0013	0.6542	1.0001	0.2040	39 versus 3	44 versus 0	29 versus 0
	26	1.0035	1.3965	1.0004	0.6102	1.0000	0.0680	46 versus 0	49 versus 0	19 versus 0
	27	1.0027	1.8209	1.0006	0.6539	1.0000	0.0702	40 versus 1	46 versus 0	22 versus 0
8	25	1.0118	1.5318	1.0045	0.9484	1.0031	1.0071	44 versus 1	47 versus 0	25 versus 0
	26	1.0129	1.5940	1.0043	0.9684	1.0023	0.9282	45 versus 1	49 versus 0	31 versus 0
	33	1.0060	2.9440	1.0009	0.9494	1.0000	0.1790	47 versus 0	49 versus 0	25 versus 1
	34	1.0068	3.1381	1.0014	0.9965	1.0000	0.1591	45 versus 2	50 versus 0	34 versus 0
	41	1.0051	4.4093	1.0009	1.0021	1.0000	0.0926	43 versus 1	49 versus 0	30 versus 0
	42	1.0047	4.8096	1.0010	1.0349	1.0000	0.0985	45 versus 3	47 versus 0	32 versus 0
10	31	1.0134	2.3415	1.0035	1.0645	1.0015	0.9706	50 versus 0	50 versus 0	31 versus 0
	32	1.0137	2.7656	1.0035	1.1057	1.0015	0.9799	50 versus 0	50 versus 0	35 versus 1
	41	1.0091	4.6090	1.0013	1.1144	1.0001	0.3332	49 versus 0	50 versus 0	28 versus 0
	42	1.0085	4.8631	1.0015	1.2015	1.0000	0.2246	48 versus 0	50 versus 0	36 versus 0
	51	1.0057	8.3668	1.0008	1.4589	1.0000	0.0767	50 versus 0	50 versus 0	31 versus 0
	52	1.0065	8.0540	1.0015	1.5565	1.0000	0.2062	48 versus 1	50 versus 0	41 versus 0
Average		**1.0074**	**2.4432**	**1.0032**	**0.8180**	**1.0022**	**0.3738**	**36.87 versus 2.00**	**40.83 versus 0.00**	**24.16 versus 0.04**

The bold rows are the average values over all rows

frequently and has to do its search until getting the maximum allowed number of schedule evaluation.

Tables 4, 5 and 6 present the results for experimental framework E_3 when the processing times are generated within $U(1{,}100)$, $U(100{,}200)$ and $U(100{,}800)$, respectively. As can be found, both *GES* and *GES* + *PIA* algorithms reveal better performance than *SA* on all problems in each table. There is no problem for which the mean performance of *SA* becomes less than that of *GES* or *GES* + *PIA*. There is also no instance (among $3 \times 24 \times 50$ test instances) for which *GES* could report a better performance in comparison with *GES* + *PIA*. This indicates that the mean performance of *SA* is not optimal since all of its ratios in Tables 4, 5 and 6 are greater than one. This means that for each problem, there are some test instances for which *SA* is not able to find the optimal makespan. Such an observation is almost true but less highlighted for *GES*. In Table 4, there are at least 13 problems out of 24 problems that *GES* + *PIA* is able to find the optimal makespan in all of 50 trials. This record is 11 out of 24 for *GES* + *PIA* in Table 5. As the range of processing times becomes wider, the gap between *GES* + *PIA* and *SA* in terms of the success rate becomes more sensible. In Table 4, on average on 25.16 test instances out of 50, *GES* + *PIA* performs better than *SA*. This record is 40.83 and 44.20 out of 50 in Tables 5 and 6, respectively. The sensible gap between the success rate of *GES* + *PIA* and *GES*, and between the success rate of *GES* and *SA* is also observable. Therefore, when the range gets broader, *GES* + *PIA* is expected to strictly outperform other algorithms.

Table 7 presents the results for experiment E_4. For the experiments in E_4 since the problem sizes are small, all of the algorithms perform almost similar. However, again *GES* + *PIA* algorithm performs slightly better. From the above discussions and from the results on the mean performance and success rate of algorithms we can conclude that the relation *GES* + *PIA* > *GES* > *SA* is held where ">" indicates better performance.

In Tables 3, 4 and 5, from the row with the caption "*Average,*" it can be inferred that the average time taken

Table 6 Results for experiment E_3: $p \sim U(100,800)$

m	*n*	*SA*		*GES*		*GES* + *PIA*		*GES* versus *SA*	*GES* + *PIA* versus *SA*	*GES* + *PIA* versus *GES*
		Mean	Time(s)	Mean	Time(s)	Mean	Time(s)			
3	10	1.0089	0.2215	1.0083	0.4839	1.0078	0.5377	10 versus 6	10 versus 0	6 versus 0
	11	1.0056	0.2803	1.0053	0.5107	1.0046	0.5400	10 versus 11	13 versus 2	10 versus 0
	13	1.0024	0.3703	1.0017	0.5176	1.0010	0.5619	24 versus 10	31 versus 1	21 versus 0
	14	1.0018	0.4834	1.0009	0.4779	1.0005	0.8769	35 versus 8	42 versus 0	25 versus 1
	16	1.0014	0.5665	1.0005	0.4764	1.0001	0.3366	34 versus 2	41 versus 0	25 versus 0
	17	1.0014	0.6343	1.0004	0.4717	1.0001	0.3357	40 versus 5	45 versus 0	30 versus 2
5	16	1.0102	0.5659	1.0076	0.7082	1.0058	0.8068	32 versus 4	42 versus 0	26 versus 0
	17	1.0088	0.7147	1.0050	0.6882	1.0033	0.7725	37 versus 6	47 versus 0	29 versus 1
	21	1.0046	1.1940	1.0023	0.7051	1.0007	0.7868	39 versus 8	46 versus 0	48 versus 0
	22	1.0046	1.2043	1.0020	0.7247	1.0005	0.7453	42 versus 5	50 versus 0	45 versus 0
	26	1.0035	1.7693	1.0013	0.7366	1.0002	0.7266	39 versus 9	49 versus 0	43 versus 1
	27	1.0029	1.9490	1.0012	0.7550	1.0002	0.7444	41 versus 7	49 versus 0	44 versus 1
8	25	1.0119	1.5862	1.0066	0.9413	1.0032	1.2760	33 versus 11	49 versus 0	42 versus 0
	26	1.0096	1.9043	1.0058	0.9597	1.0024	1.2732	37 versus 7	49 versus 0	47 versus 0
	33	1.0058	3.4806	1.0031	0.9893	1.0008	1.3827	40 versus 7	50 versus 0	49 versus 0
	34	1.0064	3.5006	1.0029	1.0174	1.0007	1.3350	42 versus 6	50 versus 0	47 versus 0
	41	1.0036	6.0312	1.0021	1.0489	1.0003	1.3338	37 versus 8	49 versus 0	49 versus 0
	42	1.0041	6.1071	1.0020	1.4352	1.0003	1.3759	39 versus 7	50 versus 0	49 versus 0
10	31	1.0125	2.5156	1.0070	1.1413	1.0030	1.6417	38 versus 9	50 versus 0	48 versus 0
	32	1.0118	2.9978	1.0059	1.1597	1.0023	3.1126	40 versus 7	49 versus 1	47 versus 0
	41	1.0059	5.8428	1.0035	1.1921	1.0008	1.7731	40 versus 9	50 versus 0	49 versus 0
	42	1.0066	6.8100	1.0035	1.2299	1.0009	1.8395	39 versus 8	50 versus 0	49 versus 0
	51	1.0046	9.2781	1.0025	1.2707	1.0004	1.9653	39 versus 9	50 versus 0	50 versus 0
	52	1.0042	9.8990	1.0026	1.2764	1.0004	1.9908	34 versus 14	50 versus 0	50 versus 0
Average		**1.0059**	**2.9127**	**1.0035**	**0.8715**	**1.0016**	**1.1696**	**35.04 versus 7.62**	**44.20 versus 0.16**	**38.66 versus 0.29**

The bold rows are the average values over all rows

Table 7 Results for experiment E_4

m	*n*	*P*	*SA*		*GES*		*GES* + *PIA*		*GES* versus *SA*	*GES* + *PIA* versus *SA*	*GES* + *PIA* versus *GES*
			Mean	Time(s)	Mean	Time(s)	Mean	Time(s)			
2	9	U(1,20)	1.0000	0.0040	1.0000	0.0028	1.0000	0.0037	0 versus 0	0 versus 0	0 versus 0
		U(20,50)	1.0009	0.0368	1.0009	0.0820	1.0009	0.0530	0 versus 0	0 versus 0	0 versus 0
		U(1,100)	1.0009	0.0412	1.0009	0.0882	1.0009	0.0876	0 versus 0	0 versus 0	0 versus 0
		U(50,100)	1.0038	0.1187	1.0040	0.2870	1.0038	0.2713	1 versus 2	1 versus 0	2 versus 0
		U(100,200)	1.0042	0.1418	1.0044	0.3247	1.0040	0.3185	3 versus 5	3 versus 0	5 versus 0
		U(100,800)	1.0017	0.1434	1.0018	0.3566	1.0017	0.3709	0 versus 3	0 versus 1	2 versus 0
3	10	U(1,20)	1.0014	0.0253	1.0014	0.0427	1.0014	0.0405	0 versus 0	0 versus 0	0 versus 0
		U(20,50)	1.0083	0.1418	1.0073	0.3394	1.0071	0.3017	6 versus 1	6 versus 0	1 versus 0
		U(1,100)	1.0095	0.1475	1.0097	0.3787	1.0092	0.3959	2 versus 2	2 versus 0	2 versus 0
		U(50,100)	1.0096	0.2012	1.0094	0.4274	1.0090	0.4477	4 versus 1	6 versus 0	3 versus 0
		U(100,200)	1.0167	0.2290	1.0162	0.5294	1.0162	0.5647	4 versus 0	5 versus 0	1 versus 0
		U(100,800)	1.0092	0.2246	1.0095	0.4948	1.0092	0.5591	6 versus 6	7 versus 1	6 versus 0
Average			**1.0055**	**0.1212**	**1.0054**	**0.2794**	**1.0052**	**0.2845**	**2.16 versus 1.66**	**2.50 versus 0.16**	**1.83 versus 0.00**

The bold rows are the average values over all rows

by *GES* + *PIA* algorithm is shorter than *GES*. This is due to the fact that hybridizing with *PIA* algorithm speeds up converging toward the global optimum. Hence, our second stopping condition, which is getting the lower bound value, is fired soon and *GES* + *PIA* algorithm stops. Paradoxically in Table 6, the time needed for *GES* is less than *GES* + *PIA*. Here the reason for the increment in the required running time of *GES* + *PIA* is most probably related to the performance of the lower bound. Inevitably, *GES* + *PIA* should generate the maximum allowable number of schedules before being stopped. We do not conduct the execution time comparison between *SA* and *GES* or *GES* + *PIA* since these algorithms have been run on different machines (the results of *SA* have been adopted from Husseinzadeh Kashan and Karimi [20]). All we say is that both of *GES* and *GES* + *PIA* are computationally efficient and can produce very good results for problems with 100 jobs and 10 machines in less than a second.

7 Conclusions and future research

The paper proposed a new solution methodology based on grouping evolution strategies (*GES*) for parallel-machines scheduling problem to minimize makespan. The proposed *GES* algorithm owns a mutation operator which is founded based on dissimilarity measure between the set of jobs assigned to a machine to work with job groups rather than jobs isolatedly. Therefore, these are the job groups that constitute the underlying building blocks of the solution in the parallel-machines scheduling problem. The rationale of the new mutation operator is analogous to the original mutation of *ES*. It works in the real-valued space while the consequences are used in the body of a two-phase algorithm to generate an offspring schedule in the discrete space with the aid of *LPT* rule. A pairwise interchange algorithm (*PIA*) was hybridized with *GES* to improve its local performance. Additionally, we generalized the design of *PIA* to work also for the problem of scheduling with nonidentical parallel-machines and proved that under suitable conditions it has descent property. Results acquired from extended computational experiments justify the competitive performance of the proposed algorithms compared to an already existing simulated annealing algorithm.

GES is very easy for implementation and can be regarded as a promising solver of the wide class of scheduling problems. For future research, the effectiveness of *GES* is worth examining on parallel-machines scheduling problem while taking account of machines' failure rates and the case of machines' unavailability. Considering criteria other than makespan and even multiple objectives would be worthwhile. Additionally, developing the grouping version of the recently introduced real-valued algorithms such as, league championship algorithm [18, 22], Optics Inspired Optimization [19] and Find-Fix-Finish-Exploit-Analyze (F3EA) metaheuristic algorithm [27] and evaluating their effectiveness on parallel-machines scheduling problem is particularly encouraged. Finally, applications of *GES* on other types of scheduling problems, e.g., task scheduling in cloud computing environment [1, 2] or scheduling batch processing machines [21], is of great interest.

References

1. Abdulhamid SM, Abd Latiff MS, Madni SHH, Abdullahi M (2016) Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm. Neural Comput Appl. doi:10.1007/s00521-016-2448-8
2. Abdullahi M, Ngadi MA, Abdulhamid SM (2016) Symbiotic organism search optimization based task scheduling in cloud computing environment. Future Gener Comput Syst 56(1):640–650
3. Balin S (2011) Non-identical parallel machine scheduling using genetic algorithm. Expert Syst Appl 38(6):6814–6821
4. Bathrinath S, Sankar SS, Ponnambalam SG, Kannan BKV (2013) Bi-objective optimization in identical parallel machine scheduling problem. In: Swarm, evolutionary, and memetic computing. Springer International Publishing, pp. 377–388
5. Bathrinath S, Sankar SS, Ponnambalam SG, Leno IJ (2015) VNS-based heuristic for identical parallel machine scheduling problem. In: Artificial intelligence and evolutionary algorithms in engineering systems. Springer India, pp 693–699
6. Chen J, Pan QK, Wang L, Li JQ (2012) A hybrid dynamic harmony search algorithm for identical parallel machines scheduling. Eng Optim 44(2):209–224
7. Coffman EG, Garey MR, Johnson DS (1978) An application of bin-packing to multi-processor scheduling. SIAM J Comput 7:1–17
8. Dell'Amico M, Martello S (2005) A note on exact algorithms for the identical parallel machine scheduling problem. Eur J Oper Res 160(2):576–578
9. Dell'Amico M, Martello S (1995) Optimal scheduling of tasks on identical parallel processors. ORSA J Comput 7(2):191–200
10. Diana ROM, de França Filho MF, de Souza SR, de Almeida Vitor JF (2015) An immune-inspired algorithm for an unrelated parallel machines' scheduling problem with sequence and machine dependent setup-times for makespan minimisation. Neurocomputing 163:94–105
11. Falkenauer E (1994) New representation and operators for GAs applied to grouping problems. Evol Comput 2:123–144
12. Fatemi Ghomi SMT, Jolai Ghazvini F (1998) A pairwise interchange algorithm for parallel machine scheduling. Prod Plan Control 9:685–689
13. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco
14. Gharbi A, Haouari M (2007) An approximate decomposition algorithm for scheduling on parallel machines with heads and tails. Comput Oper Res 34:868–883
15. Graham RL (1969) Bounds on multiprocessor timing anomalies. SIAM J Appl Math 17:416–429
16. Gupta JND, Ruiz-Torres AJ (2001) A LISTFIT heuristic for minimizing makespan on identical parallel machines. Prod Plan Control 12:28–36

17. Hashemian N, Diallo C, Vizvári B (2014) Makespan minimization for parallel machines scheduling with multiple availability constraints. Ann Oper Res 213(1):173–186
18. Husseinzadeh Kashan A (2014) League championship algorithm (LCA): a new algorithm for global optimization inspired by sport championships. Appl Soft Comput 16:171–200
19. Husseinzadeh Kashan A (2015) A new metaheuristic for optimization: optics inspired optimization (OIO). Comput Oper Res 55:99–125
20. Husseinzadeh Kashan A, Karimi B (2009) A discrete particle swarm optimization algorithm for scheduling parallel machines. Comput Ind Eng 56:216–223
21. Husseinzadeh Kashan A, Karimi B (2009) An improved mixed integer linear formulation and lower bounds for minimizing makespan on a flow shop with batch processing machines. Int J Adv Manuf Technol 40:582–594
22. Husseinzadeh Kashan A, Karimi B (2010) A new algorithm for constrained optimization inspired by the sport league championships. In: IEEE world congress on computational intelligence, WCCI 2010, pp 487–494
23. Husseinzadeh Kashan A, Karimi B, Jenabi M (2008) A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. Comput Oper Res 35:1084–1098
24. Husseinzadeh Kashan A, Jenabi M, Husseinzadeh Kashan M (2009) A new solution approach for grouping problems based on evolution strategies. In: International conference of soft computing and pattern recognition
25. Husseinzadeh Kashan A, Rezaee B, Karimiyan S (2013) An efficient approach for unsupervised fuzzy clustering based on grouping evolution strategy. Pattern Recogn 46:1240–1254
26. Husseinzadeh Kashan A, Akbari AA, Ostadi B (2015) Grouping evolution strategies: an effective approach for grouping problems. Appl Math Model 2015(39):2703–2720
27. Husseinzadeh Kashan A, Tavakkoli-Moghaddam R, Gen M (2016) A warfare inspired optimization algorithm: the find-fix-finish-exploit-analyze (F3EA) metaheuristic algorithm. In: Tenth international conference on management science and engineering management, ICMSEM 2016, pp 393–408
28. Iori M, Martello S (2008) Scatter search algorithms for identical parallel machine scheduling problems. In: Metaheuristics for scheduling in industrial and manufacturing applications. Springer, Berlin, pp 41–59
29. Jing C, Guang-Liang L, Ran L (2011) Discrete harmony search algorithm for identical parallel machine scheduling problem. In: IEEE control conference (CCC), 2011 30th Chinese, pp 5457–5461
30. Kedia SK (1971) A job scheduling problem with parallel processors. Technical Report. Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor
31. Kowalczyk D, Leus R (2016) An exact algorithm for parallel machine scheduling with conflicts. J Sched 1–18
32. Kuruvilla A, Paletta G (2015) Minimizing makespan on identical parallel machines. Int J Oper Res Inf Syst (IJORIS) 6(1):19–29
33. Lee CY, Massey JD (1988) Multiprocessor scheduling: combining LPT and MULTIFIT. Discrete Appl Math 20:233–242
34. Lee WC, Wu CC, Chen P (2006) A simulated annealing approach to makespan minimization on identical parallel machines. Int J Adv Manuf Technol 31:328–334
35. Low C, Wu GH (2016) Unrelated parallel-machine scheduling with controllable processing times and eligibility constraints to minimize the makespan. J Ind Prod Eng 33(4):286–293
36. Mellouli R, Sadfi C, Chu C, Kacem I (2009) Identical parallel-machine scheduling under availability constraints to minimize the sum of completion times. Eur J Oper Res 197(3):1150–1165
37. Min L, Cheng W (1999) A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. Artif Intell Eng 13(4):399–403
38. Mokotoff E (2004) An exact algorithm for the identical parallel machine scheduling problem. Eur J Oper Res 152(3):758–769
39. Pakzad-Moghaddam SH (2016) A Lévy flight embedded particle swarm optimization for multi-objective parallel-machine scheduling with learning and adapting considerations. Comput Ind Eng 91:109–128
40. Rechenberg I (1964) Cybernetic solution path of an experimental problem. Library Translation 1122, August 1965. Famborough Hants: royal aircraft establishment. English translation of lecture given at the Annual Conference of the WGLR, Berlin, 1964
41. Rechenberg I (1973) Evolutionsstrategie: optimierung technischer systeme nach den prinzipien der biologischen evolution. Frommann-Holzboog, Stuttgart
42. Zarandi MF, Kayvanfar V (2014) A bi-objective identical parallel machine scheduling problem with controllable processing times: a just-in-time approach. Int J Adv Manuf Technol 1–19