Grouping evolution strategies: an efficient approach for grouping problems

Ali Husseinzadeh Kashan

Department of Industrial Engineering, Faculty of Engineering, Tarbiat Modares University, Tehran, Iran.

E-mail address: a.kashan@modares.ac.ir

Abstract

Many combinatorial optimization problems include a grouping (or assignment) phase wherein a set of items are partitioned into disjoint groups or sets. Introduced in 1994, the grouping genetic algorithm (*GGA*) is the most established heuristic for grouping problems which exploits the structural information along with the grouping nature of these problems to steer the search process. The aim of this paper is to evaluate the grouping version of the classic evolution strategies (*ES*) which originally maintain the well-known Gaussian mutation, recombination and selection operators for optimizing non-linear real-valued functions. Introducing the grouping evolution strategies (*GES*) to optimize the grouping problems that are intrinsically discrete, requests for developing a new mutation operator which works with groups of items rather than scalars and is respondent to the structure of grouping problems. As a source of variation, *GES* employs a mutation operator which shares a same rational with the original *ES* mutation in the way that it works in continuous space while the consequences are used in discrete search space. A two phase heuristic procedure is developed to generate a complete feasible solution from the output of the mutation process. An extensive comparative study is conducted to evaluate the performance of *GES* versus *GGA* and *GPSO* (a recently proposed grouping particle swarm optimization algorithm) on test problem instances of the single batch-processing machine scheduling problem and the bin-packing problem. While these problems share exactly a same grouping structure and the performance of *GES* on both problems is reliable, switching from one problem to another deteriorates the performance of *GGA*. Though such a deficiency is not observed in the performance of *GPSO*, it is still inferior to *GES* on single batch-processing machine scheduling test problem instances. Beside such empirical outcomes, the paper conveys a number of core strengths that the design of *GES* supports them but the design of *GGA* does not address them.

Keywords: grouping problems, grouping genetic algorithm, grouping evolution strategies, grouping particle swarm optimization, batch-processing machine scheduling problem, bin-packing problem.

1. Introduction

Falkenauer (1994) defines a grouping or assignment type problem as the one where the aim is partitioning a set V of n items into a collection of mutually disjoint subsets (groups) G_i such that: $V = \bigcup_{i=1}^{D} G_i$ and $G_i \cap G_j = \varnothing,\ i \neq j.$ The above definition says that in a grouping problem the aim is to partition the members of V into D $(1 \leq D \leq n)$ different groups such that each item being assigned exactly to one group. Originally it is assumed that the ordering of groups is not relevant in grouping problems. However, there are many grouping type problems in which the ordering of groups is important.

Many NP-hard combinatorial optimization problems such as graph coloring problem, bin-packing problem, batch-processing machine scheduling problem, line-balancing problem, timetabling problem, identical/non-identical parallel-machines scheduling problem, cell formation problem, pickup and delivery problem etc, are well used examples of grouping problems.

In most of grouping problems, not all possible groupings are permitted since the group formations must be in such a way that a number of constraints being satisfied. Besides grouping constraints, the groups are usually formed based on an objective function which is founded on the composition of groups. Therefore, the building blocks that should be taken into consideration in an evolutionary search should be the groups or the group segments, but not the items isolatedly.

In terms of the number of groups (D) in a given solution of a grouping problem, two categories of problems are recognizable. The *constant grouping problems* are those problems in which the number of groups (D) is an input constant to the problem. Here, in terms of the number of groups, solutions of these problems are all at the same length. An example of this type of grouping problems is the identical/non-identical parallel-machines scheduling problem in which a number of jobs should be processed just by one of D available machines working in parallel to minimize the makespan. Here the task is to decompose the set of available jobs into D subsets where each subset i contains all task that should be processed by machine i. As can be seen any solution to these problems must partition the set of jobs into D subsets. On the other side, there are grouping problems in which, D is not known in advance and the objective is to find a feasible grouping yielding the minimum D. Let us refer to these problems as *variable grouping problems.* Bin-packing problem, single batch-processing machine scheduling problem and graph coloring problem are some examples of variable grouping problems.

We can further classify the grouping problems based on the type of groups, e.g., *identical* or *non-identical*. A grouping problem is referred to as the one with *non-identical groups* when the

groups differ in their characteristics. If we exchange the whole content of two groups in a given solution of such problems, the resultant grouping differs from the original grouping. For example, let us consider the non-identical parallel-machines scheduling problem in which the processor machines differ in their operational characteristics such as processing speed, cost, etc. Given the fact that the set of jobs assigned to each machine constitute a group, these groups are not identical in the sense that their corresponding possessors are different. A grouping problem with *identical groups* is the one in which all groups are similar in their characteristics. Here, the complete exchange of items between two groups does not change the situation and hence the ordering between groups is irrelevant. The identical parallel-machines scheduling problem, bin-packing, single batch-processing machine and graph coloring problems are among the grouping problems with identical groups.

There are many grouping problems in which the ordering between groups is important and the quality of the solutions in terms of the problem objective is influenced by the way in which groups are ordered. Such grouping problems are referred to as the *order dependent* grouping problems (Lewis, 2009). An example of this family of problems is the university exam time tabling problem. In an intuitive way we can introduce the *order independent* grouping problems.

Falkenauer (1994) explains that the most commonly used representations for grouping problems, e.g., number encoding and order-based representation suffer from redundancies. In the number encoding, the value of the *k*th gene represents the group that item *k* is in. For example the individual 21321 encodes the grouping in which the first item is in group 2, the second item in 1, the third item in 3, and so on. However, it is easy to check that the individual 12312 encodes exactly the same solution for a typical grouping problem with identical groups. Moreover, under number encoding, if some constraints on the groups exist, the resulting chromosomes of crossover stage will certainly contain many illegal groups. The indirect order-based representation uses a decoder to build solutions from permutations of the items. Drawbacks of such typical representations have been presented in (Falkenauer, 1994; Falkenauer 1996). To remedy these drawbacks, Falkenauer introduced the *group encoding* and used it in genetic algorithm (*GA*). The idea of group encoding is that the items belonging to the same group should be placed into the same partition. For instance, the above individual can be represented as {{2,5}, {4,1}, {3}}. Using this encoding scheme, the genetic operators can work on groups rather than items unlike in number encoding (note that the ordering within and between partitions is irrelevant). The rationale is that in grouping problems these are the groups that are the innate building blocks of the problem, which can convey information on the expected quality of the solution they are part of, and not the particular positions of any one item on its own. Therefore the representations and resulting evolutionary search

operators need to be defined such that they allow the groupings of items to be propagated. For a review of suitable encoding representations for grouping problems, readers may refer to Ülker *et al*, (2007) and Ülker *et al*, (2008).With this in mind, a standard *grouping genetic algorithm* (*GGA*) has been proposed by Falkenauer in 1994 which is a genetic algorithm that uses group encoding and related operators for solving grouping problems. There has since been applications of *GGA* to a number of grouping problems, with varying degrees of success. Table 1 gives a list of problems where *GGA* has been successfully applied to them.

>>Insert Table 1 about here<<

Almost all researches that have used group encoding and operators have only relied upon genetic algorithm (*GA*) as their evolutionary search mechanism. A lot of *GGA* methodologies have been emerged and adapted to different grouping problems, without any effort put on developing the grouping version of other meta-heuristics (e.g., simulated annealing (*SA*), tabu search (*TS*), evolution strategies (*ES*), Particle swarm optimization (*PSO*) etc). The notion of group related encoding and operators can be simply applied to *SA* to obtain the grouping version of *SA* (*GSA*) (this can be done using *GGA* mutation operator as the source of neighborhood generation in *SA*). While such a notion works also for *TS* too, it is not applicable for *ES* or *PSO* to obtain *GES* or *GPSO* algorithms. The difficulty with developing *GES* or *GPSO* is due to the following facts.

1) Both of *ES* and *PSO* have a dedicated mutation equation to produce new real-valued solution vectors during the search process. Therefore, the development of *GES* or *GPSO* should be based on introducing comparable mutation equations that work with groups/sets in place of real-valued scalars. The new equations should be developed in such a way that they inherit the major characteristics of their classic counterpart.
2) Both of *ES* and *PSO* have been originally introduced for optimizing non-linear numerical functions in real-valued search space. Without any doubt, all grouping problems are discrete since they include 0-1 assignment variables. Using a proper function to quantify the degree of dissimilarity between two groups/sets, it will be shown how we can keep the new comparable mutation working in continuous space and use its consequences in a two phase heuristic to generate a new complete solution in discrete space.

The prototype of *GES* has been first introduced in SoCPaR 2009 conference by the first author (Husseinzadeh Kashan *et al*, 2009). However, there is no detailed study on the true effectiveness and efficiency of *GES* on grouping problems. Moreover, the design strengths of *GES* and its

advantages over *GGA* have remained unexplored. During these years, a grouping version of *PSO* has been also introduced by Husseinzadeh Kashan *et al*, (2013a). It is therefore of the particular interest to examine the strengths of *GES* over *GPSO*, too.

In this paper we extend the outcome of our previous research to find out its merit and real potency when dealing with grouping problems. An extensive comparative study is conducted to evaluate the performance of *GES* versus *GGA* (in three versions) and *GPSO* on test problem instances of the single batch-processing machine scheduling problem and the bin-packing problem. While these problems share exactly a same grouping structure and the performance of *GES* on both problems is reliable, switching from one problem to another deteriorates the performance of *GGA*, significantly. Beside such empirical outcomes, the paper conveys a number of core strengths that the design of *GES* supports them but the design of *GGA* or *GPSO* does not necessarily address them.

The paper is organized as follows. In the two following sections we briefly review the basic preliminaries of *ES* and *GGA*. In section 4 we show how to construct the new mutation equations of *GES* and how to build the new solution, accordingly. Section 5 introduces a grouping version of the particle swarm optimization algorithm (*GPSO*) as a basis to evaluate the performance of *GES*. Section 6 introduces briefly the grouping problems on which the performance of *GGA*, *GPSO* and *GES* is tested. These problems are the single batch-processing machine scheduling problem and its special case, the bin-packing problem. Section 7 gives more details on the implementation issues. Effectiveness comparisons are investigated in Section 8. The paper will be concluded in Section 9.

2. Evolution strategies

According to Darwin's theory about the development of species, the principle of variation and selection can be considered as the fundamental principles of the Darwinian evolution. Evolution strategies (*ES*) of Rechenberg (1964) 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 $(\mu/\rho^{+},\lambda)-ES$ notation. All members operate with a population Π^t, which contains μ individuals (time proceeds in discrete steps (generation) and is indicated by superscript t). In each generation *t*, a set Q^t of λ offspring solutions are produced from Π^t via recombination and mutation operators. The symbol ρ indicates the number of parental solutions involved in the creation of every single offspring solution. When $\rho=1$, it will be omitted. The new population Π^{t+1} is created by means of the selection schemes based on individual fitness.

Selection is the goal-directed component of the evolutionary search and is based on the ranking of the individuals' fitness. Selection in *ES* is symbolized with "$^{+}_{,}$", denoting the two mutually exclusive selection types. Depending on the selection type, selection can be either from $\Pi^t \cup Q^t$ or from Q^t. Using "+" 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}. The process of creating an offspring candidate solution involves recombination and mutation. In the *Recombination* process, ρ parental candidate solutions are selected at random and their centroid is computed. *Mutation* operator introduces small variations by adding a point symmetric perturbation to the result of recombination. This perturbation is drawn from an isotropic normal distribution. For a population $\Pi^t = \{X_1^t, X_2^t, ..., X_\mu^t\}$ in which $X_k^t = (x_{k1}^t, x_{k2}^t, ..., x_{kD}^t)\ \forall k = 1, ..., \mu$, is a D dimensional candidate solution in the real-valued search space, the set of offspring candidate solutions (Q^t) consists of vectors such as $Y_i^t = (y_{i1}^t, y_{i2}^t, ..., y_{iD}^t)\ \forall i = 1, ..., \lambda$, where:

$$y_{id}^t = \frac{1}{\rho} \sum_{k=1}^{\rho} x_{i_k d}^t + z_d^t, \qquad \forall d = 1, ..., D,\ \forall i = 1, ..., \lambda \tag{1}$$

with $z_d^t = \sigma^t N_d(0,1)$, where $N_d(0,1)$ is a normally distributed random scalar associated to dimension d (i.e., the variation source). The scalar variable σ^t, which is learnt during the evolution process, is known as the strategy parameter or mutation strength. It determines the expected variation of an offspring from the centroid of its parents. There may be situations where instead of one endogenous strategy parameter it is beneficial to have a vector $\sigma^t = (\sigma_1^t, \sigma_2^t, ..., \sigma_D^t)$ of strategy parameters. Indices i_k are independently drawn with replacement and with equal probability from $\{1, ..., \mu\}$.

The mutation strength σ^t needs to be adapted during the search. Too low values for mutation strength can slow down the progress, while too high values may lead to divergence. Therefore, a mutation strength adaptation component is an important part of evolution strategies. The first mutation strength adaptation scheme, known as 1/5-success rule, was proposed by Rechenberg (1973) for the $(1+1)-ES$. Rechenberg found that in order to obtain a nearly optimal performance of the $(1+1)-ES$ in real-valued search spaces, tune the mutation strength in such a way that the success rate is about 1/5. For tuning mutation strength, it is enough to monitor the portion of time an offspring candidate solution is superior to its parent over a number of time steps (i.e., estimate the success probability). If the observed estimate of the success probability exceeds 1/5, σ^t must

be increased, whereas in the opposite case the mutation strength must be decreased. Here, we abstain to give more introduction to *ES* mechanism, its variants and alternative operators. For greater details, reader may refer to Beyer and Schwefel (2002), and Arnold and Beyer (2003).

3. An introduction to grouping genetic algorithm (*GGA*) components

GGA is a class of evolutionary algorithms especially modified to exploit the structural knowledge of grouping problems through using an appropriate chromosomal representation and genetic operators. In the remainder of this section we review shortly the group encoding and operators, since we need to refer them in the subsequent parts of the paper.

3-1. The group encoding

There are many applications of *GA* to solve grouping problems, but what makes *GGA* a well-designed methodology is the type of encoding used by Falkenauer (1994). In general, the group encoding consists of two parts, namely: an item part and a group part containing the group composition. The item part consists of an array of size n (n is the number of items). The group part keeps a list of labels associated to each groups of the solution. The kth member in the item part can take any of the D group labels and represents the group to which kth item is assigned. To illustrate the encoding scheme used in *GGA*, let us consider the example in Figure 1 which shows a grouping of 5 items and its related group encoding. The letters represent the groups and the numbers represent the items. In fact, the chromosome related to the solution could also be introduced as {{2,5}, {4,1}, {3}}. The length of the item part is fixed for a given problem, but the group part length is not fixed and varies from one individual to another. Therefore, different chromosomes in the same population may have different lengths.

>>Insert Figure 1 about here<<

The grouping representation encodes on a basis of one gene for one group. This gives an ability to devise for operators that work on the group part of the chromosomes. The group encoding respects the spirit of the building blocks because *GGA* always manipulate groups as the meaningful building blocks. One item taken isolatedly has little or no meaning during the search process (Falkenauer, 1994).

3-2. The GGA crossover operator

Crossover is one of the most important operators in genetic algorithms. The crossover pattern used for the *GGA* is shown in Figure 2 (De Lit et al, 2000). As can be seen from Figure 2, in the last step of crossover operator, a problem dependent method must be used for reinsertion of missing items to construct a perfect solution. Therefore, the crossover used will not be the same for all of grouping problems.

>>Insert Figure 2 about here<<

3-3. The GGA mutation operator

Mutation operator includes inserting small perturbations in an individual in order to explore new regions of the search space and also escape from local optima when the algorithm is near convergence. The mutation operator of *GGA* must work with groups rather than items. A simple mutation strategy is to randomly choose a few groups and remove them from the solution. The items belonging to the eliminated groups are thus missing from the solution. The method used for the insertion of missing items back to groups is generally same as the one used for the crossover operator.

3.4- The GGA inversion operator

The role of the inversion operator in *GGA* is to change the position of some groups in a given individual via overturning the order of groups between two crossing points in the group part of the individual, without influencing the membership of items. The idea behind using such an operator is due to the fact that a single solution may have different redundant presentations, and because the position of crossing sites has an influence on the work of crossover operator, the way in which a solution is presented influences the crossover results. Since the first group in the group part of a chromosome has a less probability to be chosen than the other groups, it is therefore important to include the inversion operator in *GGA*. Figure 3 illustrates the mechanism of inversion operator through an example.

>>Insert Figure 3 about here<<

4. On developing the grouping evolution strategies (*GES*)

When comparing *GES* with the classic *ES*, differences are exactly similar to those obtained by comparing *GGA* and the classic *GA*. Unlike *ES* which works with the commonly used

representations, *GES* employs the group encoding in order to take the advantages of incorporating the structural knowledge along with grouping problems. Under group encoding, a particular mutation operator is used in *GES* which works based on the composition of groups and not the items isolatedly. The mutation strategy is implemented via a two phase heuristic and more importantly, it maintains the major characteristics of the classic mutation equation of *ES*.

4.1. Solution encoding

In order to tackle a problem using an evolutionary algorithm, the candidate solutions must be encoded in a suitable form. For different combinatorial optimization problems, problem-specific solution encodings are necessary. The representation used in *GES* is the group encoding shortly explained in Section 3-1.

When optimizing a real-valued function by *ES*, each solution is represented with a vector of length D of real numbers (D is the number of variables). In a similar way, one can represent a solution i of a given grouping problem with D_i number of groups, as a structure whose length is equal to the number of groups. Such a structure encodes on a basis of one gene/block for one group (see the structure in the left side of Figure 1). Since the aim in a grouping problem is to determine the optimal composition of the groups, here groups can be treated as the problem variables.

It should be noted that the feasible solutions to a grouping problem do not essentially share the same number of groups. This is especially true for the variable grouping problems such as bin-packing problem, single batch-processing machine scheduling problem or graph coloring problem, in which the number of groups (i.e., bins, batches or colors) is not constant. Therefore, the design of *GES* should be in such a way that makes it enable to deal with solutions with different lengths in the same population. Through the paper, we represent the number of groups in a feasible solution X by D_X.

4.2 The *GES* mutation operator

Given the group encoding, the main motivation behind *GES* is to reconstruct the *ES* mutation equation and at the same time to preserve its major characteristics, to obtain the one which facilitates working with groups of items instead of scalars. Reconstructing (1) to enforce working with groups, the key idea would be to use appropriate set functions (a function whose input is a set and its output is usually a number) in place of arithmetic operators.

In particular, *GES* uses a group dissimilarity measure as a set function which serves with the aim similar to that of arithmetic subtraction ("–"), which is used implicitly in the mutation equation

of *ES*. Similar to arithmetic subtraction which finds the difference between two scalars, a group dissimilarity measure quantifies the degree of difference or dissimilarity between two groups. We use $Dissimilarity(G,G')$ to quantify the amount of dissimilarity between groups *G* and G'. That is, to quantify how similar *G* and G' are to each other or how far apart they are from each other.

Let $|G|$ demonstrate the cardinality of set *G* as a measure of the number of elements of *G*. There a number of coefficient of dissimilarities which are fit to our purpose of quantifying the amount of dissimilarity between *G* and G'. Among them are:

- Kulczynski's coefficient (Kulczynski, 1927) which is defined as:

$$Kulczynski\ \ Dissimilarity(G,G') = 1 - \frac{1}{2}\left(\frac{|G \cap G'|}{|G|} + \frac{|G \cap G'|}{|G'|}\right) \tag{2}$$

- Sørensen–Dices' coefficient (Sørensen, 1948; Dice, 1945) which is defined as:

- $$Sorensen\text{-}Dice\ Dissimilarity(G,G') = 1 - \frac{2|G \cap G'|}{|G| + |G'|} \tag{3}$$

- Tversky's coefficient (Tversky, 1977) which is defined as:

$$Tversky\ Dissimilarity(G,G') = 1 - \frac{|G \cap G'|}{|G \cap G'| + \alpha|G - G'| + \beta|G' - G|},\ \ \alpha,\beta \geq 0 \tag{4}$$

However, the presumably most widely used dissimilarity measure is the Jaccard's coefficient of dissimilarity (Jaccard, 1901) defined as follows:

$$Jaccard\ Disimilarity\ (G,G') = 1 - \frac{|G \cap G'|}{|G \cup G'|} \tag{5}$$

This measure has a clear direct interpretation as the proportion of units present in *G* or G', but not in both of them. The measure is equal to 0 if $G = G'$ and is equal to 1 if $G \cap G' = \varnothing$ and in general $0 \leq Jaccard\ Disimilarity\ (G,G') \leq 1$. While the triangle inequality is not fulfilled for the Kulczynski, Sørensen–Dice and Tversky coefficients it is held for Jaccard's coefficient and hence it is a metric. Although we can use any of the mentioned dissimilarity coefficients to develop the mutation operator of *GES*, we will use only the Jaccard's coefficient of dissimilarity due to its simplicity and most popularity.

Now we are ready to develop the most essential part of *GES* based on the concept of group dissimilarity coefficient. To develop an analogous Gaussian mutation for *GES*, let us set $\rho = 1$. This means that the new offspring inherits only from one parent; for the reason that the centroid of several groups may have no direct meaning. Reshaping classic *ES* mutation (1) in form of $y_{id}^t - x_{i_k d}^t = z_d^t$ and substituting the subtraction operator ("–") by "*Dissimilarity*" operator (i.e.,

Jaccard's coefficient of Dissimilarity), the comparable Gaussian mutation in *GES* can be introduced as follows:

$$Disimilarity(y_{id}^t, x_{i_k d}^t) \approx z_d^t \tag{6}$$

where $d = 1,...,D_{X_{i_k}^t}$, $i = 1,...,\lambda$ and $z_d^t = \sigma^t N_d(0,1)$. Indices i_k are drawn independently from $\{1,...,\mu\}$ with replacement. The reader should note that in (6), $x_{i_k d}^t$ and y_{id}^t denote the *d*th group in solutions $X_{i_k}^t$ and Y_i^t respectively, while in (1) these are scalars. The symbol "≈" predicates "almost equal". Here we abstain to use "=", for the reason that it may not be possible to form the new group y_{id}^t of the offspring solution Y_i^t, such that $Dissimilarity(y_{id}^t, x_{i_k d}^t) = z_d^t$. The reader should note that the running index *d* is used to differentiate the group caption and not to address the group order

Comparing both sides in (6), while on the right side there is an isotropic normal random variable which can take any arbitrary value, the range of "*Dissimilarity*" operator in the left side is only real values in [0,1]. This is evidence that z_d^t may not be a suitable source of variation in *GES* due to its wide range of possible values. Hence, we should devise for a different type of random variable as an alternative source of variations. Considering the range of *Dissimilarity* values, it is highly desirable that the candidate random variable takes 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 for a flexible PDF that provides different chances for getting a value in a specific sub-range in [0,1] by means of the function input parameter(s). There exist already several types of such ideal PDFs, e.g., the Triangular distribution, Beta distribution, Kumaraswamy distribution, Logit-normal distribution, etc. All of these distributions can cover only [0,1] domain and are versatile enough (especially, the Beta, Kumaraswamy and Logit-normal distributions are very flexible in shape). The triangular distribution has three parameters, but the other distributions have two parameters. Although, the Logit-normal distribution takes both J shape and humped shape which are fit to mutation purpose, for some values of scale parameter it takes U shape which is not suitable. The Kumaraswamy distribution is as versatile as the Beta distribution but has simple closed forms for both the CDF and the PDF. However, we choose Beta distribution because it has one input parameter less than the Triangular distribution, it models skew quite well, it is more popular than Kumaraswamy distribution and many programming software (like MATLAB 7.3.0) have a Beta random number generation module.

The Beta distribution is introduced by $f_X(x,\alpha,\beta)=\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)+\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1}$, where α, β>0 are shape parameters and Γ is the gamma function. Figure 4a illustrates some instances of Beta PDFs for various values of the shape parameters.

In conclusion, we are at the point to use Beta distribution in place of isotropic normal distribution typically used in *ES*, as the source of variation in *GES*. From (6), we obtain the mutation relation in *GES* as follows:

$$Dissimilarity(y_{id}^t, x_{i_k d}^t) \approx Beta_d(\alpha^t, \beta^t), \quad \forall d = 1,...,D_{X_{i_k}^t}, \forall i = 1,...,\lambda, \; i_k \in \{1,...,\mu\}, \tag{7}$$

where $Beta_d(\alpha^t, \beta^t)$ is a beta random number associated to group *d* with shape parameters α^t and β^t. Using isotropic normal distribution in *ES*, only one strategy parameter σ^t should be learnt during the evolution process. While, the mutation relation in (7) calls for updating two strategy parameters α^t and β^t (initialized at α^0 and β^0, respectively) during the search process. Back to Figure 4a, only the unimodal J shape and humped shape beta PDFs are suitable to supply variations. Because U shape PDFs are symmetric bimodal functions which put a higher density on the extreme values. Through U shape beta PDFs, it is more likely to get random values nearby either 0 or 1 rather than interior values. This implies that with equal chance, y_{id}^t must be similar or dissimilar to $x_{i_k d}^t$, which seems unrealistic. Figure 4b demonstrates that fixing one of the shape parameters, say β^t, at a proper level consents our aim of modeling skew through unimodal Beta PDFs. Figure 5 shows the relationship between different values of Beta shape parameters and the Beta PDF shape. It can be seen that either for $\alpha \geq 1$ or $\beta \geq 1$, Beta distribution takes all unimodal forms suitable for our purpose. Keeping the value of β^t constant and equal to β ($\beta > 1$), we can only consider α^t as the endogenous strategy parameter. The final *GES* mutation relation is now obtained as follows:

$$Dissimilarity(y_{id}^t, x_{i_k d}^t) \approx Beta_d(\alpha^t, \beta), \quad \forall d = 1,...,D_{X_{i_k}^t}, \forall i = 1,...,\lambda, \; i_k \in \{1,...,\mu\}. \tag{8}$$

>>Insert Figure 4 and 5 about here<<

4.3. Generation of the new solution

Similar to *GGA*, generation of the new solution in *GES* takes two phases. The first phase is the *inheritance phase*, in which the decision is made on those parts of the parent that the offspring should inherit. This phase is done by copying or maintaining parents' groups in *GGA*. During the

inheritance phase a number of items may be disappeared from groups. Therefore, in the second phase which is the *post-assignment* or *reinsertion phase*, missing items are put in the existing or new groups (see Figure 2).

In *GES*, the inheritance phase is triggered by (8) which implies that the construction of the new group y_{id}^t of Y_i^t, during the inheritance phase, should be in such a way that its dissimilarity degree with the parent group ($x_{i_k d}^t$) be around the value $Beta_d(\alpha^t, \beta)$. This implies that the similarity degree between $x_{i_k d}^t$ and y_{id}^t should be almost equal to $1 - Beta_d(\alpha^t, \beta)$. Let $n_{id}^t = | y_{id}^t \cap x_{i_k d}^t |$ denotes the number of shared items between groups $x_{i_k d}^t$ and y_{id}^t. The shared items which form the composition of the new group y_{id}^t partially, are indeed one of the parts that the offspring solution Y_i^t inherits from the parent solution $X_{i_k}^t$. Based on the definition of Jaccards's coefficient of dissimilarity, the value of n_{id}^t has a direct influence on keeping the value of $Dissimilarity(y_{id}^t, x_{i_k d}^t)$ close to the value of $Beta_d(\alpha^t, \beta)$.

It is reasonable that during the inheritance phase we assume $y_{id}^t \subseteq x_{i_k d}^t$ (because, y_{id}^t can inherit up to all items of $x_{i_k d}^t$). Given the value of $Beta_d(\alpha^t, \beta)$ and Starting from (8), we have:

$$Dissimilarity(y_{id}^t, x_{i_k d}^t) = Jaccard\ Dissimilarity(y_{id}^t, x_{i_k d}^t) = 1 - \frac{n_{id}^t}{|x_{i_k d}^t|} \approx Beta_d(\alpha^t, \beta)$$

$$\Rightarrow n_{id}^t \approx (1 - Beta_d(\alpha^t, \beta))|x_{i_k d}^t|. \quad (9)$$

Since n_{id}^t is the number of shared items, it must be an integer value. It is therefore set $n_{id}^t = \lfloor (1 - Beta_d(\alpha^t, \beta))|x_{i_k d}^t| \rfloor$, where $\lfloor . \rfloor$ denotes the integer part. Once the value of n_{id}^t was determined by (9), the inheritance phase is started and n_{id}^t items are selected from $x_{i_k d}^t$ and are copied into y_{id}^t (initialized as a null group). Thereafter, the post-assignment phase is started to reassign orphaned items to existing or new groups.

The above logic for generating the offspring candidate solution Y_i^t from the parent solution $X_{i_k}^t$ has been summarized in the following *NSG* (New Solution Generator) algorithm.

Algorithm NSG

Step 1. *For each group d* $(d = 1,...,D_{X^t_{i_k}})$ *of* $X^t_{i_k}$, *draw a random number from Beta distribution with shape parameters* α^t, β *and give its value to* $Beta_d(\alpha^t, \beta)$; *Compute* $n^t_{id} = \lfloor (1 - Beta_d(\alpha^t, \beta)) | x^t_{i_k d} | \rfloor$;

Step 2 (*the inheritance phase*). *Select* n^t_{id} *number of items from group* $x^t_{i_k d}$ *based on any logic and assign them to the new group* y^t_{id};

Step 3 (*the post-assignment phase*).*Allocate a group* (*either among the existing groups or the new groups*) *to each of the remaining items that have not been selected during the inheritance phase. The allocation of groups to items should guarantee the feasibility of the generated solution;*

Problem-dependent heuristics can be employed at Step 2 and Step 3 of the *NSG* algorithm. With the aim of generating possibly better solutions, items can be selected in a greedy fashion in Step 2, based on any preference defined on their attribute values (e.g., item sizes in bin-packing problem). Here is where we can use the structural knowledge of the problem. Constructive heuristics can be used at Step 3 of *NSG* algorithm. Best-fit and first-fit strategies are often among the simplest and effective constructive heuristics for most of the grouping problems. First-fit strategy always finds the first group that can fit the requirement of the current item. Best-fit strategy differs with first-fit in such a way that it finds the group that best fits the requirement of the current item. Once both of first-it and best-fit strategies found the suitable group, they assign the item to it.

At Step 1 of *NSG* algorithm when for a given value of *d* the value of n^t_{id} becomes zero, it indicates that y^t_{id} inherits no item from $x^t_{i_k d}$. This means that the new group y^t_{id} is not constructed during the inheritance phase and we may obtain a solution with less number of groups than the parent solution, after termination of *NSG* algorithm. This observation is essential for the variable grouping problems with identical groups (e.g., bin-packing problem or graph coloring problem) in which the optimization task is to minimize the number of groups.

If we meet the requirements on the feasibility of the existing or newly opened groups in Step 3 of *NSG* algorithm, the output solution will be always feasible. Supportive of this conclusion is the following lemma which is held (under general definition of groping problems) for each partially constructed group at Step 2 of *NSG* algorithm.

Lemma 1. If $x_{i_k d}^t$ is a feasible group to a grouping problem, then any subset $y_{id}^t \subseteq x_{i_k d}^t$ also forms a feasible group.

The selection process in *GES* can be exactly similar to that of classic *ES*. Similar to the notation used to introduce the multi-member *ES*, we can use $(\mu^+, \lambda) - GES$ notation to introduce the family of *GES* algorithms.

To update the endogenous strategy parameter (α^t) of *GES*, and in a manner similar to the "1/5-success rule", we increase α^t (which results in getting a larger value for n_{id}^t on average) if the observed estimation of the success probability exceeds a given threshold (P_s) during *G* successive iterations, whereas in the opposite case α^t must be decreased.

In our implementation, we use $(1+\lambda) - GES$ family whose steps are detailed as follows. We omit index i_k because of the case of $\mu = 1$. We also use $f(X)$ to denote the objective function or fitness value corresponding to solution *X*.

Algorithm $(1+\lambda) - GES$
Initialize*:* $\beta, \lambda, 0 < a \le 1, \alpha^0 > 0, \alpha^{\min} > 0, G \ge 1, P_s$*;*
Begin
$t \leftarrow 0; G_s \leftarrow 0; \alpha \leftarrow \alpha^0;$
Generate an initial feasible solution X^t *and evaluate it;*
While *stopping criteria are not true*
For *i*=1 *to* λ
Given the parent solution X^t*, apply NSG algorithm to obtain the offspring solution* Y_i^t*;*
End for
Apply the comparison criteria between X^t *and the* λ *generated offspring to select the best individual, which is known as* X^{t+1} (*ties are broken randomly*)*;*
If $f(X^{t+1}) < f(X^t)$
$G_s \leftarrow G_s + 1;$
End if
If $(t \bmod G) = 0$

$$\alpha \leftarrow \begin{cases} \alpha / a & \text{if } G_s / G \ge P_s \\ \max(\alpha_{\min}, a \times \alpha) & \text{if } G_s / G < P_s \end{cases}$$

$G_s \leftarrow 0;$
End if
$t \leftarrow t + 1;$
$\alpha^t \leftarrow \alpha;$
End while
End

5. A brief introduction to *GPSO* algorithm

Following the idea of using group dissimilarity measures for quantifying the degree of difference or dissimilarity between two groups, in place of the differential operation on scalars, Husseinzadeh Kashan *et al* (2013a) proposed the grouping version of the well-known particle swarm optimization algorithm. The *GPSO*'s updating equations which are comparable to the original *PSO* equations have been introduced as follows:

$$v_{id}^{t+1} = wv_{id}^{t} + c_1 r_1 Dissimilarity(p_{id}^{t}, x_{id}^{t}) + c_2 r_2 Dissimilarity(p_{gd}^{t}, x_{id}^{t}) \quad (10)$$

$$Dissimilarity(x_{id}^{t+1}, x_{id}^{t}) \approx v_{id}^{t+1} \quad (11)$$

where $d = 1,...,D_{X_i^t}$ is the group index and *i* is the solution index. x_{id}^{t}, p_{id}^{t} and p_{gd}^{t} stand for the *d*th group in solutions X_i^t (the *i*th feasible grouping solution in the population at iteration *t*), P_i^t (the best feasible grouping solution experienced by the *i*th particle untill iteration *t*) and P_g^t (the global best feasible grouping solution found untill iteration *t*), respectively. *w* is the inertia coefficient, c_1 and c_2 are constant values and r_1 and r_2 are uniformly distributed random variables in [0, 1]. v_{id}^{t} is the velocity value relevant to the *d*th group in X_i^t.

The foundation for generating the new particle solution X_i^{t+1}, is the *NSG* algorithm already described in the previous section. Equation (11) implies that the construction of the new group x_{id}^{t+1}, during the inheritance phase, should be in such a way that its degree of dissimilarity with x_{id}^{t} be around the value of v_{id}^{t+1} computed by (10). However, prior adjustments need to be considered such as using maximum weight bipartite matching (MWM) based ordering rule to pair each group *d* of X_i^t with the most similar group in P_g^t (or P_i^t), when computing *Disssimilarity* terms in (10). For more details on *GPSO* algorithm, its mechanism and implementation issues, the interest reader may refer to Husseinzadeh Kashan *et al* (2013a).

6. Applications on two grouping problems

The performance of *GES* and its rivals, *GGA* and *GPSO*, is tested on two concrete grouping problems namely, the single batch-processing machine scheduling problem with the objective of minimizing makespan and its special case, the bin-packing problem. Although both problems have a same packing structure, our experimental results reveal that the performance of *GGA* significantly defers when shifting from one problem to another one. However, this is not the case with *GES*.

6-1. The single batch-processing machine scheduling problem

A batch-processing machine can process a batch of jobs, as long as the total size of all the jobs in a batch does not violate the machine's capacity. These machines are widely used in burn-in operations in semiconductor manufacturing, where integrated circuits are subjected to thermal stress for an extended period. The burn-in operation constitutes a bottleneck in the final testing operation, and efficient scheduling of this operation to maximize throughput is of great concern to management (see Uzsoy, 1994 and Husseinzadeh Kashan *et al*, 2006). The NP-hard single batch-processing machine scheduling problem (*SBMP*) is defined as follows: There are n jobs to be processed by the machine. The required processing time of job j is denoted by p_j. Each job j has a size s_j and the machine has a capacity B. The sum of the sizes of jobs in a batch must be less than or equal to B. It is assumed that no job has a size exceeding the machine capacity and no job can be split across the batches. The processing time of a batch is given by the longest minimum exposure time among all the products in the batch. Once processing of a batch is initiated, it cannot be interrupted and no job can be added or removed from the batch. Therefore, all jobs in a batch have a common completion time which is equal to the completion time of the batch. The objective is minimizing the makespan(C_{max}), i.e. the sum of the batch processing times. Treating each batch as a group and each job as an item, it can be easily checked that *SBMP* is a grouping problem.

6-2. The bin-packing problem

In the one-dimensional bin-packing problem (*BPP*), a set of n items of different sizes s_j must be packed into a finite number of bins or containers each of capacity B in a way that minimizes the number of bins used. Like *SBMP*, the hard constraint here is that the total size of items inside the bin must not exceed the bin capacity. This strongly NP-hard problem arises explicitly or as a sub-problem in many practical applications, such as filling up containers, loading trucks with weight capacity constraints, creating file backups in media and technology mapping in field-programmable gate array semiconductor chip design.

SBMP can be regarded as a generalized version of *BPP*. When all jobs require identical processing times, *SBMP* reduces to *BPP*. Both of *SBMP* and *BPP* belong to the class of variable grouping problems with identical groups. Although, *BPP* is a special case of *SBMP* and we expect to see a same performance achieved by algorithms on these problems, our experimental results reveal that unlike *GES*, the performance of *GGA* significantly defers when shifting from one problem to another.

7. Issues on the implementation of algorithms

This section gives more details on the algorithmic issues for solving test problem instances of *SBMP* and *BPP*. Let us start by the manner by which n_{id}^t number of jobs/items are selected from batch/bin x_d^t (note that index i_k has been omitted) at Step 2 of *NSG* algorithm. For this purpose, for *SBMP* we choose n_{id}^t number of jobs with the longest processing times from batch x_d^t and for *BPP* we select n_{id}^t number of largest items from bin x_d^t. However, we apply such a selection strategy in a probabilistic manner. That is, in each iteration *t* and for each group *d* a random number is drawn from [0,1]. If it becomes less than a given threshold *R*, selection is done based on the job/item processing times/sizes. Otherwise, job/item selection is done randomly.

To reassign to groups the orphaned jobs/items left over in Step 2 of *NSG* algorithm, the following best-fit decreasing strategies are used for *SBMP* and *BPP* respectively.

Best-fit decreasing strategy for SBMP

Every time among unassigned jobs, select the job with the longest processing time and put it in a feasible batch with the batch processing time longer than processing time of the selected job and with the smallest residual capacity. If there is no such a feasible batch having a longer processing time, put the job in a feasible batch with the longest batch processing time (*break ties in favor of the batch having the smallest residual capacity*). *If the selected job fits in no existing batches, create a new batch and assign it the job.*

Best-fit decreasing strategy for BPP

Every time among unassigned items, select the largest item and put it in a feasible bin with the smallest residual capacity (*break ties in favor of the bin having the lowest index*). *If the selected item fits in no existing bins, create a new bin and assign it the item.*

The above strategies reassign jobs or items based on decreasing order of their processing times/sizes. When the ordering is not important, the strategy is known as "best-fit strategy".

Now we focus on the issues related to the implementation of *GGA*. The procedure of this steady-state order-based genetic algorithm and the recommended values for control parameters can be found in Falkenauer (1994). Let us give the name "*GGA*1" to this algorithm. When applying the mutation operator in *GGA*, based on Falkenauer's statements, only a *small* number of groups are eliminated randomly and their items are reassigned to some of existing or newly opened groups. Quantifying the term "*small*", we consider two levels for the group elimination rate and use a

Bernoulli process (in which the probability of success is equal to the chosen rate) to randomly decide that which groups should be eliminated. As we found, the performance of *GGA* is very sensitive to the chosen rate and as the rate gets larger, generally the performance of *GGA* is enhanced in case of *SBMP*, while it is deteriorated in case of *BPP*. In our experiments we report the performance of *GGA* on two rates, i.e., 0.1 and 0.5. *GGA*1 uses 0.1. In the second implementation of *GGA*, which is called *GGA*2, the number of iterations is set to 1250 instead of 5000 iterations which is used in *GGA*1. Finally in the third implementation of *GGA*, which is called *GGA*3, settings are as follows: the number of iterations is 1250 and the rate of group eliminations is 0.5.

As an influencing factor on the performance of *GGA*, selection between "best-fit" or "best-fit decreasing" strategies is critical in the reinsertion step of the mutation stage (the post assignment phase of *NSG* algorithm). As we perceived, the appropriate strategy for *BPP* which has been adopted by Falkenauer (1994) is the best-fit strategy. But, the proper strategy for *SBMP* is the best-fit decreasing strategy. Unlike in the mutation stage, in the crossover stage, the best-fit decreasing strategy is seems to be the best choice for both *BPP* (as used by Falkenauer (1994) and Falkenauer (1996)) and *SBMP*.

In our implementation of *GGA*, we employed Falkenauer's recommendation on the mutation stage which says the emptiest bin should always be eliminated, and at least three bins should be eliminated. We also adopt the fitness function proposed by Falkenauer (1994) and Falkenauer (1996) for *BPP* for both *GES* and *GGA*. Instead of counting the number of bins used to pack all items, Falkenauer suggests the cost function to maximize the average, over all bins, of the *k*th power of bin efficiency to measure the exploitation of a bin's capacity.

When solving *SBMP* test problems, the stopping criteria for all algorithms are either reaching the maximum number of iterations or finding the solution whose C_{max} value is equal to the lower bound value (i.e., getting the optimal solution). In case of *BPP*, the stopping condition is only reaching the maximum number of iterations.

8. Computational experiments

The aim of this section is to test the performance of *GES* versus *GGA* and *GPSO* on two grouping problems described in the previous section. Although the most effective algorithms for solving *SBMP* or *BPP* test problems are based on hybridization with local search methods, but the search performance of the algorithm will be tangled with the local search performance and this will give less indication on the true performance of the algorithm. For this reason, we abstain to employ any knowledge of the search landscape acquired by a local search method and initialize all algorithms randomly using best-fit strategies.

A series of computational experiments are carried out in which all algorithms are tested on a number of *SBMP* and *BPP* test problem instances. To generate test problem instances of *SBMP*, the capacity of batch-processing machine is set equal to 100 units ($B = 100$). Job processing times are generated from discrete uniform distribution within 1 to 100 ($U(1,100)$). Four categories of test problem instances are considered based on varying the type of job sizes. Problems with small jobs (SM problems) are those in which job sizes are drawn from $U(1,25)$; problems with small to medium jobs (SM-ME problems) are those in which job sizes are drawn from $U(1,50)$; problems with small to relatively large jobs (SM-RL problems) are those in which job sizes are drawn from $U(1,75)$; and problems with small to large jobs (SM-LA problems) are those in which job sizes are drawn from $U(1,99)$. For each of four problem categories, 6 tiers of problems with different number of jobs (n) are generated. The number of jobs in the first and last tier is 50 and 300, respectively; with an increasing size step of 50 jobs for the interior tiers (the number of jobs for the second tier is 100, for the third tire is 150 and so forth) Therefore, 4×6 combinations of problem categories and the number of job tiers are considered. For each combination, 10 random problem instances are generated and the average results are reported (consequently, 240 test problem instances of *SBPM* are solved). It is worth noting that in the scheduling literature, problem instances of *SBMP* with 100 jobs are reasonably large size problems and hence problems with 300 jobs are large-scale.

For bin-packing problem, three sets of benchmark problems are selected from literature. The first set includes 20 problem instances in which item sizes have been drawn from $U(20,100)$ and the bin capacity is 150. There are 120 items in each problem instance. Each problem instance is introduced with the run code U120-v with $v = 0,1,...,19$. This set of problems was introduced by Martello and Toth (1990) and based on Falkenauer's report (Falkenauer, 1996) this set gives the most difficult problem instances for the reduction method of Martello and Toth. The second set of *BPP* problem instances is called triplets (Falkenauer, 1996). These problem instances are differentiated with their especial structure such that in their optimal solution, each bin contains three items; two of which are smaller than the third of the bin capacity, and one is larger. Therefore, it is possible to put three small items or two large ones into a bin; but then the optimum packing will be inevitably missed. The triplets set includes 20 problem instances. There are 120 items in each problem instance and the bin capacity is equal to100. Each problem in this set is initialized with a run code T120-v with $v = 0,1,...,19$. All of U120 and T120 problem instances are available at the OR-library via: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/binpackinfo.html. Finally the third set of problems has been introduced by Scholl *et al*, (1997). There are 10 problem instances in this set each introduced with the run code HARDv with $v = 0,1,...,9$. These problem instances which have been recognized

as the hard problem instances of *BPP* are available on-line at http://www.wiwi.uni-jena.de/Entscheidung/binpp/bin3dat.htm. There are totally 50 test problem instances of BPP in our experiments where each problem instance is solved just once by each algorithm and results are reported.

To solve the problem instances of *SBMP*, the control parameters of *GES* are determined as follows: the maximum iteration is 3000 for SM problems and 2200 otherwise; *G* is 12 for SM problems and 6 otherwise; P_s is 12^{-1} for SM problems and 6^{-1} otherwise; *R* is 0.6 for SM problems, 0.7 for SM-ME problems, 0.8 for SM-RL problems and 0.9 for SM-LA problems; *a* is 0.98; λ is 3; β is 6; α^0 is 8; and $\alpha^{\min} \in [0.5,0.8]$. We found these values suitable through preliminary experiments. We have done the value selection process in a greedy fashion, though there may be other combinations yielding even better result. When solving *BPP* test problem instances, *GES* is more relaxed to the setting of parameters and always do its best. For the sake of consistency, we use the setting used for solving test problem instances of type SM-ME of *SBMP* problem.

All of *GGA*, *GPSO* and *GES* algorithms has been computerized in MATLAB 7.3.0 and executed on a Pentium 4 computer with 3.2 GHz of CPU speed and 1 GB of RAM. Results on problem instances of *SBMP* are tabulated in Tables 2 to 5. Results on *BPP* problem instances are presented in Tables 6 and 7. It is worth mentioning that results reported for *GPSO* algorithm have been adopted directly from Husseinzadeh Kashan *et al*, (2013a). For this reason, the performance of *GPSO* has not been reported in Table 2 since the authors have not investigated the performance of *GPSO* on problems of type SM. Their justification is that the structure of this type of problems is such that in which jobs are highly packable and hence the batch capacity constraints are more relaxed in comparison with other problems which include larger jobs.

In Tables 2 to 5, the performance quality of a given algorithm such as *A* is measured by the *PI* index defined as $PI = AVG\left(\frac{(Z^A - Z^{LB})}{Z^{LB}} \times 100\%\right)$. For a given problem, Z^A denotes the makespan obtained by *A*, Z^{LB} denotes a lower bund of the optimal makespan (see Husseinzadeh Kashan *et al*, (2006) for calculating such a bound) and *AVG*(.) is the averaging operation. It is worth to note that the performance index overestimates the gap between Z^A and Z^{LB}. The true distance between Z^A and the optimal makespan is not greater than its distance with Z^{LB}.

The columns with the title of "*Bins*" in Tables 6 and 7 report the minimum number of bins used by the algorithm to pack all items. Values in the last column indicate either the optimal number of bins or a theoretical lower bound on the optimal number of bins.

>>Insert Tables 2 to 7 about here<<

Among three *GGA* schemes, this is *GGA*3 which shows a better performance on *SBMP* problem instances. In spite of 5000 generations used in *GGA*1, *GGA*3 with just 1250 generations always achieves better results. However, this is not true to say that the number of generations has little impact on the performance of *GGA*. Because, the maximum number of iterations in *GGA*2 is smaller than that of *GGA*1 and this is the only reason that *GGA*2 performs weaker than *GGA*1.

From Tables 2 to 5, it is clear that *GES* is more dependable than both of *GGA* and *GPSO* in providing better quality solutions for *SBMP*. Especially for problems of type SM-LR and SM-LA, *GES* performs better than *GPSO* and all of *GGA* versions on all problems, without regards of the problem size (the number of jobs).These problem categories are expected to be harder than the other two categories, because jobs are poorly packable and many job assignments results in batch capacity violations. Here, *GPSO* performs relatively better than *GGA*1 and *GGA*2. But it is neither superior nor inferior to *GGA*3, and the performance of these two algorithms is rather comparable. For problems of type SM, the average performance of *GES* on the largest problems (with 300 jobs) is similar to *GGA*3 (the best among three *GGA* versions), but for the other problems of this type, *GES* is a little bit superior. This is only for SM-ME problems, that *GGA*3 performs better than *GES* on large problems. However, the scenario is reversed for the smaller problem instances. *GGA*1 achieves a little bit better performance than *GPSO* on SM–ME problems. While *GPSO* is superior over *GGA*2 on SM–ME problems, it is inferior to *GES* and *GGA*3 on all problems of various sizes.

In terms of the required running time, generally for SM and SM-ME problems *GES* takes less time than *GGA*3. However, for the other two categories there is not a significant difference between two algorithms. While *GGA*1 is the most time consuming algorithm, *GGA*2 is the most time saving. The time taken by both *GES* and *GGA*3 lies between theses extremes. The rank of *GPSO* in terms of the spent time is somewhere between *GGA*1 and *GGA*2 for large problem instances. However for small problems, *GPSO* spends less time than all *GGA* algorithms. Although on large scale problems the times taken by *GES* and *GPSO* come close to each other, the execution time of *GPSO* is generally less than *GES* on many problems. However, this is not a critical weakness given the smaller number of solutions generated by *GPSO* in comparison with *GES*, *GGA*2 and *GGA*3. It can be concluded that the time needed to generate a feasible solution in *GPSO* is generally longer than others. The reason may be related to the use of *MWM* ordering rule which requests for solving a maximum weight matching sub-problem, repeatedly. Based on the results, increasing the number of iterations has a positive effect on the performance of *GGA*, because there is no problem for which *GGA*2 be able to provide better result than *GGA*1. However, in comparison with *GES*, *GGA*1 is not

beneficial given the higher computation time and relatively poorer performance. Comparing *GGA*2 and *GGA*3, the time difference is due to the mutant solutions. The more the number of reinsertions (because of the elimination of more groups), the more time is required to get a complete mutant solution.

It is worth noting that all of *GGA*, *GPSO* and *GES* algorithms use a two phase mechanism to generate a new solution in course of search. The time complexity to generate a new solution in *GGA*, *GPSO* and *GES* algorithms is equal to the time required to generate a solution using first/best-fit strategy. Although this time may be problem dependent, for both bin-packing and single batch-processing machine scheduling problems the time complexity of running a first/best-fit strategy is $O(n\log n)$. Therefore, the time complexity of *GGA*, *GPSO* and *GES* algorithms reduces to the time complexity of a typical *GA*, *PSO* and *ES*, respectively. One important observation made on the execution time of the algorithms is that for all algorithms, the required time increases linearly when the problem size increases. This can be inferred from Figure 6. Following the above discussions, we can come at this conclusion that *GES* is the most effective and efficient algorithm among four rivals.

Comparing algorithms on *BPP* problem instances, once again *GES* achieves the best results. On U120 problem instances, *GES* performs very well and finds the optimal packing for all problem instances in a single run (see Table 6). Here, the performance of *GPSO* is also remarkable and optimal. For U120-8 and U120-19, Falkenauer (1996) conjectured that the optimum number of bins is 51 and 50, respectively. However, both of *GES* and *GPSO* find a packing with the optimal number of bins for these two problems which are 50 and 49, respectively. It is therefore remarkable that *GES* and *GPSO* find the optimal packing. However, none of *GGA*1, *GGA*2 and *GGA*3 could find the optimal packing for all problems. *GGA*1 finds the optimal packing for 15 problems (out of 20). For *GGA*2 and *GGA*3 this number is 10 and 0 (out of 20), respectively.

It is worth mentioning that a powerful solver for bin-packing problem which is based on hybridization of *GGA* with the *MTP* reduction method (called *HGGA*), has been applied by Falkenauer (1996) to U120 problem instances where the number of succeeded runs were 18 out of 20. As can be seen, the results obtained by *GGA*1 are similar to those which were obtained by *HGGA* (Falkenauer, 1996), even though *HGGA* uses the *MTP* reduction method.

On T120 set of problem instances, all algorithms exhibit a same non-optimal performance. For each of 20 problem instances of this set, 41 bins are opened by all algorithms, except for *GGA*3. But the optimal packing can be produced with 40 bins (*GGA*3 always opens two more bins, i.e. 42 bins). However, this result does not sound disappointing since many existing algorithms end up with 41 bins.

For HARD set of problem instances, both of *GES* and *GPSO* could reach the best packing, for 8 problem instances (see Table 7). While for *GGA*1, *GGA*2 and *GGA*3 this number is 5, 3 and 0, respectively.

Considering both effectiveness (solutions quality) and efficiency (execution times) measures, one can conclude that *GES* is the superior algorithm. It beats all *GGA* versions in terms of both measures and beats *GPSO* in terms of efficiency measure.

Unlike *SBMP*, when solving *BPP* test problem instances, putting more effort on the mutation stage by *GGA*3 results in more deterioration of performance. While on *SBMP* problem instances, *GGA*3 is superior over *GGA*1 and *GGA*2, on *BPP* problem instances it is inferior. As a special case of *SBMP*, it is expected to see the same performance achieved by algorithms on *BPP* problem instances. While *GES* acts in accordance with this expectation, *GGA* does not follow. Indeed, the amounts of reinsertions during the mutation stage and the type of reinsertion strategy (best-fit or best-fit decreasing) are crucial to *GGA* and shifting from *SBMP* to *BPP* requests for fundamental changes in these factors (best-fit is the proper strategy for *BPP*, but best-fit-decreasing is the proper strategy for *SBMP*).

The following issues can justify that why we believe *GES* is much more flexible than *GGA* or *GPSO* and can be regarded as an effective solver for grouping problems.

- The foundation of *GES* is completely in accordance with the original *ES*. *GES* uses the same set of operators used by *ES* and hence it maintains the potential advantages of *ES*. One of the main advantages is the ease of implementation.
- Similar to *GGA*, *GES* can exploit the structural knowledge along with the problem to generating new solutions of grouping problems. Unlike *GGA, GES* mutation operator is more flexible in selection of items in the inheritance phase and it can exploit the structural pattern that a good solution typically owns. For example, inspired by the best-fit decreasing strategy, bringing jobs with a longer processing time together in the same batches is more reasonable for a feasible solution of *SBMP*. Such an advantage is also along with *GPSO*.
- One of the main advantages of *GES* is that it can be implemented with few operators. However, the implementation of *GGA* may request for problem specific crossover or mutation operators or selection among existing operators of the same type. Such an advantage is also along with *GPSO*.
- When the structure of the problem is in such a way that the number of groups in 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, let us

consider the two-machines parallel-machines scheduling problem with the objective of minimizing makespan or the classic knapsack problem. Every solution of such grouping problems has inevitably two groups. Here, the *GGA* crossover and mutation operators as those described in Section 3 are completely inapplicable because there are only three positions for crossing sections. This deficiency is also visible in our experiments when solving small problem instances of *SBMP* using *GGA*. As reported by Lewis (2006), the performance of *GGA* is worsened on these problem instances 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.

- One main difference between *GGA* and *GES* is in generation of the new solution. The inheritance phase in *GES* is based on sharing a sub-group of items. But the new solution in *GGA* is produced based on sharing the whole-group. In other words, in the crossover or mutation stages of *GGA*, whole of a group is inserted or eliminated. Although the positive or negative consequences of these sharing strategies are not clear for us, an immediate perception is that under whole-group sharing rational, a group with bad configuration may be propagated to the next generations or a group of items forming a good configuration may be lost, soon. However under sub-group sharing rational, the information on group structures are retained partially.
- Although the order of groups is irrelevant in grouping problems, Ülker *et al*, (2007) believe that if majority of group IDs of the items can be maintained for a long period of time, then it is possible to make a low-redundant search by *GGA*. Therefore, the type of group ordering rule can become a critical problem in *GGA* (for example, Ülker *et al*, (2007) have used the cardinality based ordering rule or lowest index based ordering rule for graph coloring problem). Such an ordering problem is observed in *GPSO*, too. Although this difficulty can be mitigated via employing the maximum weight bipartite matching based rule in *GPSO*, it is at the expense of increasing the computation times. Fortunately, there is no need for post ordering of groups in any solution generated during the evolutionary process in *GES*.
- Computational results indicate that the performance of *GGA* is sensitive to the type of grouping problem. As we observed, while both of the *BPP* and *SBMP* problems have a same packing structure and just differ in their objective function definition, the performance of *GGA* significantly differs on these problems. Indeed, the type of reinsertion strategy (best-fit or best-fit decreasing) and the amount of reinsertions (relatively low and relatively high) in the mutation stage are crucial for *GGA*, insomuch as the best setting of these factors for *BPP* problem instances provide the worst results on *SBMP* problem instances. Fortunately, the performance of

GES is constant under best-fit decreasing strategy on both *BPP* and *SBMP*. The amount of reinsertions is also controlled by the self-adaptive nature of *GES*.

- While we can simply apply *GES* and *GPSO* algorithms on the grouping problems with non-identical groups, this is not true for *GGA*. Let us consider the parallel-machines scheduling problem with non identical machines and the objective of minimizing makespan. Since *GGA* performs under whole-group sharing rational, it cannot be easily applied on this scheduling problem. However, under sub-group sharing rational which is followed by *GES*, there is no matter whether groups are identical or non-identical.
- The grouping problems have been defined originally as those in which each item must be assigned to a single group. Though in these problem it is assumed a well-defined group boundaries (let us refer to this type of grouping as *hard grouping*), boundaries between natural groups may be overlapping. It is therefore hard to assign a certain item to a single group, but not to allow it to be partially belonging to other groups. A well-known example in which such a grouping is allowed is the fuzzy data clustering problem (let us refer to this type of grouping as *soft/fuzzy grouping*). Although it is not straightforward to employ *GGA* for soft grouping purposes, *GES* is able to do soft grouping task as well. In a recent paper, Husseinzadeh Kashan *et al*, (2013b) have adapted *GES* for soft/fuzzy grouping purposes and tested the performance of the resultant algorithm, called *FuzzyGES*, on unsupervised fuzzy data clustering.

9. Concluding remarks

Introduced in 1994, the grouping genetic algorithm (*GGA*) is the most established heuristic algorithm for grouping problems which exploits the structural information along with the grouping nature of these problems. Adopting the group encoding, the grouping version of the evolution strategies, called *GES*, has been proposed. *GES* owns a mutation operator which is founded based on a group dissimilarity measure to work with groups rather than items isolatedly. Because, these are the groups that constitute the underlying building-blocks of the solution in a grouping problem. The rational 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 candidate solution in discrete space.

Our extensive computations revealed that *GES* performs almost more effective and efficient than *GGA* and *GPSO* (a recently proposed grouping particle swarm optimization algorithm) on many test problem instances of the single batch-processing machine scheduling problem and various problem instances of the bin-packing problem. Although these problems share exactly a same grouping

structure and the performance of *GES* on both problems was reliable, switching from one problem to another deteriorated the performance of *GGA*. Though such a deficiency was not observed in the performance of *GPSO*, it is still inferior to *GES* on single batch-processing machine scheduling test problems.

GES is very easy for implementation and can be regarded as a promising solver of the wide class of grouping problems. For future research, the effectiveness of *GES* is worth to be examined on other grouping problems such as graph coloring problem, cell formation problem, multiple traveling salesmen problem, etc. Besides, developing the grouping version of the recently introduced real-valued algorithms such as, league championship algorithm (Husseinzadeh Kashan, 2009; Husseinzadeh Kashan, 2011; Husseinzadeh Kashan, 2014) and Optics Inspired Optimization (Husseinzadeh Kashan, 2013) is particularly encouraged.

References

Agustín-Blas L E, Salcedo-Sanz S, Ortiz-García E G, Portilla-Figueras A, Pérez-Bellido Á M (2008). A hybrid grouping genetic algorithm for assigning students to preferred laboratory groups. Expert Systems with Applications, doi:10.1016/j.eswa.2008.09.020.

Agustín-Blas L E, Salcedo-Sanz S, Vidales P, Urueta G, Portilla-Figueras J A (2011). Near optimal citywide WiFi network deployment using a hybrid grouping genetic algorithm. Expert Systems with Applications 38, 9543–9556.

Agustı´n-Blas L E, Salcedo-Sanz S, Jiménez-Fernández S, Carro-Calvo L, Del Ser J, Portilla-Figueras J A (2012). A new grouping genetic algorithm for clustering problems, Expert Systems with Applications, :10.1016/j.eswa.2012.02.149.

Arnold D V, Beyer H G (2003). A comparison of evolution strategies with other direct search methods in the presence of noise. Computational Optimization and Applications, 24, 135-159.

Beyer H G, Schwefel H P (2002). Evolution strategies: A comprehensive introduction. Natural Computing, 1, 3–52.

Brown E C, Ragsdale C T, Carter A E (2004). Formulating the multiple traveling salesperson problem for a grouping genetic algorithm. IIE Annual Conference and Exhibition, 2001-2005.

Brown E C, Sumichrast R T (2001). CF-GGA: A grouping genetic algorithm for the cell formation problem. International Journal of Production Research, 39, 3651-3669.

Brown E C, Vroblefski M (2004). A grouping genetic algorithm for the microcell sectorization problem. Engineering Applications of Artificial Intelligence, 17, 589-598.

Chen Y, Fan Z P, Ma J, Zeng S (2011). A hybrid grouping genetic algorithm for reviewer group construction problem. Expert Systems with Applications 38, 2401–2411.

De Lit P, Falkenauer F, Delchambre A (2000). Grouping genetic algorithms: an efficient method to solve the cell formation problem. Mathematics and Computers in Simulation 51, 257–271.

Dice LR (1945). Measures of the amount of ecologic association between species. Ecology, 26, 297-302.

Erben W (2001). A grouping genetic algorithm for graph colouring and exam timetabling. PATAT 2000, Springer-Verlag, Lecture Notes in Computer Science, 132-156.

Falkenauer E (1994). New representation and operators for GAs applied to grouping problems, Evolutionary Computation, 2, 123-144.

Falkenauer E (1996). A hybrid grouping genetic algorithm for bin packing, Journal of Heuristics, 2, 5-30.

Fukunaga A S (2008). A new grouping genetic algorithm for the multiple knapsack problem. IEEE Congress on Evolutionary Computation, CEC 2008, 2225-2232.

Hu L, Yasuda K (2006). Minimising material handing cost in cell formation with alternative processing routes by grouping genetic algorithm. International Journal of Production Research, 44, 145-179.

Hung C Y, Sumichrast R T, Brown E C (2003). CPG_{EA}: a grouping genetic algorithm for material cutting plan generation. Computers & Industrial Engineering, 44, 651-672.

Husseinzadeh Kashan A (2009). League Championship Algorithm: a new algorithm for numerical function optimization. SoCPaR 2009 IEEE International Conference of Soft Computing and Pattern Recognition, 43-48.

Husseinzadeh Kashan A (2011). An Efficient Algorithm for Constrained Global Optimization and Application to Mechanical Engineering Design: League Championship Algorithm (LCA). Computer-Aided Design, 43, 1769-1792.

Husseinzadeh Kashan A (2013). A new metaheuristic for optimization: optics inspired optimization (OIO). Technical Report, Department of Industrial Engineering, Faculty of Engineering, Tarbiat Modares University.

Husseinzadeh Kashan A (2014). League Championship Algorithm (LCA): A new algorithm for global optimization inspired by sport championships. Applied Soft Computing, 16, 171-200.

Husseinzadeh Kashan A, Karimi B, Jolai F (2006). Effective hybrid genetic algorithm for minimizing makespan on a single-batch-processing machine with non-identical job sizes. International Journal of Production Research, 44, 2337-2360.

Husseinzadeh Kashan A, Husseinzadeh Kashan M, Karimiyan S (2013a). A particle swarm optimizer for grouping problems. Information Sciences, 252-81-95.

Husseinzadeh Kashan A, Jenabi M, Husseinzadeh Kashan M (2009). A new solution approach for grouping problems based on evolution strategies. SoCPaR 2009 IEEE International Conference of Soft Computing and Pattern Recognition, 88-93.

Husseinzadeh Kashan A, Rezaee B, Karimiyan S (2013b). An efficient approach for unsupervised fuzzy clustering based on grouping evolution strategy. Pattern Recognition, 46, 1240-1254.

James T L, Brown E C, Keeling K B (2007). A hybrid grouping genetic algorithm for the cell formation problem. Computers & Operations Research, 34, 2059–2079.

James T, Vroblefski M, Nottingham Q (2007). A hybrid grouping genetic algorithm for the registration area planning problem. Computer Communications, 30, 2180-2190.

Kreng V B, Lee T P (2004). Modular product design with grouping genetic algorithm: a case study. Computers & Industrial Engineering, 46, 443-460.

Kulczynski S (1927): Die Panzenassoziationen der Pieninen. Bulletin International de l'Academie Polonaise des Sciences et des Lettres, Classe des Sciences Mathematiques et Naturelles, B, 57-203.

Lewis R (2006). Metaheuristics for university course timetabling. Ph.D thesis, Napier University.

Lewis R (2009). A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. Computers & Operations Research, 36, 2295-2310.

Lewis R, Paechter B (2007). Finding feasible timetables using group based operators. IEEE Transactions on Evolutionary Computation, 11, 397-413.

Li P Y, Wang C F, Mao Y S (2006). Hybrid grouping genetic algorithm for one-dimensional cutting stock problem. Journal of Shanghai Jiaotong University, 40, 1015-1023.

Martello S, Toth P (1990). Lower bounds and reduction procedures for the bin packing problem. Discrete Applied Mathematics, 22, 59-70.

Pankratz G (2005). A grouping genetic algorithm for the pickup and delivery Problem with time windows. OR Spectrum 27, 21–41.

Rechenberg I (1964). Cybernetic solution path of an experimental problem. Royal aircraft Establishment, Library Transaction 1122, Farnborough.

Rechenberg I (1973). Evolutionsstrategie: Optimierung technischer Systeme nach den Prinzipien der biologischen Evolution, Frommann-Holzboog: Stuttgart.

Rekiek B, De Lit P, Pellichero F, L'Eglise T, Fouda P, Falkenauer E, Delchambre A (2001). A multiple objective grouping genetic algorithm for assembly line design. Journal of Intelligent Manufacturing, 12, 467-485.

Rekiek B, Delchambre A, Aziz Saleh H (2006). Handicapped person transportation: an application of the grouping genetic algorithm. Engineering Applications of Artificial Intelligence, 19, 511-520.

Scholl A, Klein R, Jürgens C (1997). BISON: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Computers & Operations Research, 24, 627–645.

Singh A, Baghel A S (2007). A new grouping genetic algorithm for the quadratic multiple knapsack problem. Evolutionary Computation in Combinatorial Optimization: 7th European Conference, EvoCOP 2007, Springer-Verlag, Lecture Notes in Computer Science, 210-218.

Singh A, Baghel A S (2009). A new grouping genetic algorithm approach to the multiple traveling salesperson problem. Soft Computing, 13, 95-101.

Sørensen T (1948). A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. Kongelige Danske Videnskabernes Selskab, 5, 1-34.

Ülker Ö, Özcan E, Korkmaz E E (2007). Linear linkage encoding in grouping problems: Applications on graph coloring and timetabling. PATAT06, Springer-Verlag, Lecture Notes in Computer Science, 347-363.

Ülker Ö, Özcan E, Korkmaz E E (2008). A grouping genetic algorithm using linear linkage encoding for bin packing. In Parallel Problem Solving from Nature PPSN X, LNCS 5119, G. Rudolph *et al*, Eds. Springer-Verlag Berlin Heidelberg, 1140 – 1149.

Uzsoy R (1994). A single batch processing machine with non-identical job sizes. International Journal of Production Research, 32, 1615-1635.

Vroblefski M, Brown E C (2006). A grouping genetic algorithm for registration area planning. Omega, 34, 220-230.

Vin E, De Lit P, Delchambre A (2005). A multiple-objective grouping genetic algorithm for the cell formation problem with alternative routings. Journal of Intelligent Manufacturing, 16, 189-205.

Yasuda K, Hu L, Yin Y (2005). A grouping genetic algorithm for the multi-objective cell formation problem. International Journal of Production Research, 43, 220-230.

Table 1. A list of the problems where *GGA* has been applied on them.

Problems	*Related references*
Bin packing problem	Falkenauer (1994,1996);
Economy of scale problem	Falkenauer (1994);
Assembly line design problem	Rekiek *et al*, (2001);
Graph coloring problem	Erben (2001); Ülker *et al*, (2007);
Pickup and delivery problem	Pankratz (2005);
Cell formation problem	De Lit *et al*, (2000); James *et al*, (2007a) ; Yasuda *et al*, (2005) ; Vin *et al*, (2005) ; Brown and Sumichrast (2001) ; Hu and Yasuda (2006) ;
Multiple traveling salesperson problem	Brown *et al*, (2004); Singh and Baghel (2009).
Registration area planning problem	James *et al*, (2007b); Vroblefski and Brown (2006);
Handicapped person transportation problem	Rekiek *et al*, (2006);
Microcell sectorization problem	Brown and Vroblefski (2004);
Modular product design problem	Kreng and Lee (2004);
Material cutting plan problem	Hung *et al*, (2003);
Multiple knapsack problem	Singh and Baghel (2007); Fukunaga (2008).
One-dimensional cutting stock problem	Li *et al*, (2006);
Timetabling problems	Erben (2001); Lewis and Paechter (2007); Ülker *et al*, (2007); Agustín-Blas *et al*, (2008);
Clustering problem	Agustín-Blas *et al*, (2012);
WiFi network deployment problem	Agustín-Blas *et al*, (2011);
Reviewer group construction problem	Chen *et al*, (2011);

Table 2. Results for the problems of type SM

Number of jobs (*n*)	*GES*		*GGA*1		*GGA*2		*GGA*3	
	PI (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)
50	0.34	18.7	0.57	111.2	0.63	24.3	0.44	29.0
100	0.13	39.0	0.42	156.3	0.57	47.4	0.27	56.8
150	0.15	50.6	0.43	334.1	0.61	91.8	0.40	121.1
200	0.26	131.4	0.50	412.0	0.58	110.6	0.32	155.4
250	0.14	182.1	0.29	496.9	0.66	130.9	0.18	193.2
300	0.17	244.9	0.27	610.4	0.46	155.6	0.18	229.7
Average	0.19		0.41		0.58		0.29	

Table 3. Results for the problems of type SM-ME

Number of jobs (*n*)	*GES*		*GGA*1		*GGA*2		*GGA*3		*GPSO*	
	PI (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)
50	2.02	47.0	2.76	194.5	3.01	48.9	2.48	61.3	2.53	29.3
100	1.50	89.0	1.92	298.6	2.96	75.5	1.91	107.5	2.01	58.8
150	1.51	134.4	1.95	422.3	2.07	105.9	1.56	151.7	1.88	95.1
200	1.39	182.6	1.70	554.7	2.30	139.7	1.34	204.4	1.90	135.9
250	1.43	228.6	1.62	709.8	2.02	178.3	1.28	256.0	1.70	182.6
300	1.44	270.8	1.38	845.3	1.84	212.0	1.17	308.5	1.67	228.1
Average	1.54		1.88		2.36		1.62		1.97	

Table 4. Results for the problems of type SM-RL

Number of jobs (*n*)	*GES*		*GGA*1		*GGA*2		*GGA*3		*GPSO*	
	PI (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)
50	6.71	62.4	7.75	221.7	8.38	55.4	7.39	68.8	7.33	39.95
100	5.09	122.3	6.11	367.9	6.31	92.6	5.78	123.2	5.69	82.98
150	4.79	185.8	6.53	545.0	7.51	135.9	5.13	181.7	5.20	142.0
200	4.54	246.5	5.85	717.8	6.76	183.0	4.91	242.6	5.03	199.1
250	3.76	309.7	4.28	900.0	4.65	225.7	3.85	299.9	4.52	278.6
300	3.46	374.5	4.71	1118.5	5.61	279.6	3.76	374.2	4.52	363.0
Average	4.72		5.87		6.53		5.13		5.38	

Table 5. Results for the problems of type SM-LA

Number of jobs (*n*)	*GES*		*GGA*1		*GGA*2		*GGA*3		*GPSO*	
	PI (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)	*PI* (%)	*Time* (Sec)
50	9.56	72.5	10.08	252.1	9.90	62.9	9.80	75.9	9.56	47.8
100	7.62	142.5	8.25	444.0	8.64	111.1	7.96	138.6	8.10	103.5
150	7.78	216.6	8.51	655.4	8.72	164.2	7.99	208.4	7.93	180.2
200	6.67	289.0	7.74	873.5	8.06	223.6	7.10	280.9	7.08	261.2
250	6.59	367.8	7.36	1127.9	7.56	283.6	7.10	359.1	6.99	369.4
300	5.61	444.3	6.12	1390.1	6.53	347.3	5.92	440.5	5.93	486.9
Average	7.30		8.01		8.23		7.64		7.59	

Table 6. Results on U120 test problem instances

	GES		*GGA*1		*GGA*2		*GGA*3		*GPSO*		*min num of bins*
Problem	*Bins*	*Time* (Sec)	*Bins*	*Time* (Sec)	*Bins*	*Time* (Sec)	*Bins*	*Time* (Sec)	*Bins*	*Time* (Sec)	
U120-0	48	82.5	48	391.3	48	92.2	49	99.2	48	126.8	48
U120-1	49	83.7	49	391.7	49	100.9	50	100.3	49	126.6	49
U120-2	46	79.3	46	375.3	46	91.1	47	95.7	46	119.8	46
U120-3	49	84.1	49	394.9	50	96.6	50	98.4	49	128.7	49
U120-4	50	85.3	50	395.9	50	97.6	51	100.6	50	129.2	50
U120-5	48	82.7	48	393.2	48	98.6	49	96.7	48	123.9	48
U120-6	48	82.3	48	389.1	49	94.5	49	105.7	48	124.3	48
U120-7	49	84.4	50	389.9	50	94.1	50	101.8	49	126.3	49
U120-8	50	85.9	51	402.3	51	97.1	51	97.1	50	129.9	50
U120-9	46	85.9	47	369.2	47	94.2	47	96.5	46	120.1	46
U120-10	52	89.5	52	409.1	53	98.1	53	102.2	52	133.9	52
U120-11	49	84.1	49	397.2	49	95.0	50	100.0	49	126.7	49
U120-12	48	83.0	49	390.4	49	98.9	49	100.4	48	125.3	48
U120-13	49	83.0	49	388.5	49	97.4	50	98.6	49	126.1	49
U120-14	50	84.9	50	395.4	50	98.6	51	99.8	50	129.8	50
U120-15	48	82.5	48	399.5	49	99.2	49	102.4	48	125.4	48
U120-16	52	88.4	52	396.8	52	101.6	53	100.6	52	135.4	52
U120-17	52	89.9	52	411.1	53	98.4	53	106.2	52	134.2	52
U120-18	49	84.2	49	391.5	49	96.6	50	98.3	49	126.6	49
U120-19	49	85.2	50	404.5	50	94.8	51	100.5	49	131.4	49
Average	49.05	84.5	49.3	393.8	49.55	96.8	50.1	100.0	49.05	127.5	49.05

Table 7. Results on HARD test problem instances

	GES		*GGA*1		*GGA*2		*GGA*3		*GPSO*		*min num of bins*
Problem	*Bins*	*Time* (Sec)	*Bins*	*Time* (Sec)	*Bins*	*Time* (Sec)	*Bins*	*Time* (Sec)	*Bins*	*Time* (Sec)	
HARD0	56	103.7	56	517.4	56	117.8	58	115.7	56	168.5	56
HARD1	57	110.0	57	473.4	57	122.3	58	121.1	57	167.3	57
HARD2	57	105.8	57	446.1	57	121.3	59	128.0	57	165.3	56
HARD3	56	102.9	56	432.3	57	118.1	58	142.1	56	163.4	55
HARD4	57	110.5	58	452.9	58	121.2	59	141.4	57	167.4	57
HARD5	56	105.1	57	483.8	57	115.6	58	144.7	56	162.5	56
HARD6	57	104.0	57	440.4	58	126.5	59	136.1	57	165.5	57
HARD7	55	107.4	55	431.2	56	114.5	57	129.0	55	151.0	55
HARD8	57	106.2	57	465.7	58	117.7	59	134.8	57	160.3	57
HARD9	56	102.9	57	485.3	57	115.5	58	137.2	56	171.8	56
Average	56.4	105.8	56.7	462.8	57.1	119.0	58.3	133.0	56.4	164.3	56.2

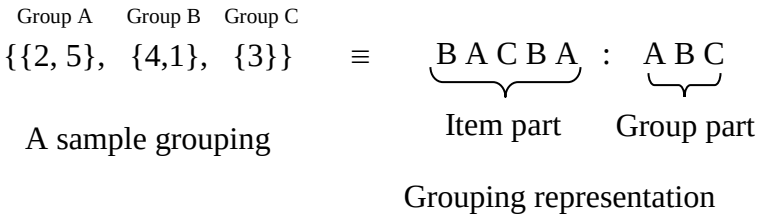
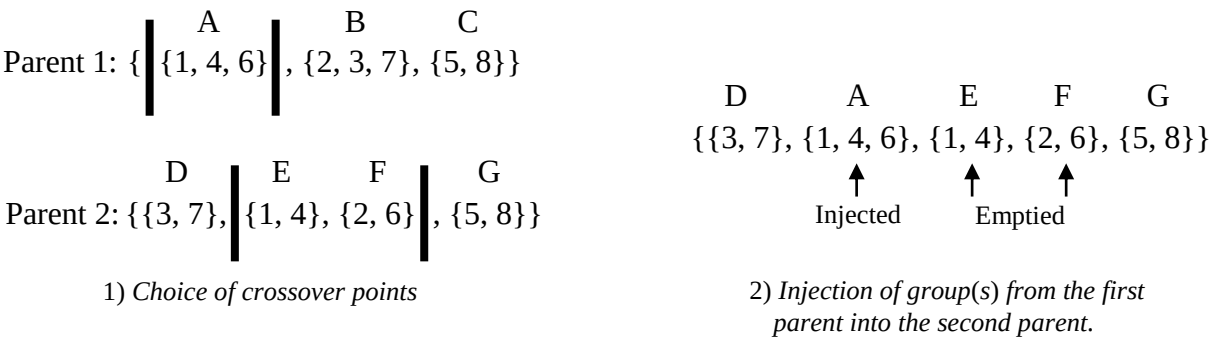
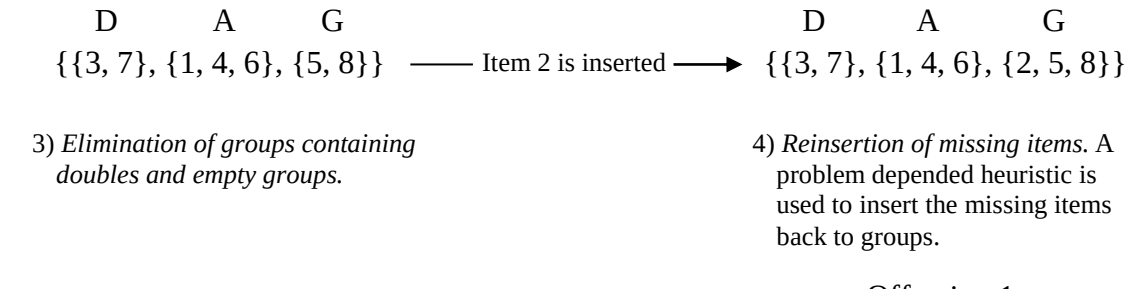


Figure 1. A sample grouping and its relevant group encoding.





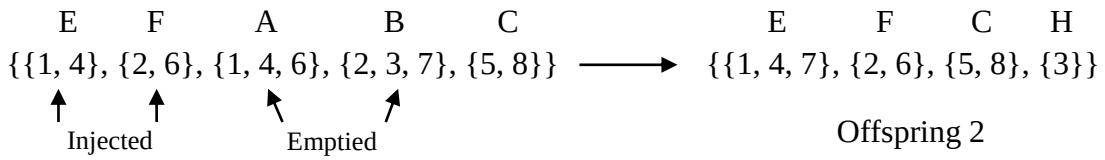


Figure 2. The *GGA* crossover

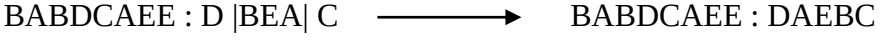


Figure 3. The *GGA* inversion

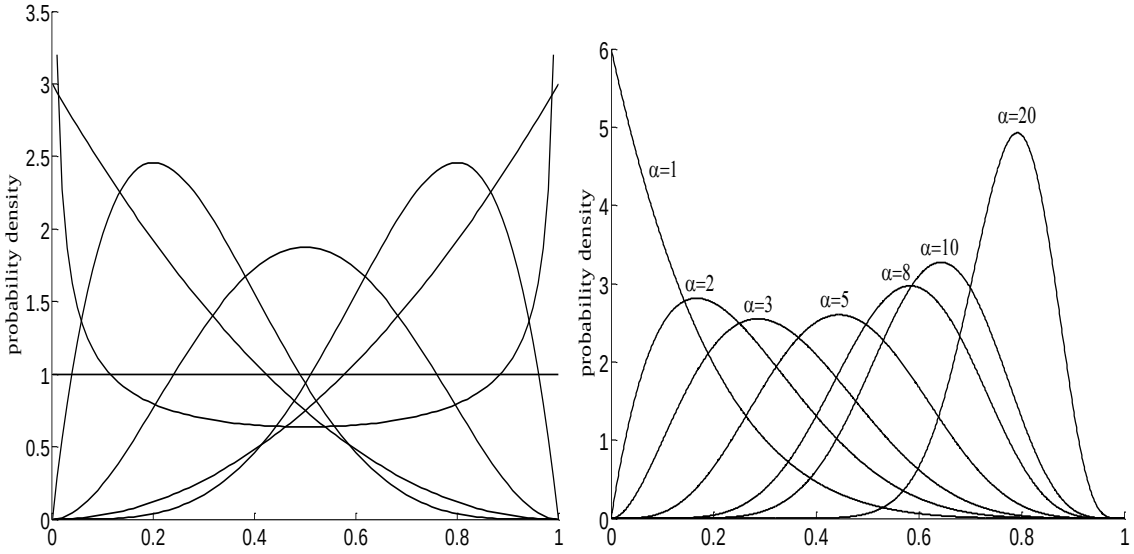


Figure 4. a) Examples of Beta PDF for various values of the shape parameters.

b) Examples of Beta PDF for various values of α at the fixed level of β=6.

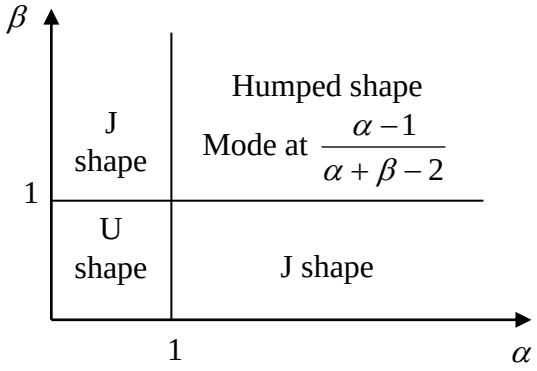


Figure 5. Shape of the Beta distribution as a function of α and β

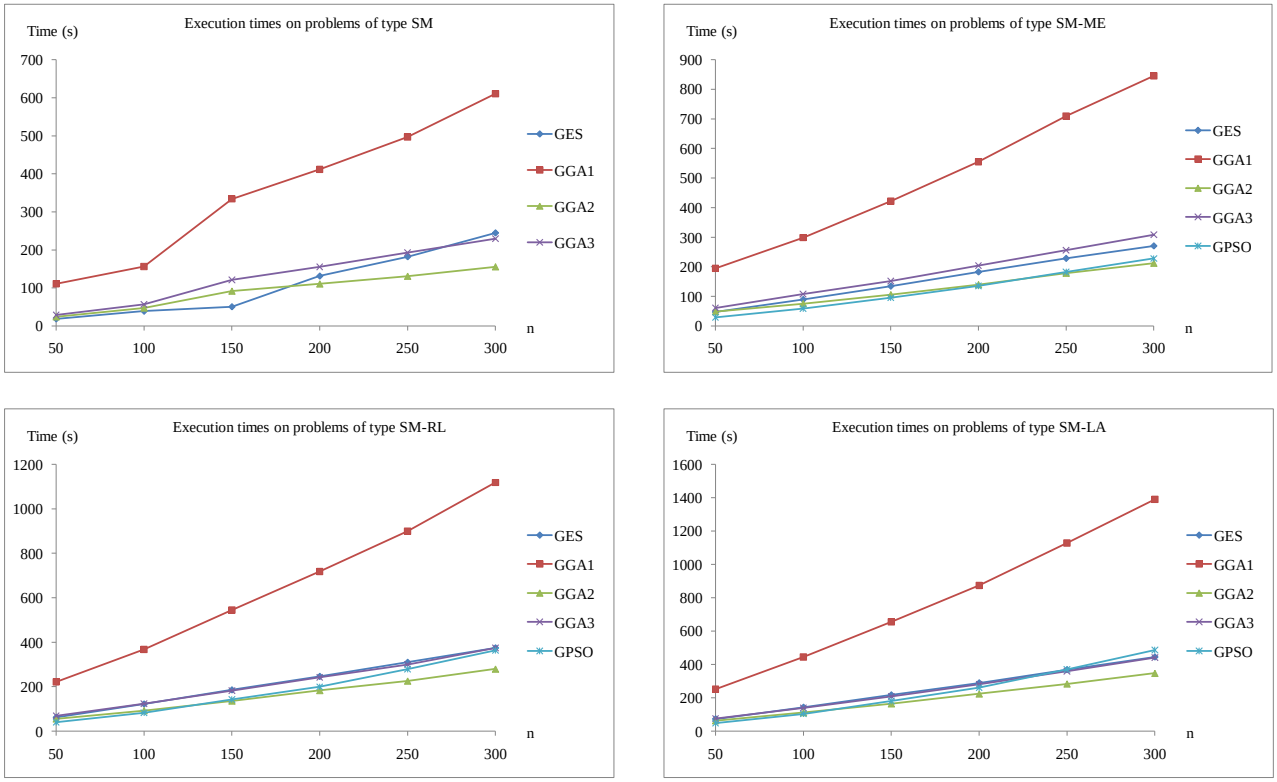


Figure 6. Plot of execution times on different *SBMP* problem categories.