The two-stage recombination operator and its application to the multiobjective 0/1 knapsack problem: a comparative study

Brahim AGHEZZAF and Mohamed NAIMI

Laboratoire d'Informatique et d'Aide à la Décision
Département de Mathématiques & Informatique
Université Hassan II-Aïn Chock, Faculté des Sciences
B.P. 5366 Maârif, Casablanca, Morocco
Email: b.aghezzaf@fsac.ac.ma; naimi_simo11@yahoo.fr

Abstract. In this paper, we propose a new recombination operator and test its performance in the context of the multiobjective 0/1 knapsack problem (MOKP). The proposed recombination operator generates only one offspring solution from a selected pair of parents according to the two following stages. In the first stage, called genetic shared-information stage or similarity-preserving stage, the generated offspring inherits all parent similar genes (i.e.; genes or decision variables having the same positions and the same values in both parents). In the second stage, called problem fitness-information stage, the parent non-similar genes (i.e.; genes or decision variables having the same positions but different values regarding the two parents) are selected from one of the two parents using some fitness information. In fact, we propose three different approaches (i.e.; versions) for the second stage. The first approach (i.e.; the general approach) can be applied to any multiobjective combinatorial problem and can be summarized as follows: each parent non-similar gene is selected from the first parent (respectively the second parent) according to a probability which depends on some fitness function (e.g., the linear scalarizing function) evaluating the fitness of the first parent (respectively the second parent). The second approach (i.e.; the restricted approach) can only be applied to the multiobjective combinatorial problems for which each gene has its own fitness called the gene's fitness and can be summarized as follows: each parent non-similar gene is selected from the first parent (respectively the second parent) according to a probability which depends on the gene's fitness value induced by the value of this gene in the first parent (respectively the second parent). The third approach (i.e.; the MOKP-specific approach) is specific to the MOKP and can be summarized as follows: the parent non-common items (i.e.; items included in one and only one parent solution regarding the current pair of parents) undergo a greedy insertion heuristic which gives priority to the best fitting parent non-common items (i.e.; those having a high quality and ensuring the offspring feasibility regarding the MOKP structure). The two-stage recombination operator is compared against three traditional crossovers (in fact, the one-point, the two-point and the uniform crossovers) using two well-known multiobjective evolutionary algorithms (MOEAs); namely SPEA and NSGA-II. The experimental results show that all the three versions of the proposed recombination operator (i.e.; the general version, the restricted version and the MOKP-specific version) significantly outperform both the one-point and the two-point crossovers in terms of convergence. The convergence performance of the general version (respectively the restricted version) in comparison with the uniform crossover is however sensitive to the number of generations (respectively the MOKP instance and the MOEA context), whereas the MOKP-specific version always outperforms the uniform crossover. As for the diversity performance, both the restricted and the MOKP-specific versions outperform all the three traditional crossovers. However, the diversity performance of the general version is essentially sensitive to the MOEA context.

Key words: Multiobjective evolutionary algorithms (MOEAs), crossover operators, multiobjective combinatorial optimization (MOCO), multiobjective 0/1 knapsack problem (MOKP).

1 Introduction and motivation

The crossover operator is often considered as the primary search operator (i.e.; responsible for creating new solutions from old ones) for many genetic-based evolutionary algorithms [1]. The n-point crossover [2] and the uniform crossover [3] are the most frequently used recombination operators in the literature since they can be easily adapted to many optimization problems (e.g., those whose solutions are binary-coded). Spears [4] has thoroughly analyzed these two recombination methods in the context of a single evolutionary algorithm and has exhibited some typical features (disruptive effect, constructive potential, exploratory power…) characterising each one of the considered crossover operators. Nevertheless, the Spears' study resulted in no absolute good performance of any of the studied crossover operators with respect to another. Indeed, in addition to the recombination scheme itself, the so-called crossover performance may be affected by other factors like population size, population diversity, selection operator and the search space characteristics (i.e.; the specific structure of the problem to be solved) [4,5].

The general principle of the above-mentioned crossover methods (i.e.; the n-point crossover and the uniform crossover) can be summarized as follows. Two parent solutions are selected from the current population and some of their corresponding substrings are exchanged in a randomized stochastic way to create two new offspring solutions. That is, these crossover operators do not take into account any fitness information during the whole recombination process. One can say that this should not be quite necessary to produce the so-called good offspring solutions since the parent solutions have successfully passed the selection procedure in which some fitness information have been certainly used. In this way, the selected parent solutions are expected to contain good partial gene combinations and hence the resulting offspring solutions have a rather good chance to be so good or better than their parents even if this may not occur in every crossover operation. One can also say that if some of the so-called bad offspring solutions are generated, they will get eliminated in next generations by means of the selection operator [1]. However, a careful thought about the latter remark lead to another remark that is: the so-called purely stochastic behaviour of both the n-point crossover and the uniform crossover (i.e.; the exchange of genes between the parents does not undergo any fitness evaluation rule but it is performed in a randomized stochastic way) can sometimes increase the rate of bad solutions within the population especially in early generations so that the selection procedure should take more time (i.e.; more generations) to eliminate them. Moreover, even if we consider the case in which those crossover operators generate good offspring solutions, there is no guarantee that such generated solutions effectively inherit the majority of their parent good attributes because, once again, of the above-mentioned purely stochastic behaviour. So, this is can induce a so-called weak improvement of solutions' quality from a generation to another. Therefore, a repeatedly occurrence of such phenomena during the search process is likely to slow down and even not to efficiently favour the convergence performance of the applied algorithm. Furthermore, a number of empirical studies have shown the beneficial effect of exploiting the particular structure of the problem to be solved while using a genetic-based evolutionary algorithm [6-8].

Besides, still considering the n-point crossover and the uniform crossover, one can easily check that both of them preserve all parent similarities while generating new offspring. In fact, such a so-called similarity-preserving principle should be promising and even natural since the best individuals (i.e.; those having more chance to be selected for taking part in the recombination operations) often have several similar common properties [1,9]. Indeed, for many multiobjective combinatorial problems and especially for the MOKP, the corresponding good solutions being close in the objective space are expected to be rather similar in the decision space [10,11]. That is, the similarity-preserving principle allows the crossover operator to maintain the so-called good gene combinations which can be constructed during the search process.

Durand [12] has proposed a crossover operator especially adapted to the single objective problems having a partially separable objective function (i.e.; a function for which each decision variable x_k assigns a local fitness F_k that isolates its contribution in the objective function). One can also see [13] for more details about the partially separable functions in a single optimization context. This crossover operator can be summarized as follows. First a parameter Δ is specified. Then each offspring decision variable x_k is chosen from one of the two parents P1 and P2 according to the following cases: if $|F_k(\mathrm{P1}) - F_k(\mathrm{P2})| > \Delta$, x_k is selected from the parent having the best value of F_k, otherwise x_k is randomly selected either from P1 or from P2. Durand has applied its crossover operator to some air traffic problems [14]. However, this crossover operator is limited to single problems with partially separable objective functions and it is Δ-dependent since the user has to adjust the value of the parameter Δ. So there is an additional parameter to be appropriately specified whenever we use such a crossover operator.

The above-mentioned remarks motivate us to propose a new recombination scheme called the two-stage recombination. The proposed recombination scheme generates only one offspring solution from two given parent solutions according to the following two successive stages. In the first stage (i.e.; the genetic shared-information stage or the similarity-preserving stage), each parent similar gene is placed into the offspring string with respect to its position. In the second stage (i.e.; the problem fitness-information stage), each parent non-similar gene is selected from one of the two parent solutions according to some fitness information. Of course, we must take into account the problem structure so that can help us to conceive appropriately the selection procedure regarding the parent non-similar genes. That is, we firstly distinguish two cases concerning the second stage. If the problem allows a sort of fitness evaluation of each gene; i.e.; each gene has its own fitness whose value changes with the value of this gene, we first evaluate the fitness of each parent non-similar gene according to the two values of this gene in both parent solutions. Then we select each parent non-similar gene either from the first parent or from the second parent with a probability that is proportional to its fitness evaluation regarding its value in the corresponding parent solution (i.e.; either the first parent or the second parent). This first case corresponds to what we call the restricted version of the two-stage recombination operator. Otherwise, the two parent solutions are first evaluated according to some fitness function (e.g., the linear scalarizing function in the case of the MOKP) and then each parent non-similar gene is selected either from the first parent or from the second parent with a probability that is proportional to the fitness evaluation of the corresponding parent solution (i.e.; either the first parent or the second parent). The latter case is more general than the former case and hence it corresponds to what we call the general version of the two-stage recombination operator.

On the other hand, the MOKP is an interesting multiobjective combinatorial NP-complete problem [15] which can be used to model many real-life applications (capital budgeting, project selection, resource allocation, loading problems …). It can also be found as a sub-problem of many other more general applications. In the last decade, the MOKP have been frequently used as test problem for several state-of-the-art multiobjective metaheuristics among which some multiobjective evolutionary algorithms (MOEAs) (see [1,16] for extensive details about MOEAs in general) and some multiobjective memetic algorithms (i.e.; hybrid MOEAs where a MOEA is hybridized with a local search algorithm). Thus, Zitzler and Thiele [17] have proposed a MOEA called SPEA which was compared to other multiobjective metaheuristics in the MOKP context. Knowles and Corne [18] have proposed a hybrid MOEA called M-PAES with also a comparative study focused on the MOKP. Jaszkiewicz [19] has also proposed a hybrid MOEA called MOGLS whose performance was evaluated in the MOKP context. More recently, Alves and Almeida [20] have suggested a MOEA called MOTGA which is more specific to the MOKP. One can see for example the Coello site [21] for an extensive bibliography about other MOEAs which have been proposed in the literature.

Besides, the particular structure of the MOKP allows us to apply the restricted version of the above-proposed two-stage recombination. Furthermore, by further exploiting the MOKP specific structure, we develop another version of the two-stage recombination operator called the MOKP-specific version which can be summarized as follows. First all parent common items (i.e.; items which are simultaneously loaded in both the current parent instances) are inserted into the current offspring instance: the first stage. Then the parent non-common items (i.e.; items which are loaded in one and only one parent instance regarding the current pair of parent instances) are classified on the base of a so-called item's fitness so that iteratively allows the insertion of the best candidate parent non-common item which ensures the feasibility of the current offspring instance: the second stage. That is, in our early experiments we have first included the proposed two-stage recombination within a simple multiobjective genetic algorithm [22] and then we have included it within a simple multiobjective hybrid genetic algorithm [23]. The obtained results were promising and hence they have encouraged us to further improve the proposed recombination operator and also to extensively test its performance in the context of the MOKP.

The main purpose of this paper is thus to demonstrate through experiments how the inclusion of a recombination operator exploiting some problem fitness information (in fact, the two-stage recombination) into a given MOEA can, not only improves the convergence performance of this MOEA, but also its diversity performance. To bear out this claim, we tested each of the above-proposed versions of the two-stage recombination operator against other more traditional crossover operators (in fact, the one-point crossover, the two-point crossover and the uniform crossover) using two state-of-the-art MOEAs: SPEA [17] and NSGA-II [24]. It should be noted that we did not use any multiobjective memetic algorithm in our experiments since such an algorithm includes a sort of local search operator which can hardly alter the real effect of the recombination operator.

In this section the interest and motivation of the proposed work have been provided whereas the remainder of this paper is organized as follows. Section 2 provides some basic definitions in multiobjective combinatorial optimization and the formulation of the test problem tackled in this paper (in fact, the MOKP). In section 3, we formally describe the three versions of the proposed recombination operator. Section 4 contains an overview of test algorithms (SPEA and NSGA-II). The performance measures for evaluation and comparison are briefly presented in section 5. Section 6 is devoted to the design of computational experiments. The experimental results are provided and discussed in section 7. In the last section conclusions and future work are given.

2 Basic definitions and test problem

2.1 Basic definitions in multiobjective combinatorial optimization

The general Multiple Objective Combinatorial Optimization (MOCO) [25] (one can also see [26] for a survey) problem can be stated as follows:

$$\text{(MOCO)}\begin{cases}\text{Maximize } f(x) = z = (f_1(x) = z_1,\ldots,f_m(x) = z_m)\\ \text{subject to } x \in S\end{cases}$$

where S is a discrete subset of R^n defining the *decision space* or the set of *feasible solutions,* and $f_1(x),\ldots,f_m(x)$ are m ($m \geq 2$) objective functions defined on S. Let $z = f(x)$ be the objective vector associated with the solution x, the space $Z = \{z = f(x) \mid x \in S\}$ is called the *objective space*. An objective vector $z = f(x)$ is called a *point* of Z.

- Point $z^1 = f(x^1)$ *dominates* point $z^2 = f(x^2)$, denoted by $z^1 \succ z^2$, if $z^1_j \geq z^2_j$ $\forall j \in \{1,\ldots,m\}$ and there exists at least one $k \in \{1,\ldots,m\}$ such that $z^1_k > z^2_k$. Solution x^1 *dominates* solution x^2 if z^1 *dominates* z^2.
- A solution $x \in S$ is said to be *Pareto-optimal* (*efficient*) if there is no $x^* \in S$ that dominates x. In this case, point $z = f(x)$ is called *nondominated*. The set of all Pareto-optimal solutions is called the *Pareto-optimal set*. The image of the Pareto-optimal set in the objective space is called the *Pareto front*.
- An *approximation to the Pareto front* is a set A of points (and corresponding solutions) such that there is no pair of points $z^1, z^2 \in A$ satisfying $z^1 \succ z^2$; i.e.; set A is composed of mutually nondominated points.

2.2 Test problem: the MOKP

The MOKP is a particular case of MOCO problems. We are interested in the same version as the one presented in Zitzler and Thiele [17]: the MOKP with m knapsacks (i.e.; m objectives and m constraints) and n items. According to this version, the MOKP can be written as follows:

$$\text{(MOKP)}\begin{cases}Max\ \ f_i(x) = \sum_{j=1}^{n} p_{ij}x_j\ ,\ i = 1,\ldots,\ m\\ \sum_{j=1}^{n} w_{ij}x_j \leq c_i\ ,\ i = 1,\ldots,\ m\\ x_j \in \{0,1\},\ \ j = 1,\ldots,\ n\end{cases}$$

where:

- $x = (x_1,\ldots,x_n)$ is an n-dimensional binary vector defining a MOKP solution such that: $x_j = 1$ if item j is loaded in all knapsacks and $x_j = 0$ otherwise.
- p_{ij} is the profit of item j according to knapsack i.
- w_{ij} is the weight of item j according to knapsack i.
- c_i is the capacity of knapsack i.

3 The two-stage recombination operator

3.1 General description

Given two parent solutions denoted by P1 and P2, the two-stage recombination operator generates only one single offspring C from P1 and P2 through two successive stages. According to the structure of the problem to be solved, we first propose two versions of the two-stage recombination operator. It should be noted that we will propose a third version in subsection 3.2 while applying the proposed recombination operator to the MOKP. Let us begin with some useful notations:

- P1 = (x_1(P1),…, x_n(P1)).
- P2 = (x_1(P2),…, x_n(P2)).
- C = (x_1(C),…, x_n(C)).

3.1.1 The general version

This general version of the two-stage recombination operator can be applied to any MOCO problem, so we give below its full description in a general way (i.e.; without specifying the problem to be solved).

a) **First stage:** the genetic shared-information stage or the similarity-preserving stage in which C inherit all genetic information shared by P1 and P2. Thereby, first all parent similar genes (i.e.; genes that are similar in both P1 and P2 or decision variables having the same positions and the same values in both P1 and P2) are identified and then fitted into C. So, for each parent similar gene k (i.e.; each decision variable x_k such that $x_k(\text{P1}) = x_k(\text{P2})$), we set $x_k(\text{C}) = x_k(\text{P1}) = x_k(\text{P2})$.

b) **Second stage:** the problem fitness-information stage which concerns the parent non-similar genes (i.e.; genes that are not similar regarding P1 and P2 or decision variables having the same positions but not having the same values regarding P1 and P2). This stage should use some fitness information while selecting the parent non-similar genes to be ultimately fitted into C. To satisfy this goal for any structure of MOCO problems, we suggest the following procedure. We first evaluate the quality of P1 and P2 through the use of some fitness function F (for example a utility function based on the linear scalarizing function whose weights are randomly generated in every recombination operation). Then, the probability of selecting any parent non-similar gene from P1 (respectively P2) is conceived in such a way to be proportional to $F(\text{P1})$ (respectively $F(\text{P2})$). So, for any parent non-similar gene k (i.e.; any decision variable x_k such that $x_k(\text{P1}) \neq x_k(\text{P2})$), the probability $\pi(\text{P1})$ (respectively $\pi(\text{P2})$) to set $x_k(\text{C}) = x_k(\text{P1})$ (respectively $x_k(\text{C}) = x_k(\text{P2})$) is given as follows:

$$\pi(\text{P1}) = \frac{F(\text{P1})}{F(\text{P1}) + F(\text{P2})} \text{ (respectively } \pi(\text{P2}) = \frac{F(\text{P2})}{F(\text{P1}) + F(\text{P2})}\text{)} \tag{1}$$

It should be noted that this probability has the same value regarding all parent non-similar genes and is conceived in such a way to give more chance to those belonging to the parent solution (either P1 or P2) having the high value of F (between $F(\text{P1})$ and $F(\text{P2})$).

3.1.2 The restricted version

This restricted version of the two-stage recombination operator can be applied only to a particular class of MOCO problems. That is, a solution of any problem belonging to this particular class can be represented as a string of genes (i.e.; decision variables) so that we can explicitly evaluate the contribution of any gene in the global fitness. This propriety leads to a straightforward comparison between two values of a given gene based on the contribution of each value in the global fitness of the corresponding solution. In this way each gene can be viewed as a partial solution having its own fitness, called gene's fitness, which can be evaluated whenever this gene takes a new value. An example from this particular class is a MOCO problem in which each objective function is partially separable so that each decision variable has its own local fitness with respect to each objective function. That is, for any gene (i.e.; decision variable), we obtain a sort of gene's fitness that can be a sort of aggregation (e.g., weighted sum) of all local fitness corresponding to this decision variable (see the MOKP example in the following subsection). Because this version is problem-dependent, we will just give one possible description which can be considered as one of the most general possible descriptions.

a) **First stage:** the same as that has been described in the general version (subsection 3.1.1).

b) **Second stage:** this stage concerns the parent non-similar genes, so let us denote by k a given parent non-similar gene and let us denote by G_k the gene's fitness function associated with k. Furthermore, let $G_k(\text{P1}) = G_k(x_k(\text{P1}))$ (respectively $G_k(\text{P2}) = G_k(x_k(\text{P2}))$ be the gene's fitness value of k according to P1 (respectively P2). So, the probability to select $x_k(\text{P1})$ (respectively $x_k(\text{P2})$) is conceived in such a way to be proportional to $G_k(\text{P1})$ (respectively $G_k(\text{P2})$). Formally, the probability $\pi_k(\text{P1})$ (respectively $\pi_k(\text{P2})$) to set $x_k(\text{C}) = x_k(\text{P1})$ (respectively $x_k(\text{C}) = x_k(\text{P2})$) is given as follows:

$$\pi_k(\text{P1}) = \frac{G_k(\text{P1})}{G_k(\text{P1}) + G_k(\text{P2})} \text{ (respectively } \pi_k(\text{P2}) = \frac{G_k(\text{P2})}{G_k(\text{P1}) + G_k(\text{P2})}\text{)} \tag{2}$$

It should be noted that each parent non-similar gene has its own probability value that may be different from the probability value of another parent non-similar gene. This probability is conceived in such a way to give more chance to selecting a parent non-similar gene from the parent solution (either P1 or P2) that induces the best gene's fitness value of this gene.

3.2 Application to the MOKP

In this section we consider that we work in the context of the MOKP. The application of the general version of the two-stage recombination operator to the MOKP seems to be rather obvious if we take into consideration the description given in the subsection 3.1.1. Indeed, we have only to provide the mathematical expression of the fitness function F used in the second stage. So, we use the linear scalarizing function as fitness function:

$$F(x) = \sum_{i=1}^{m} \lambda_i f_i(x) \tag{3}$$

where

$$\lambda = (\lambda_1,...,\lambda_m) : \lambda_i \geq 0 \; \forall i = 1,...,m \text{ and } \sum_{i=1}^{m} \lambda_i = 1 \tag{4}$$

is a weight vector which is randomly generated in every recombination operation in order to maintain diversity in the objective space [19,27]. However if we have some specific knowledge about some favourite regions in the objective space, we could choose the weight vectors in such a way to give priority to such regions [20]. In addition to the linear scalarizing function used here, other fitness functions can also be used: the same fitness function used by the algorithm during its selection process, a utility function based on some decision maker's preferences. Since these other fitness functions are either algorithm-dependent or decision maker-dependent, we have finally adopted in our experiments the above-mentioned approach based on the linear scalarizing function (with random weight vectors) which seems to be more general.

On the other hand, a simple examination of the MOKP structure shows that the restricted version of the two-stage recombination operator can be applied to the MOKP. Indeed, we can explicitly calculate the contribution of each gene k (i.e.; decision variable x_k) in the expression of each objective criterion f_i (i = 1,…, m): $p_{ik}x_k$. Therefore, if we aggregate these contributions of k with respect to all objective criteria, we obtain the gene's fitness G_k associated with the gene k:

$$G_k(x_k) = \sum_{i=1}^{m} \lambda_i p_{ik} x_k \tag{5}$$

where λ is a weight vector which is defined as in (4). That is, $G_k(x_k)$ can be viewed as the partial contribution of k (i.e.; x_k) to the solution fitness (i.e.; the fitness of the complete solution $x = (x_1,…, x_n)$) represented here by the linear scalarizing function. Now we apply the restricted version, as it is described in the subsection 3.1.2, to the MOKP. The first stage is obvious and do not bring out any problem. Before describing the second stage, let us remind some proprieties which will be taken into account while applying this stage to the MOKP: a MOKP solution is represented by a string of bits (i.e.; genes or decision variables taking each either the value 0 or the value 1) where each bit represents the state of its corresponding item (bit equal to 1 means that the corresponding item is loaded whereas bit equal to 0 means that the corresponding item is not loaded). Therefore, if k is a parent non-similar gene (i.e.; a decision variable x_k such that $x_k(\text{P1}) \neq x_k(\text{P2})$), one can easily see that we will always have one and only one of the following two cases:

1) $$G_k(\text{P1}) = \sum_{i=1}^{m} \lambda_i p_{ik} \text{ and } G_k(\text{P2}) = 0 \tag{6}$$

2) $$G_k(\text{P1}) = 0 \text{ and } G_k(\text{P2}) = \sum_{i=1}^{m} \lambda_i p_{ik} \tag{7}$$

Thereby, with respect to (2), we will always have one and only one of the following two cases:

1) $$\pi_k(\text{P1}) = 1 \text{ and } \pi_k(\text{P2}) = 0 \tag{8}$$

2) $$\pi_k(\text{P1}) = 0 \text{ and } \pi_k(\text{P2}) = 1 \tag{9}$$

In the context of the MOKP, (8) and (9) mean that all parent non-common items are inserted into the generated offspring solution C without taking into account the quality or the fitness (e.g., regarding profits and/or weights) associated with each of them. In this way, the MOKP offspring instance may be highly overloaded in most of cases. So, as we will show through the experimental results, the application of such a version (i.e.; the restricted version) requires a relatively considerable CPU time in order to restore the feasibility of offspring solutions. Furthermore, the so-called item's fitness is not directly considered by the recombination process itself but it is rather directly used during the repair procedure (i.e.; the procedure of eliminating some items from infeasible solutions in order to restore their feasibility).

That is, in order to effectively speed up the convergence towards the Pareto front, we propose a third version of the two-stage recombination operator called the MOKP-specific version in which the first stage is the same as in the general and the restricted versions whereas the second stage is described as follows. We first introduce an item's fitness function to allow the classification of all parent non-common items according to such fitness. In our experiments, we use an item's fitness based on the formula proposed by Jaszkiewicz [19] in its repair procedure. So, the item's fitness value of each candidate item l (i.e.; for us a candidate item is a parent non-common item: an item which is loaded in one and only one parent instance regarding the two parent instances represented by P1 and P2) is given as follows:

$$H(l) = \frac{\sum_{i=1}^{m} \lambda_i p_{il}}{\sum_{i=1}^{m} w_{il}} \tag{10}$$

where λ is a randomly weight vector defined as in (4). Then, we apply a greedy insertion heuristic according to the decreasing order of the item's fitness *H*. In other words, the greedy heuristic inserts iteratively a candidate item that fits in all knapsacks (regarding the current instance of C) and has the highest item's fitness value. The greedy insertion heuristic is stopped when we can't insert any other candidate item without violating at least one of the MOKP constraints or when the list of candidate items becomes empty. In this way the offspring solution C generated by the MOKP-specific version is always feasible.

4 Overview of test algorithms

To test the performance of the proposed two-stage recombination operator, we will use two state-of-the-art MOEAs, namely, SPEA and NSGA-II. We now introduce an overview of these two algorithms (see for example Deb's book [1] for more details).

4.1 SPEA overview

Zitzler and Thiele [17] proposed a MOEA which they called the Strength Pareto Evolutionary Algorithm (SPEA). SPEA begins with a randomly created population *P* of size *N* and an external archive *Q* (initially empty) with a maximum capacity of N_{max}. In any generation, the non-dominated solutions of the current population P are copied to the current external archive *Q* and, thereafter, the solutions being dominated in *Q* are deleted. Next if the size of the modified external archive exceeds N_{max}, SPEA performs a clustering procedure to reduce the number of individuals in *Q* so that allows keeping solutions which are less crowded in the Pareto front. Then, SPEA uses genetic operators (selection, recombination, mutation) to generate a new population. So, the first step is to assign fitness to each individual in both populations *P* and *Q*. In fact, the algorithm assigns a fitness to each member *x* of the external archive (*Q*) first. The fitness value assigned to *x* is proportional to the number of current population (*P*) members which are dominated by *x*. Thereafter, each member *y* of *P* is assigned a fitness value that is superior to the sum of the fitness values of all external archive members which weakly dominate *y*. Thus, like the individuals of *P*, all the individuals of *Q* participate in selection operation to select pairs of parent solutions. Then, SPEA applies the recombination operation to all selected pairs of parents to generate offspring solutions to which the mutation operation can be applied. Finally, the generated new solutions replace old solutions in the current population *P*.

4.2 NSGA-II overview

NSGA-II was proposed by Deb *et al.* [24] and it can be viewed as an improved version of the Nondominated Sorting Genetic Algorithm (NSGA) which was introduced by Srinivas and Deb [28]. NSGA-II is initialized with a random population. In the NSGA-II algorithm, the fitness evaluation of each individual is based on both its dominance rank and its crowding distance. Indeed, all nondominated individuals are classified into one category (i.e.; first front). The first front is then temporarily ignored and another layer of nondominated individuals (i.e.; second front) is considered. This classification process continues until all individuals in the current population are classified. Furthermore, the so-called crowding distance of an individual is estimated as the density of its neighbours in the population. That is, during the selection process, nondominated individuals are preferred over dominated individuals, but between two individuals with the same dominance rank, the one having the less crowding distance (i.e.; residing in the less crowded region) is preferred. This sort of crowded comparison is used to achieve the goal of diversity without specifying any additional parameters contrary to the traditional fitness sharing which needs specifying the parameter sharing. Besides, NSGA-II adopts an elitism approach called ($\mu + \lambda$)-selection [1] where offspring population and parent population are combined together to form another population.

In this way both offspring and parent solutions participate in selection operation to find a new population using the recombination and mutation operators.

5 Performance measures

In this article, we use two performance measures to assess the performance of the above-detailed versions of the two-stage recombination operator. In the following, we provide the formal definitions of these measures.

5.1 Coverage of two sets

This measure has been proposed by Zitzler [29] and can be used to compare two given approximations of the Pareto front using the dominance relation. Let *A* and *B* be two approximations to the Pareto front, the coverage measure is defined as:

$$C(A,B) = \frac{\left| \left\{ z' \in B \,\middle|\, \exists z \in A : z \succ z' \right\} \right|}{|B|}$$

The value $C(A,B)=1$ means that all points in B are dominated by some points in A, while the value $C(A,B)=0$ implies that none of the points in B is dominated by a point in A. Since in general $C(A,B) \neq 1 - C(B,A)$, we have to consider both measures (i.e.; $C(A,B)$ and $C(B,A)$).

5.2 Range measure

This measure is equivalent to the so-called maximum spread of Zitzler [29] and can be introduced to give some information about the diversity of the obtained approximation to the Pareto front (in fact, it gives information regarding the spread of solutions over the Pareto front). Let *A* be an approximation to the Pareto front, the range measure (see [11]) calculate the sum of the range values for each objective function over the set *A*:

$$Range(A) = \sum_{i=1}^{m} \left[\max_{x \in A}(f_i(x)) - \min_{x \in A}(f_i(x)) \right]$$

6 Experiments design and parameter settings

The proposed two-stage recombination operator has been compared to some traditional recombination operators using SPEA and NSGA-II algorithms in the MOKP context. In practice, we compare each version of the two-stage recombination operator to the one-point crossover, the two-point crossover and the uniform crossover (i.e.; the crossover operator that consists to construct an offspring solution by selecting each bit either from the first parent or from the second parent with uniform probability equal to 0.5), respectively. Zitzler and Thiele [17] examined the performance of several MOEAs on the MOKP using nine test instances including 2, 3 and 4 objectives, associated with 250, 500 and 750 items, respectively. In this paper, we use the same instances which can be annotated by [number of objectives - number of items] (i.e.; 2-250, 2-500, 2-750, 3-250, 3-500, 3-750, 4-250, 4-500 and 4-750). The data files reporting these instances can be downloaded from the link [30]. The standard binary tournament is used as selection procedure in both SPEA and NSGA-II. Furthermore, the size of the second population (i.e.; the external archive) of SPEA is set to 15000 as in a number of experiments in the literature (e.g., [19]). Moreover, except for the MOKP-specific version of the two-stage recombination operator, all other recombination operators experimented in this paper may generate many unfeasible solutions. In addition, both the random initialization and the mutation operator (in fact, the bit-flip mutation) used in our experiments can also produce some unfeasible solutions. So, in order to restore solutions' feasibility, we apply the greedy repair procedure proposed by Jaszkiewicz [19] (i.e.; items are removed in the increasing order of the item's fitness given in (10) until the solution becomes feasible). Furthermore, all experiments are performed under the following parameter specifications:

Table 1. Some other common parameter settings

MOKP instance	Population size	Recombination probability	Mutation probability (per bit)
2-250	150	1.00	1/250
2-500	200	1.00	1/500
2-750	250	1.00	1/750
3-250	200	1.00	1/250
3-500	250	1.00	1/500
3-750	300	1.00	1/750
4-250	250	1.00	1/250
4-500	300	1.00	1/500
4-750	350	1.00	1/750

According to Table 1, we set the recombination probability to a rather high value (in fact, all parent solutions undergo the recombination procedure) so that we can clearly see the effect of the recombination operator. Besides, the population sizes are similar to those used in the experiments of Zitzler and Thiele [17] whereas the mutation probability is set according to the rule: [1/number of items]. We should note that such a rule is recommended in accordance with some experimental studies (e.g., [11]). That is, in addition to the recombination operator, both SPEA and NSGA-II include all other standard evolutionary operators (i.e.; the selection operator, the mutation operator) so as to evaluate the effect of the recombination operator on the performance of a so-called complete MOEA (i.e.; a MOEA including all the standard evolutionary operators).

As for the stopping criterion used in both SPEA and NSGA-II, we use a maximum number of generations which is set by taking into account the CPU time required by each of the recombination operators being compared in this paper. Indeed, we first fix a maximum number of generations (and so the corresponding CPU time T_1) as stopping criterion for the traditional crossover operator (i.e.; the one-point crossover, the two-point crossover and the uniform crossover, respectively). Then, we set the maximum number of generations (i.e.; the stopping criterion) for each version of the two-stage recombination operator so that the corresponding CPU time T_2 should be comparable to T_1; i.e.; we should have either [$T_2 \leq T_1$] or [T_1 is slightly smaller than T_2]. This approach, rather based on the CPU time and not on the number of evaluations, can be justified by the following arguments. First, all experiments are performed using the same software/hardware platform (in fact, our implementation was based on the standard C++ using the MOMH library of Jaszkiewicz [31] under Windows on a PC Centrino Duo 1.66 GHS). Second, the above-mentioned approach has the advantage of taking into consideration differences in time needed to generate new offspring solutions by different recombination operators [32]. In this way, this approach concentrates on testing the efficiency of the proposed recombination operator in comparison with another more traditional recombination operator (i.e.; testing its ability to generate better solutions during a time comparable to the time required by the traditional one). So we will provide in the next section, devoted to the experimental results, the stopping conditions associated with each comparative experiment. On the other hand we should note that, for all experiments carried out in this paper, each algorithm run is repeated 30 times and the average values of obtained results (e.g., CPU times and performance measures) are calculated and then reported as final results.

7 Experimental results and discussions

In the first subsection, we will give and discuss the results obtained from comparison between the general version of the two-stage recombination operator and each traditional crossover operator (in fact, the one-point crossover, the two-point crossover and the uniform crossover, respectively). The second subsection will be devoted to the comparative results regarding the restricted version of the two-stage recombination operator, whereas the third subsection will be devoted to those regarding the MOKP-specific version. But we first give the following notations:

- SPEA[one-point]: SPEA with the one-point crossover.
- SPEA[two-point]: SPEA with the two-point crossover.
- SPEA[uniform]: SPEA with the uniform crossover.

- SPEA[general-two-stage]: SPEA with the general version of the two-stage recombination.
- SPEA[restricted-two-stage]: SPEA with the restricted version of the two-stage recombination.
- SPEA[MOKP-two-stage]: SPEA with the MOKP-specific version of the two-stage recombination.
- NSGA-II[one-point]: NSGA-II with the one-point crossover.
- NSGA-II[two-point]: NSGA-II with the two-point crossover.
- NSGA-II[uniform]: NSGA-II with the uniform crossover.
- NSGA-II[general-two-stage]: NSGA-II with the general version of the two-stage recombination.
- NSGA-II[restricted-two-stage]: NSGA-II with the restricted version of the two-stage recombination.
- NSGA-II[MOKP-two-stage]: NSGA-II with the MOKP-specific version of the two-stage recombination.

7.1 Comparative results regarding the general version

In our preliminary experiments we have remarked that for a given number of generations, the CPU time required by the general version of the two-stage recombination operator is rather comparable to the CPU time required by each of the three traditional crossovers tackled in this paper. We thus decide to fix a similar number of generations as stopping condition regarding all the recombination operators tested in this subsection (in fact, the general version of the two-stage recombination, the one-point crossover, the two-point crossover and the uniform crossover). That is, the common number of generations is set to 50 and 500 generations, respectively (see Table 2) in order to clearly distinguish between the behaviour of the general version of the two-stage recombination operator with respect to a somewhat small number of generations (in fact, 50 generations) and its behaviour with respect to a somewhat great number of generations (in fact, 500 generations).

Table 2. Common stopping conditions for SPEA versions and NSGA-II versions

Algorithm version	First common stopping condition	Second common stopping condition
SPEA[one-point]	50 generations	500 generations
SPEA[two-point]		
SPEA[uniform]		
SPEA[general-two-stage]		
NSGA-II[one-point]		
NSGA-II[two-point]		
NSGA-II[uniform]		
NSGA-II[general-two-stage]		

For each of the MOKP instances considered in this paper, the average running times regarding different stopping conditions (i.e.; 50 and 500 generations, respectively) are summarized in Table 3 and Table 4, respectively. From these tables we can see that in general there is no significant difference between the CPU time required by SPEA[general-two-stage] and that required by SPEA[one-point], SPEA[two-point] and SPEA[uniform], respectively, and so these CPU times can be considered to be comparable one to another as mentioned above. This remark is also valid for NSGA-II versions.

Table 3. Average CPU times (in seconds) corresponding to SPEA versions and NSGA-II versions with 50 generations as stopping condition

	MOKP instances								
	2-250	**2-500**	**2-750**	**3-250**	**3-500**	**3-750**	**4-250**	**4-500**	**4-750**
CPU time of SPEA[one-point]	1.40	3.56	7.00	2.86	7.10	12.83	5.43	12.26	22.73
CPU time of SPEA[two-point]	1.43	3.60	7.20	2.93	7.16	13.16	5.30	12.26	22.53
CPU time of SPEA[uniform]	1.60	4.23	8.93	3.43	8.70	16.96	6.10	14.70	27.56
CPU time of SPEA[general-two-stage]	1.73	4.83	9.96	3.53	9.13	17.36	6.23	14.63	27.70
CPU time of NSGA-II[one-point]	3.26	7.13	12.83	6.26	12.80	21.86	10.70	20.33	34.23
CPU time of NSGA-II[two-point]	3.36	7.56	13.66	6.43	13.23	22.96	10.80	21.06	35.93
CPU time of NSGA-II[uniform]	3.66	8.70	16.53	6.90	15.46	28.23	11.90	24.73	44.13
CPU time of NSGA-II[general-two-stage]	3.70	8.70	16.63	7.10	15.13	26.80	11.36	23.46	41.60

Table 4. Average CPU times (in seconds) corresponding to SPEA versions and NSGA-II versions with 500 generations as stopping condition

	MOKP instances								
	2-250	**2-500**	**2-750**	**3-250**	**3-500**	**3-750**	**4-250**	**4-500**	**4-750**
CPU time of SPEA[one-point]	16.96	39.16	70.06	53.10	104.93	161.53	162.63	252.96	384.86
CPU time of SPEA[two-point]	16.00	38.90	70.70	53.26	106.20	165.90	159.70	263.10	386.93
CPU time of SPEA[uniform]	17.63	43.06	81.76	54.70	117.23	195.96	153.46	283.10	444.80
CPU time of SPEA[general-two-stage]	18.80	46.50	85.93	54.63	118.73	198.33	155.86	289.63	456.36
CPU time of NSGA-II[one-point]	31.43	66.33	112.80	60.93	117.00	191.86	109.93	195.36	309.46
CPU time of NSGA-II[two-point]	31.63	68.03	118.76	63.70	122.46	206.53	111.36	201.23	320.86
CPU time of NSGA-II[uniform]	32.80	72.76	137.33	68.63	141.80	235.86	117.70	227.30	380.03
CPU time of NSGA-II[general-two-stage]	33.86	72.93	134.33	67.63	138.13	230.30	115.50	220.03	364.06

SPEA[general-two-stage] is compared against SPEA[one-point], SPEA[two-point] and SPEA[uniform], respectively, using the coverage measure. NSGA-II[general-two-stage] is also compared against NSGA-II[one-point], NSGA-II[two-point] and NSGA-II[uniform], respectively, using the coverage measure. The average coverage measures (in percentage rate %) corresponding to 50 generations (as stopping condition) are given in Tables 5-10. From Tables 5, 6, 8 and 9, we can see that the general version of the two-stage recombination operator outperforms both the one-point crossover and the two-point crossover in terms of convergence. Moreover, we can clearly see that there is a complete outperformance for the MOKP instances with 500 and 750 items, i.e.; almost all solutions corresponding to the one-point crossover (respectively the two-point crossover) are dominated by some solutions corresponding to the general version of the two-stage recombination operator and there is no solution corresponding to the latter recombination operator that is dominated by a solution corresponding to the one-point crossover (respectively the two-point crossover). Table 7 shows that, when it is compared against the uniform crossover, the general version of the two-stage recombination operator significantly improves the convergence performance of SPEA especially on the MOKP instances with a large number of items (in fact, 500 and 750 items, respectively). As for coverage measure comparison between the general version of the two-stage recombination operator and the uniform crossover in the context of NSGA-II, the obtained results (see Table 10) show that the former recombination operator does not significantly improve the NSGA-II convergence on the two smallest MOKP instances (i.e.; 2-250 and 2-500) while its convergence performance becomes more significant on the MOKP instances having the highest number of items (i.e.; 750 items).

Table 5. Average coverage measures (%) corresponding to SPEA[general-two-stage] and SPEA[one-point] with 50 generations as stopping condition

MOKP instance	*C*(SPEA[one-point], SPEA[general-two-stage])	*C*(SPEA[general-two-stage], SPEA[one-point])
2-250	0.00%	99.52%
2-500	0.00%	100%
2-750	0.00%	100%
3-250	0.02%	98.60%
3-500	0.00%	100%
3-750	0.00%	100%
4-250	0.00%	98.25%
4-500	0.00%	100%
4-750	0.00%	100%

Table 6. Average coverage measures (%) corresponding to SPEA[general-two-stage] and SPEA[two-point] with 50 generations as stopping condition

MOKP instance	*C*(SPEA[two-point], SPEA[general-two-stage])	*C*(SPEA[general-two-stage], SPEA[two-point])
2-250	0.11%	96.48%
2-500	0.00%	100%
2-750	0.00%	100%
3-250	0.25%	96.60%
3-500	0.00%	100%
3-750	0.00%	100%
4-250	0.10%	95.34%
4-500	0.00%	99.99%
4-750	0.00%	100%

Table 7. Average coverage measures (%) corresponding to SPEA[general-two-stage] and SPEA[uniform] with 50 generations as stopping condition

MOKP instance	*C*(SPEA[uniform], SPEA[general-two-stage])	*C*(SPEA[general-two-stage], SPEA[uniform])
2-250	12.55%	73.98%
2-500	0.19%	99.29%
2-750	0.00%	98.57%
3-250	19.41%	49.49%
3-500	6.04%	75.10%
3-750	0.53%	93.24%
4-250	20.98%	39.25%
4-500	9.62%	60.56%
4-750	4.11%	75.10%

Table 8. Average coverage measures (%) corresponding to NSGA-II[general-two-stage] and NSGA-II[one-point] with 50 generations as stopping condition

MOKP instance	*C*(NSGA-II[one-point], NSGA-II[general-two-stage])	*C*(NSGA-II[general-two-stage], NSGA-II[one-point])
2-250	0.00%	99.59%
2-500	0.00%	100%
2-750	0.00%	100%
3-250	0.01%	98.68%
3-500	0.00%	100%
3-750	0.00%	100%
4-250	0.02%	98.81%
4-500	0.00%	100%
4-750	0.00%	100%

Table 9. Average coverage measures (%) corresponding to NSGA-II[general-two-stage] and NSGA-II[two-point] with 50 generations as stopping condition

MOKP instance	*C*(NSGA-II[two-point], NSGA-II[general-two-stage])	*C*(NSGA-II[general-two-stage], NSGA-II[two-point])
2-250	0.08%	97.52%
2-500	0.00%	100%
2-750	0.00%	100%
3-250	0.28%	94.03%
3-500	0.00%	100%
3-750	0.00%	100%
4-250	0.24%	93.39%
4-500	0.00%	99.95%
4-750	0.00%	100%

Table 10. Average coverage measures (%) corresponding to NSGA-II[general-two-stage] and NSGA-II[uniform] with 50 generations as stopping condition

MOKP instance	*C*(NSGA-II[uniform], NSGA-II[general-two-stage])	*C*(NSGA-II[general-two-stage], NSGA-II[uniform])
2-250	34.70%	44.22%
2-500	40.13%	45.23%
2-750	18.75%	67.69%
3-250	26.46%	40.71%
3-500	21.85%	50.50%
3-750	12.90%	62.32%
4-250	19.77%	38.47%
4-500	12.91%	48.82%
4-750	11.08%	52.05%

We now present (see Tables 11-16) and discuss the obtained average coverage measures corresponding to 500 generations as stopping condition. Tables 11, 12, 14 and 15 show that the general version of the two-stage recombination operator still outperforms both the one-point crossover and the two-point crossover with respect to the convergence aspect. In this sense, we should note that the best results corresponding to the general version of the two-stage recombination operator (in comparison with the one-point crossover and the two-point crossover, respectively) in the SPEA context are achieved on the large-scale MOKP instances (in fact, 3-500, 3-750, 4-500 and 4-750) while its corresponding best results in the NSGA-II context are achieved on the MOKP instances with a large number of items (i.e.; 500 and 750 items, respectively). Compared to the uniform crossover in the SPEA context (see Table 13), the general version of the two-stage recombination operator still improves, but not very significantly, the convergence performance of SPEA and the corresponding best results are achieved on the MOKP instances having the highest number of items (i.e.; 750 items). However when it is compared to the uniform crossover in the NSGA-II context (see Table 16), the general version of the two-stage recombination operator is in general slightly outperformed by the uniform crossover on the smallest MOKP instances (i.e.; 2-250, 2-500, 2-750 and 3-250) while it slightly outperforms the uniform crossover on the largest MOKP instances (i.e.; 3-500, 3-750, 4-250, 4-500 and 4-750).

From the above analysis it can be clearly seen that the general version of the two-stage recombination operator significantly outperforms the one-point and the two-point crossovers with respect to both the two stopping conditions used here (i.e.; 50 and 500 generations, respectively), while its convergence performance against the uniform crossover degrades when the number of generations grows from 50 to 500. That is, this difference between the convergence performance of the general version of the two-stage recombination operator (especially in comparison with the uniform crossover) with respect to a somewhat small number of generations (in fact, 50 generations) and its convergence performance with respect to a somewhat great number of generations (in fact, 500 generations) can be discussed as follows. In early generations (i.e.; when the number of generations is small), there may be significant differences between individuals regarding the fitness quality and then the use of the general version of the two-stage recombination operator can effectively improve the convergence performance of the MOEA (i.e.; SPEA or NSGA-II) since such a recombination operator takes into account some fitness information so as to favour the survival of genes belonging to parents with high fitness quality. Moreover as well as the number of generations grows, the evolutionary population becomes more and more mature and so when the number of generations reaches a rather high value, the fitness quality' differences between parent solutions become no rather significant and hence the selection probability used in the general version of the two-stage recombination operator (see equation (1)) does not have a significant positive effect on the convergence performance if it is compared to the uniform probability 0.5 used in the uniform crossover.

Table 11. Average coverage measures (%) corresponding to SPEA[general-two-stage] and SPEA[one-point] with 500 generations as stopping condition

MOKP instance	*C*(SPEA[one-point], SPEA[general-two-stage])	*C*(SPEA[general-two-stage], SPEA[one-point])
2-250	21.45%	63.14%
2-500	13.11%	67.80%
2-750	2.92%	82.39%
3-250	5.39%	79.22%
3-500	0.08%	95.83%
3-750	0.00%	98.39%
4-250	2.08%	82.85%
4-500	0.00%	99.32%
4-750	0.00%	99.99%

Table 12. Average coverage measures (%) corresponding to SPEA[general-two-stage] and SPEA[two-point] with 500 generations as stopping condition

MOKP instance	*C*(SPEA[two-point], SPEA[general-two-stage])	*C*(SPEA[general-two-stage], SPEA[two-point])
2-250	25.58%	56.79%
2-500	14.69%	68.36%
2-750	6.92%	73.96%
3-250	7.92%	73.04%
3-500	1.00%	89.74%
3-750	0.02%	95.78%
4-250	6.00%	67.83%
4-500	0.02%	96.09%
4-750	0.00%	99.91%

Table 13. Average coverage measures (%) corresponding to SPEA[general-two-stage] and SPEA[uniform] with 500 generations as stopping condition

MOKP instance	*C*(SPEA[uniform], SPEA[general-two-stage])	*C*(SPEA[general-two-stage], SPEA[uniform])
2-250	32.71%	52.57%
2-500	37.12%	48.02%
2-750	23.66%	62.52%
3-250	30.25%	43.33%
3-500	31.16%	43.40%
3-750	28.10%	47.20%
4-250	27.12%	37.88%
4-500	24.15%	40.72%
4-750	25.81%	39.16%

Table 14. Average coverage measures (%) corresponding to NSGA-II[general-two-stage] and NSGA-II[one-point] with 500 generations as stopping condition

MOKP instance	*C*(NSGA-II[one-point], NSGA-II[general-two-stage])	*C*(NSGA-II[general-two-stage], NSGA-II[one-point])
2-250	6.99%	84.82%
2-500	0.04%	98.71%
2-750	0.00%	99.73%
3-250	0.58%	92.61%
3-500	0.35%	97.04%
3-750	0.05%	97.81%
4-250	3.58%	76.24%
4-500	0.96%	87.13%
4-750	0.04%	97.53%

Table 15. Average coverage measures (%) corresponding to NSGA-II[general-two-stage] and NSGA-II[two-point] with 500 generations as stopping condition

MOKP instance	*C*(NSGA-II[two-point], NSGA-II[general-two-stage])	*C*(NSGA-II[general-two-stage], NSGA-II[two-point])
2-250	9.44%	81.15%
2-500	0.13%	98.70%
2-750	0.24%	97.77%
3-250	3.57%	83.49%
3-500	1.64%	90.70%
3-750	0.43%	90.63%
4-250	7.56%	68.51%
4-500	3.99%	74.04%
4-750	0.61%	89.80%

Table 16. Average coverage measures (%) corresponding to NSGA-II[general-two-stage] and NSGA-II[uniform] with 500 generations as stopping condition

MOKP instance	*C*(NSGA-II[uniform], NSGA-II[general-two-stage])	*C*(NSGA-II[general-two-stage], NSGA-II[uniform])
2-250	55.96%	29.28%
2-500	47.59%	40.01%
2-750	45.95%	40.02%
3-250	38.82%	29.68%
3-500	33.96%	35.80%
3-750	32.89%	36.34%
4-250	28.92%	35.28%
4-500	26.73%	33.36%
4-750	23.96%	36.36%

On the other hand, the average results regarding the range measure are reported in the two following tables (i.e.; Tables 17 and 18). It should be noted that only the range results corresponding to 500 generations are presented and discussed here since the concluding remarks regarding the range measure are similar with respect to both the two stopping conditions corresponding to 50 and 500 generations, respectively. From Table 17 we can see that the one-point and the two-point crossovers give the best results in terms of spread over the Pareto front in the SPEA context. The best spread results in the NSGA-II context are however given by the general version of the two-stage recombination operator and the uniform crossover (see Table 18). Between the general version of the two-stage recombination operator and the uniform recombination operator, the range results given by the latter one are in general slightly better than those given by the former one with respect to both the SPEA and the NSGA-II contexts. To visually bear out our discussion about the solutions' spread, we have depicted the solutions' sets corresponding to a 500 generations' single run on the 3-750 MOKP instance with respect to each recombination operator tackled here (see Figures 1-4).

Table 17. Average range measures corresponding to SPEA versions with 500 generations as stopping condition

MOKP instance	*Range* corresponding to SPEA[one-point]	*Range* corresponding to SPEA[two-point]	*Range* corresponding to SPEA[uniform]	*Range* corresponding to SPEA[general-two-stage]
2-250	2726.83	2710.03	2344.63	2358.53
2-500	3347.23	3362.07	2730.00	2678.10
2-750	4544.37	4634.67	3594.03	3658.07
3-250	4693.43	4726.77	4465.10	4334.50
3-500	6519.93	6593.63	6313.03	6156.40
3-750	7370.67	7477.83	7044.43	6913.67
4-250	5873.10	5823.40	5400.90	5362.87
4-500	9052.30	9183.17	8049.13	7984.47
4-750	11458.30	11517.30	10595.90	10317.50

Table 18. Average range measures corresponding to NSGA-II versions with 500 generations as stopping condition

MOKP instance	*Range* corresponding to NSGA-II[one-point]	*Range* corresponding to NSGA-II[two-point]	*Range* corresponding to NSGA-II[uniform]	*Range* corresponding to NSGA-II[general-two-stage]
2-250	1914.70	2048.13	2227.20	2070.90
2-500	2729.17	2917.43	3250.00	3041.77
2-750	4009.90	4196.53	5120.17	4679.03
3-250	3061.03	3082.23	3407.00	3336.70
3-500	3698.33	3861.50	4776.27	4472.17
3-750	3857.43	4595.73	5556.23	5215.80
4-250	3342.23	3668.27	3970.17	3774.47
4-500	4076.13	4562.97	5827.60	5355.97
4-750	4764.80	5075.70	6665.70	6343.40

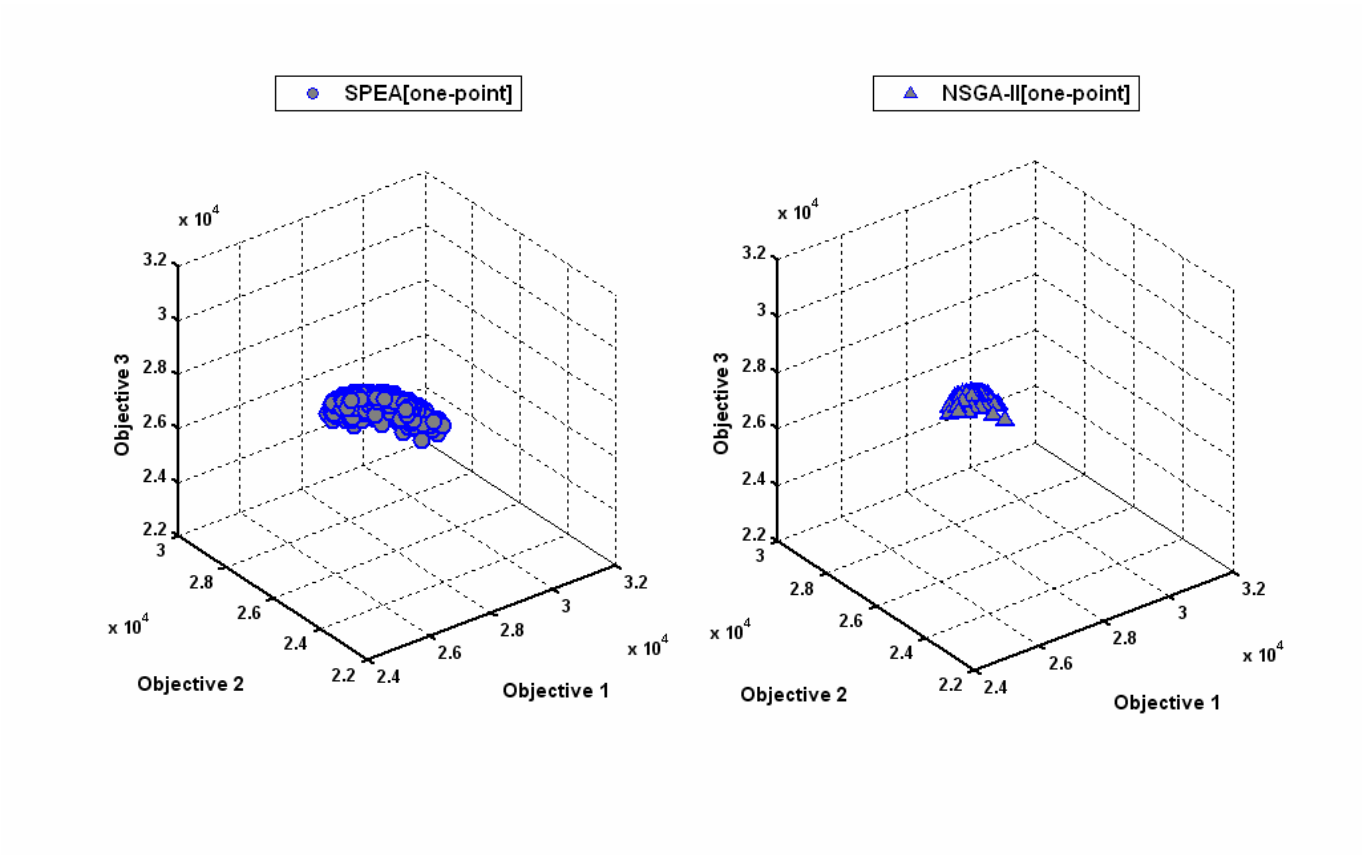

Figure 1. Solutions' distribution corresponding to the one-point crossover with respect to SPEA (left) and NSGA-II (right) for a single run on the 3-750 MOKP instance.

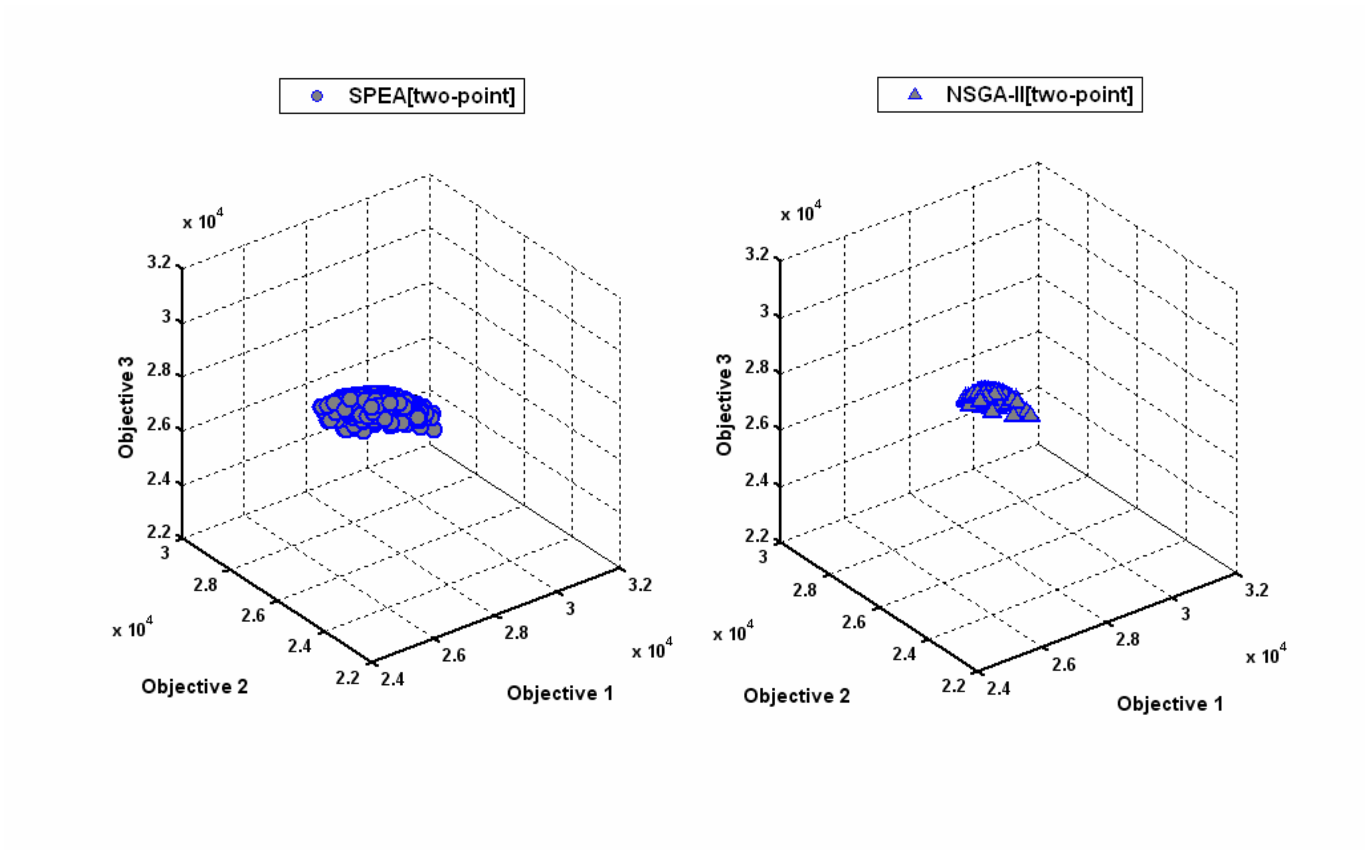

Figure 2. Solutions' distribution corresponding to the two-point crossover with respect to SPEA (left) and NSGA-II (right) for a single run on the 3-750 MOKP instance.

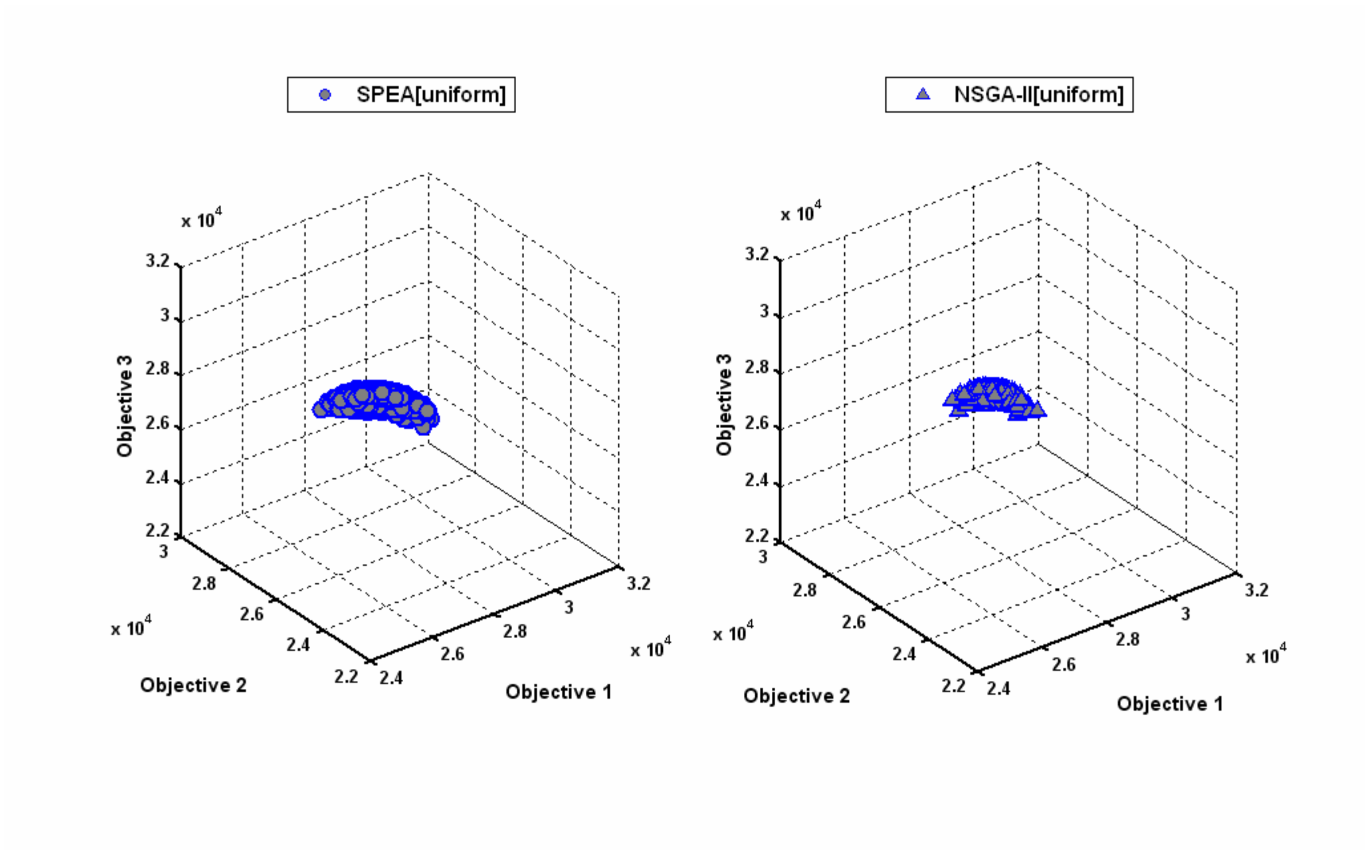

Figure 3. Solutions' distribution corresponding to the uniform crossover with respect to SPEA (left) and NSGA-II (right) for a single run on the 3-750 MOKP instance.

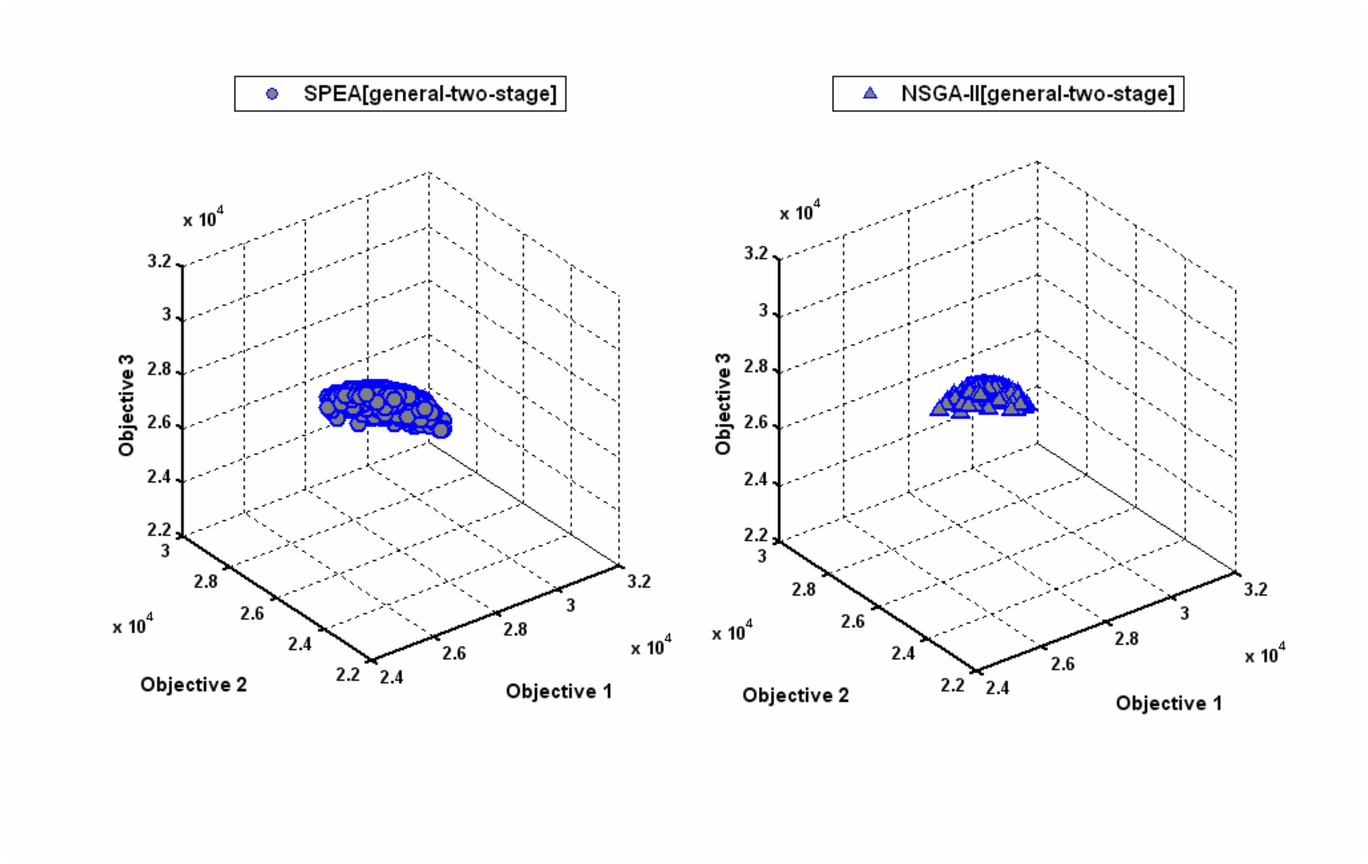

Figure 4. Solutions' distribution corresponding to the general version of the two-stage recombination operator with respect to SPEA (left) and NSGA-II (right) for a single run on the 3-750 MOKP instance.

7.2 Comparative results regarding the restricted version

We have realized through running tests that the restricted version of the two-stage recombination operator needs much more CPU time than that needed by the traditional crossover operators tackled for comparison (i.e.; the one-point crossover, the two-point crossover and the uniform crossover). That is, to have comparable CPU times while setting the stopping condition to 500 generations for the traditional crossover operators, we set the stopping condition to 30 generations for the restricted version of the two-stage recombination operator (see Table 19). From Table 4 and Table 20, we can indeed see that the CPU time required by the restricted version of the two-stage recombination operator is comparable to that required by each of the three traditional crossover operators.

Table 19. Stopping conditions for SPEA versions and NSGA-II versions

Algorithm version	Stopping condition
SPEA[one-point]	500 generations
SPEA[two-point]	500 generations
SPEA[uniform]	500 generations
SPEA[restricted-two-stage]	30 generations
NSGA-II[one-point]	500 generations
NSGA-II[two-point]	500 generations
NSGA-II[uniform]	500 generations
NSGA-II[restricted-two-stage]	30 generations

Table 20. Average CPU times (in seconds) corresponding to SPEA[restricted-two-stage] and NSGA-II[restricted-two-stage] with 30 generations as stopping condition

	MOKP instances								
	2-250	**2-500**	**2-750**	**3-250**	**3-500**	**3-750**	**4-250**	**4-500**	**4-750**
CPU time of SPEA[restricted-two-stage]	7.53	34.10	95.06	18.10	78.80	187.76	32.76	126.00	310.06
CPU time of NSGA-II[restricted-two-stage]	8.60	37.13	103.80	17.50	72.83	168.20	31.30	116.76	291.43

SPEA[restricted-two-stage] is compared against SPEA[one-point], SPEA[two-point] and SPEA[uniform], respectively, using the coverage measure. NSGA-II[restricted-two-stage] is also compared against NSGA-II[one-point], NSGA-II[two-point] and NSGA-II[uniform], respectively, using the coverage measure. The average coverage measures (in percentage rate %) are given in Tables 21-26. Tables 21-23 show that the restricted version of the two-stage recombination operator outperforms the one-point, the two-point and the uniform crossovers in the SPEA context. We should indeed note that the best coverage results of the restricted version of the two-stage recombination operator in the SPEA context (e.g., the results corresponding to a complete outperformance especially against the one-point and the two-point crossovers) are achieved on the MOKP instances with a large number of items (in fact, 500 and 750 items, respectively). From Table 24 and Table 25 we can clearly see that the restricted version of the two-stage recombination operator outperforms both the one-point and the two-point crossovers in the NSGA-II context. Moreover, the best convergence performance of the restricted version of the two-stage recombination operator (in comparison with the one-point and the two-point crossovers) in the NSGA-II context is reached on the MOKP instances having a large number of items and a small number of objectives (e.g., 2-500 and 2-750 where there is a complete outperformance). In comparison with the uniform crossover in the NSGA-II context (see Table 26), the restricted version of the two-stage recombination operator still in general gives the best coverage results on the majority of the MOKP instances tackled in this paper. However the convergence performance of the restricted version of the two-stage recombination operator against the uniform crossover in the NSGA-II context drastically degrades on the MOKP instances having a small number of items and a high number of objectives (e.g., the restricted version of the two-stage recombination operator is even outperformed by the uniform crossover on the 4-250 MOKP instance).

Table 21. Average coverage measures (%) corresponding to SPEA[restricted-two-stage] and SPEA[one-point]

MOKP instance	*C*(SPEA[one-point], SPEA[restricted-two-stage])	*C*(SPEA[restricted-two-stage], SPEA[one-point])
2-250	13.44%	69.23%
2-500	0.44%	97.85%
2-750	0.03%	99.67%
3-250	2.41%	86.26%
3-500	0.00%	99.99%
3-750	0.00%	100%
4-250	1.03%	87.51%
4-500	0.00%	100%
4-750	0.00%	100%

Table 22. Average coverage measures (%) corresponding to SPEA[restricted-two-stage] and SPEA[two-point]

MOKP instance	*C*(SPEA[two-point], SPEA[restricted-two-stage])	*C*(SPEA[restricted-two-stage], SPEA[two-point])
2-250	17.77%	62.65%
2-500	0.66%	95.75%
2-750	0.07%	99.34%
3-250	3.71%	81.86%
3-500	0.00%	99.88%
3-750	0.00%	100%
4-250	3.39%	71.75%
4-500	0.00%	99.98%
4-750	0.00%	100%

Table 23. Average coverage measures (%) corresponding to SPEA[restricted-two-stage] and SPEA[uniform]

MOKP instance	*C*(SPEA[uniform], SPEA[restricted-two-stage])	*C*(SPEA[restricted-two-stage], SPEA[uniform])
2-250	21.94%	48.34%
2-500	6.23%	71.30%
2-750	2.73%	86.06%
3-250	11.56%	49.47%
3-500	0.62%	89.27%
3-750	0.36%	91.79%
4-250	10.63%	37.81%
4-500	0.25%	86.06%
4-750	0.00%	99.00%

Table 24. Average coverage measures (%) corresponding to NSGA-II[restricted-two-stage] and NSGA-II[one-point]

MOKP instance	*C*(NSGA-II[one-point], NSGA-II[restricted-two-stage])	*C*(NSGA-II[restricted-two-stage], NSGA-II[one-point])
2-250	0.82%	96.76%
2-500	0.00%	100%
2-750	0.00%	100%
3-250	0.67%	86.59%
3-500	0.07%	95.78%
3-750	0.00%	99.32%
4-250	5.91%	17.26%
4-500	0.53%	42.94%
4-750	0.00%	92.03%

Table 25. Average coverage measures (%) corresponding to NSGA-II[restricted-two-stage] and NSGA-II[two-point]

MOKP instance	*C*(NSGA-II[two-point], NSGA-II[restricted-two-stage])	*C*(NSGA-II[restricted-two-stage], NSGA-II[two-point])
2-250	0.65%	96.50%
2-500	0.00%	100%
2-750	0.00%	99.97%
3-250	3.46%	59.24%
3-500	0.70%	73.75%
3-750	0.13%	91.33%
4-250	8.90%	10.30%
4-500	1.28%	25.99%
4-750	0.09%	68.05%

Table 26. Average coverage measures (%) corresponding to NSGA-II[restricted-two-stage] and NSGA-II[uniform]

MOKP instance	*C*(NSGA-II[uniform], NSGA-II[restricted-two-stage])	*C*(NSGA-II[restricted-two-stage], NSGA-II[uniform])
2-250	12.78%	68.13%
2-500	8.22%	70.67%
2-750	16.99%	48.68%
3-250	13.17%	20.62%
3-500	5.40%	25.24%
3-750	4.59%	27.35%
4-250	15.36%	6.36%
4-500	3.54%	10.63%
4-750	0.98%	20.29%

To have an idea about the diversity performance of the restricted version of the two-stage recombination operator, we report its corresponding average range measures in Table 27 and depict its corresponding solutions' sets for a single run on the 3-750 MOKP instance in Figure 5. From Tables 17, 18 and 27, it can be clearly seen that the restricted version of the two-stage recombination operator provides the best range results in comparison with the one-point crossover, the two-point crossover and the uniform crossover, respectively, in both the SPEA and the NSGA-II contexts. This high ability of the restricted version of the two-stage recombination operator to spread out the solutions over a large area in the Pareto front can be visually shown in Figure 5. We should also note that the comparison between Figure 5 and Figures 1-3 show that the restricted version of the two-stage recombination operator succeeds to find much more solutions and these solutions are well-dispersed and even well-distributed than those found by each of the three traditional crossovers. This good diversity performance of the restricted version of the two-stage recombination operator may be due to its own mechanism of inserting the parent non-common items in its second stage. Indeed, in every recombination operation, all parent non-common items are inserted into the offspring instance so that there is more chance to finally obtain an offspring solution belonging to a region being different from those corresponding to its parent solutions even if we must use the repair procedure before obtaining the final feasible offspring solution since the weight vector used in such a procedure is randomly updated. In other words, each item and especially each parent non-common item can be viewed as a solution in the MOKP context and hence we can define its location in the search space (i.e.; either the decision space or the objective space) so that an offspring solution in which are combined many parent non-common items may be likely located in another search region (i.e.; other than that corresponding to each of the two parent solutions) generated by both the parent common items and the parent non-common items.

Table 27. Average range measures corresponding to SPEA[restricted-two-stage] and NSGA-II[restricted-two-stage]

MOKP instance	*Range* corresponding to SPEA[restricted-two-stage]	*Range* corresponding to NSGA-II[restricted-two-stage]
2-250	4417.23	4427.07
2-500	7375.17	7343.17
2-750	12849.20	12713.90
3-250	7274.90	7124.93
3-500	13613.10	13363.30
3-750	19687.90	19161.90
4-250	8883.10	8802.73
4-500	17993.20	18195.10
4-750	26938.20	27381.60

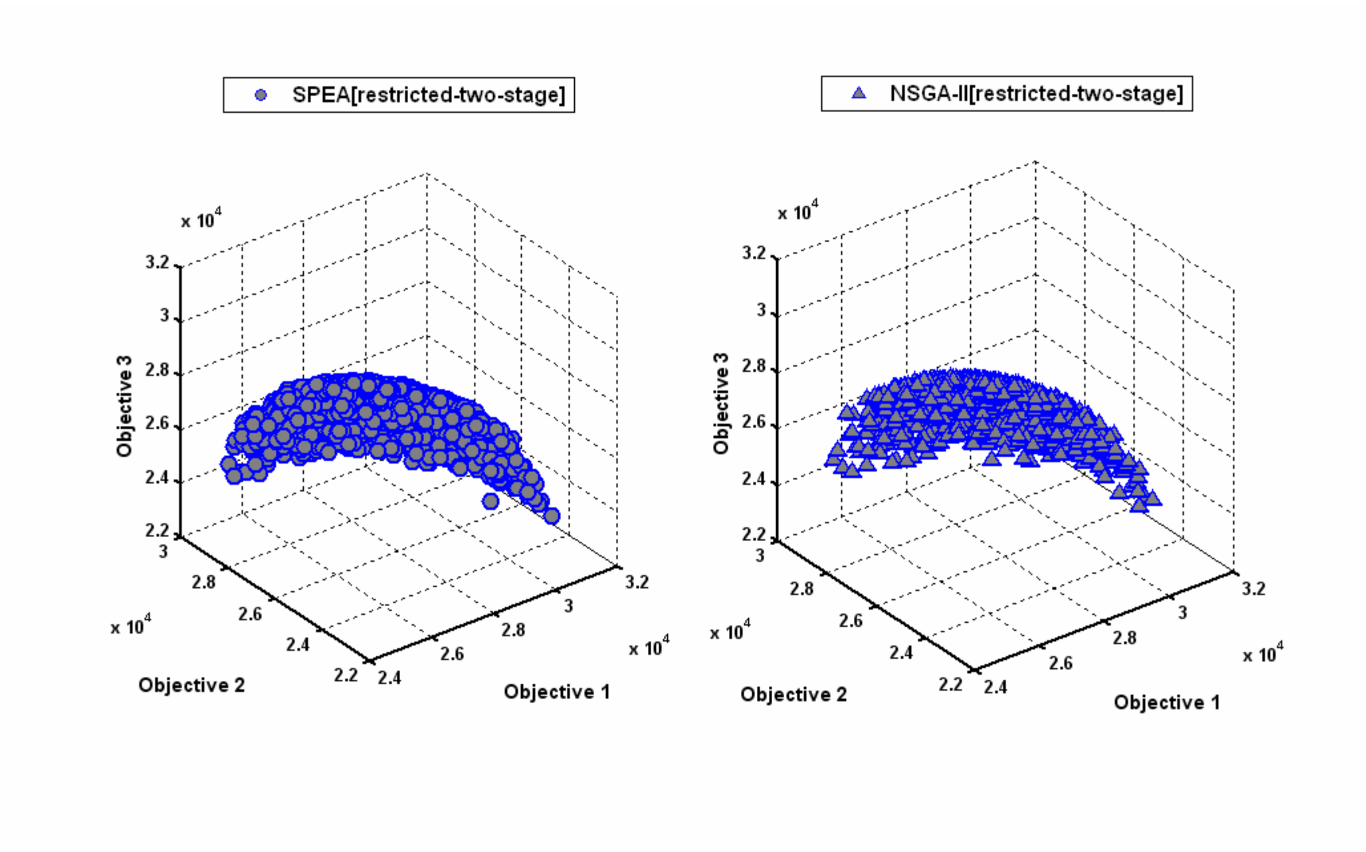

Figure 5. Solutions' distribution corresponding to the restricted version of the two-stage recombination operator with respect to SPEA (left) and NSGA-II (right) for a single run on the 3-750 MOKP instance.

7.3 Comparative results regarding the MOKP-specific version

Running tests have showed that the MOKP-specific version of the two-stage recombination operator also needs much more CPU time than that needed by the traditional crossover operators tackled for comparison (i.e.; the one-point crossover, the two-point crossover and the uniform crossover) but not so much as that needed by the restricted version for example. That is, we finally set the stopping condition to 100 generations for the MOKP-specific version of the two-stage recombination operator in order to have a CPU time comparable to that required by each of the three traditional crossover operators with 500 generations as stopping condition (see Table 28). Moreover, comparison between Table 4 and Table 29 shows that the CPU time corresponding to the MOKP-specific version of the two-stage recombination operator is less than (and hence comparable to) that corresponding to the one-point crossover, the two-point crossover and the uniform crossover, respectively.

Table 28. Stopping conditions for SPEA versions and NSGA-II versions

Algorithm version	Stopping condition
SPEA[one-point]	500 generations
SPEA[two-point]	500 generations
SPEA[uniform]	500 generations
SPEA[MOKP-two-stage]	100 generations
NSGA-II[one-point]	500 generations
NSGA-II[two-point]	500 generations
NSGA-II[uniform]	500 generations
NSGA-II[MOKP-two-stage]	100 generations

Table 29. Average CPU times (in seconds) corresponding to SPEA[MOKP-two-stage] and NSGA-II[MOKP-two-stage] with 100 generations as stopping condition

	MOKP instances								
	2-250	**2-500**	**2-750**	**3-250**	**3-500**	**3-750**	**4-250**	**4-500**	**4-750**
CPU time of SPEA[MOKP-two-stage]	7.20	24.10	63.50	20.86	61.46	131.93	47.76	140.50	274.83
CPU time of NSGA-II[MOKP-two-stage]	8.13	21.46	46.83	16.36	40.83	80.03	30.03	76.00	142.60

SPEA[MOKP-two-stage] is compared against SPEA[one-point], SPEA[two-point] and SPEA[uniform], respectively, using the coverage measure. NSGA-II[MOKP-two-stage] is also compared against NSGA-II[one-point], NSGA-II[two-point] and NSGA-II[uniform], respectively, using the coverage measure. The average results regarding the coverage measure (in percentage rate %) are reported in Tables 30-35. From Tables 30, 31, 33 and 34, one can easily see that the MOKP-specific version of the two-stage recombination operator significantly outperforms both the one-point and the two-point crossovers in both the SPEA and the NSGA-II contexts so that there is even a complete outperformance especially on the MOKP instances with a large number of items (in fact, 500 and 750 items, respectively). Table 32 shows that the MOKP-specific version of the two-stage recombination operator also significantly outperforms the uniform crossover in the SPEA context and this outperformance can be considered to be complete on the MOKP instances with a large number of items (in fact, 500 and 750 items, respectively). Compared to the uniform crossover in the NSGA-II context (see Table 35), the MOKP-specific version of the two-stage recombination operator provides the best coverage results on all the nine MOKP instances tackled in this paper. Moreover, the best convergence performance of the MOKP-specific version of the two-stage recombination operator in comparison with the uniform crossover in the NSGA-II context is also achieved on the MOKP instances with a large number of items (in fact, 500 and 750 items, respectively).

Table 30. Average coverage measures (%) corresponding to SPEA[MOKP-two-stage] and SPEA[one-point]

MOKP instance	*C*(SPEA[one-point], SPEA[MOKP-two-stage])	*C*(SPEA[MOKP-two-stage], SPEA[one-point])
2-250	0.23%	99.00%
2-500	0.00%	100%
2-750	0.00%	100%
3-250	0.01%	99.47%
3-500	0.00%	100%
3-750	0.00%	100%
4-250	0.00%	99.57%
4-500	0.00%	100%
4-750	0.00%	100%

Table 31. Average coverage measures (%) corresponding to SPEA[MOKP-two-stage] and SPEA[two-point]

MOKP instance	*C*(SPEA[two-point], SPEA[MOKP-two-stage])	*C*(SPEA[MOKP-two-stage], SPEA[two-point])
2-250	0.66%	97.92%
2-500	0.00%	100%
2-750	0.00%	100%
3-250	0.01%	99.41%
3-500	0.00%	100%
3-750	0.00%	100%
4-250	0.03%	98.65%
4-500	0.00%	100%
4-750	0.00%	100%

Table 32. Average coverage measures (%) corresponding to SPEA[MOKP-two-stage] and SPEA[uniform]

MOKP instance	*C*(SPEA[uniform], SPEA[MOKP-two-stage])	*C*(SPEA[MOKP-two-stage], SPEA[uniform])
2-250	1.07%	95.71%
2-500	0.00%	100%
2-750	0.00%	99.96%
3-250	0.12%	97.93%
3-500	0.00%	100%
3-750	0.00%	99.98%
4-250	0.21%	95.99%
4-500	0.00%	99.93%
4-750	0.00%	99.99%

Table 33. Average coverage measures (%) corresponding to NSGA-II[MOKP-two-stage] and NSGA-II[one-point]

MOKP instance	*C*(NSGA-II[one-point], NSGA-II[MOKP-two-stage])	*C*(NSGA-II[MOKP-two-stage], NSGA-II[one-point])
2-250	0.00%	99.72%
2-500	0.00%	100%
2-750	0.00%	100%
3-250	0.01%	99.49%
3-500	0.00%	100%
3-750	0.00%	100%
4-250	0.30%	92.85%
4-500	0.00%	99.86%
4-750	0.00%	99.78%

Table 34. Average coverage measures (%) corresponding to NSGA-II[MOKP-two-stage] and NSGA-II[two-point]

MOKP instance	*C*(NSGA-II[two-point], NSGA-II[MOKP-two-stage])	*C*(NSGA-II[MOKP-two-stage], NSGA-II[two-point])
2-250	0.00%	99.72%
2-500	0.00%	100%
2-750	0.00%	100%
3-250	0.28%	94.32%
3-500	0.00%	99.99%
3-750	0.00%	100%
4-250	0.75%	88.22%
4-500	0.00%	99.48%
4-750	0.00%	99.61%

Table 35. Average coverage measures (%) corresponding to NSGA-II[MOKP-two-stage] and NSGA-II[uniform]

MOKP instance	*C*(NSGA-II[uniform], NSGA-II[MOKP-two-stage])	*C*(NSGA-II[MOKP-two-stage], NSGA-II[uniform])
2-250	2.23%	91.95%
2-500	0.29%	98.51%
2-750	1.96%	91.84%
3-250	6.35%	59.73%
3-500	0.00%	99.74%
3-750	0.08%	97.33%
4-250	3.87%	70.64%
4-500	0.05%	96.22%
4-750	0.05%	90.80%

Furthermore, the average results corresponding to the range measure are provided in the following table (i.e.; Table 36). From this table and Tables 17-18, we can clearly see that the best range results are achieved by the MOKP-specific version of the two-stage recombination operator in comparison with all the three traditional crossover operators (in fact, the one-point, the two-point and the uniform crossovers) in both the SPEA and the NSGA-II contexts. This remark regarding the good diversity performance of the MOKP-specific version of the two-stage recombination operator against the one-point crossover, the two-point crossover and the uniform crossover, respectively, can be visually confirmed by comparing Figure 6 with Figure 1, Figure 2 and Figure 3, respectively.

Table 36. Average range measures corresponding to SPEA[MOKP-two-stage] and NSGA-II[MOKP-two-stage]

MOKP instance	*Range* corresponding to SPEA[MOKP-two-stage]	*Range* corresponding to NSGA-II[MOKP-two-stage]
2-250	3581.47	3649.47
2-500	6346.70	6410.80
2-750	10809.00	11204.40
3-250	5518.70	5526.63
3-500	10662.90	10815.40
3-750	16083.50	16181.30
4-250	7055.70	6956.47
4-500	14474.40	14449.20
4-750	22140.60	22038.90

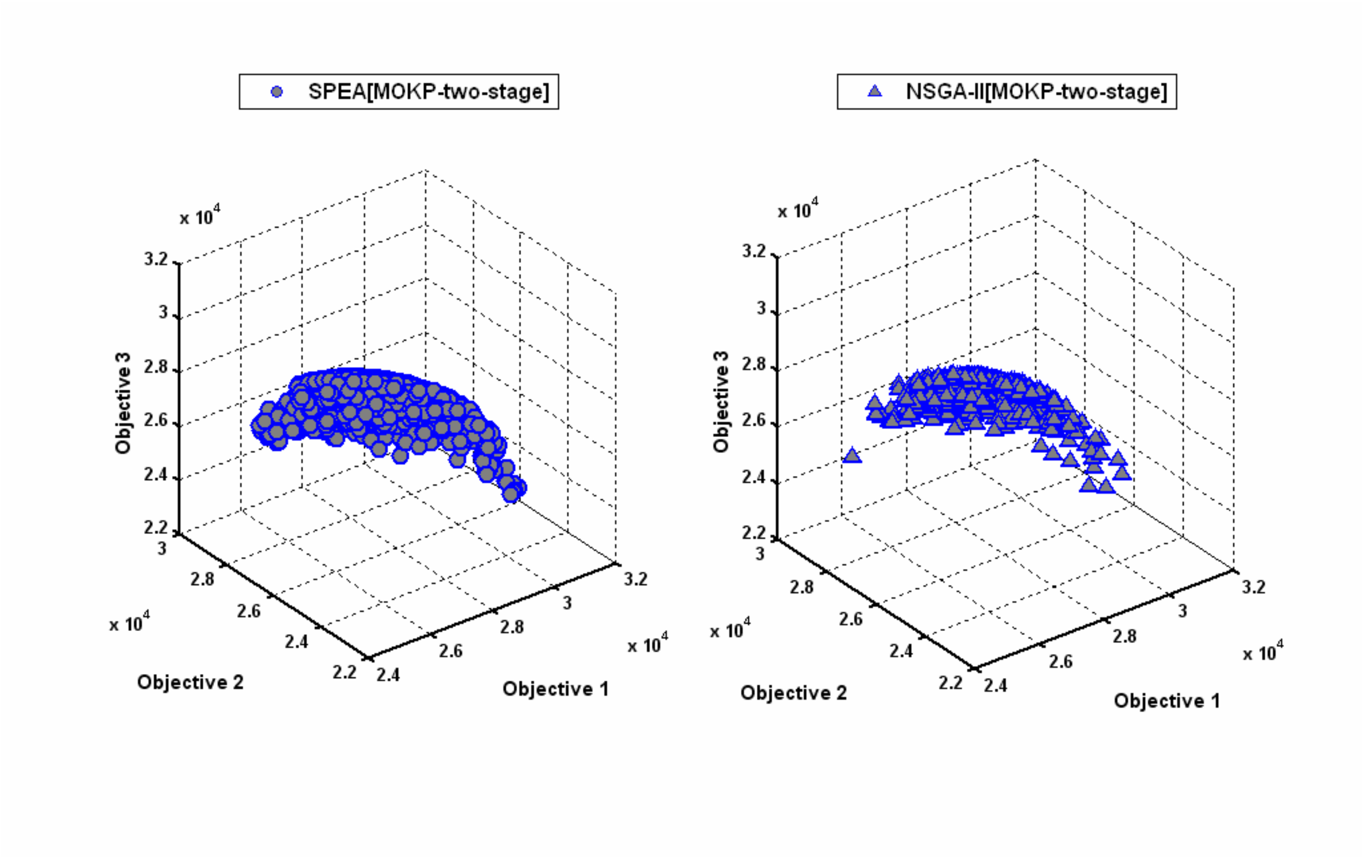

Figure 6. Solutions' distribution corresponding to the MOKP-specific version of the two-stage recombination operator with respect to SPEA (left) and NSGA-II (right) for a single run on the 3-750 MOKP instance.

8 Conclusions and future work

In this paper, we proposed a new recombination operator scheme called the two-stage recombination operator. Moreover, three different versions of the two-stage recombination operator (i.e.; the general version, the restricted version and the MOKP-specific version) have been proposed and applied to the MOKP. That is, each of these three versions has been compared to the one-point crossover, the two-point crossover and the uniform crossover, respectively, using two state-of-the-art MOEAs (SPEA and NSGA-II). Let us first summarize the concluding remarks regarding the convergence aspect. The experimental results showed that all the three versions of the two-stage recombination operator significantly outperform both the one-point and the two-point crossovers in terms of convergence in both the SPEA and the NSGA-II contexts. However, the convergence performance of the general version in comparison with the uniform crossover essentially depends on the number of generations: such a convergence performance is good if the number of generations is small and degrades as well as this number grows. As for the convergence performance of the restricted version in comparison with the uniform crossover, it depends on both the MOEA context and the MOKP instance: such a convergence performance is generally good on all the nine MOKP instances in the SPEA context and it is still rather good in the NSGA-II context but only on the MOKP instances having a high number of items and a small number of objectives (i.e.; 2 or 3 objectives). The convergence performance corresponding to the MOKP-specific version of the two-stage recombination operator is much better than that corresponding to the uniform crossover on all the nine MOKP instances in both the SPEA and the NSGA-II contexts.

We now move to the concluding remarks regarding the diversity aspect. The experimental results showed that the solutions' spread corresponding to the general version in the SPEA context is generally worst than that corresponding to the one-point crossover, the two-point crossover and the uniform crossover, respectively. The solutions' spread corresponding to the general version in the NSGA-II context is however better than that corresponding to both the one-point and the two-point crossovers but it is slightly worst than that corresponding to the uniform crossover. Thereby, the diversity performance of the general version is likely MOEA-dependent. However, the diversity performance (in terms of spread as well as in terms of distribution) corresponding to the restricted version (respectively the MOKP-specific version) is much better than that corresponding to the one-point crossover, the two-point crossover and the uniform crossover, respectively. We should note that the best compromise convergence-diversity is given by the MOKP-specific version which can be due to the fact that this version appropriately benefits from the particular structure of the MOKP. On the other hand, the usefulness of the two-stage recombination operator is yet to be further tested. That is, we are currently pursuing the appropriate adaptations of the proposed recombination operator to other MOCO problems. Additionally, we will examine the effects of changing some MOEA' standard components and parameters (e.g., the selection operator, the mutation probability, the recombination probability and the population size) on the two-stage recombination performance.

References

[1] K. Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley, 2001.

[2] K. De Jong. *Analysis of the behaviour of a class of genetic adaptive systems*. Ph. D. thesis, University of Michigan, Ann Arbor, 1975.

[3] G. Syswerda. Uniform crossover in genetic algorithms. *International Conference on Genetic Algorithms*, Vol. 3, pp. 2-9. Morgan Kaufmann, 1989.

[4] W. Spears. *The role of mutation and recombination in evolutionary algorithms*. Ph. D. thesis, George Mason University, Virginia, 1998.

[5] D. Schaffer and L. Eshelman. On crossover as an evolutionarily viable strategy. *International Conference on Genetic Algorithms*, Vol. 4, pp. 61-68. Morgan Kaufmann, 1991.

[6] J. J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann, pp. 42-60, 1987.

[7] R. Hill and C. Hiremath. Improving genetic algorithm convergence using problem structure and domain knowledge in multidimensional knapsack problems. *International Journal of Operational Research*, Vol. 1, N° 1/2, pp. 145-159, 2005.

[8] P. C. Chu and J. E. Beasley. A genetic algorithm for the multiconstraint knapsack problem. *Journal of Heuristics*, Vol. 4, N° 1, pp. 63-86, 1998.

[9] D. E. Goldberg. *Genetic algorithms for search, optimization, and machine learning*. Reading, MA: Addison-Wesley, 1989.

[10] A. Jaszkiewicz. *Multiple objective metaheuristic algorithms for combinatorial optimization*. Habilitation thesis, Poznan University of Technology, 2001.

[11] H. Ishibuchi, K. Narukawa, N. Tsukamoto and Y. Nojima. An empirical study on similarity-based mating for evolutionary multiobjective combinatorial optimization. To appear in *European Journal of Operational Research*, Vol. 188, pp. 57-75, July 2008.

[12] N. Durand and J.M. Alliot. Genetic crossover operator for partially separable functions. In *Genetic Programming Conference*, 8 pages, July 1998.

[13] A. Griewank and Ph. L. Toint. On the unconstrained optimization of partially separable functions. In *Nonlinear Optimization*. M. J. D. Powell, 1982.

[14] N. Durand. *Algorithmes génétiques et autres outils d'optimisation appliqués à la gestion de trafic aérien* (in french). Habilitation thesis, National Polytechnic Institute of Toulouse, 2004.

[15] H. Kellerer, U. Pferschy and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[16] C. A. Coello Coello, D. A. Van Veldhuizen and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, 2002.

[17] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, Vol. 3, pp. 257-271, 1999.

[18] J. D. Knowles and D. W. Corne. Approximating the nondominated front using Pareto Archived Evolution Strategy. *Evolutionary Computation Journal*, Vol. 8, pp. 149-172, 2000.

[19] A. Jaszkiewicz. On the performance of Multiple Genetic Local Search on the 0/1 knapsack problem: a comparative experiment. *IEEE Transactions on Evolutionary Computation*, Vol. 6, pp. 402-412, 2002.

[20] M. J. Alves and M. Almeida. MOTGA: A multiobjective Tchebycheff based genetic algorithm for the multidimensional knapsack problem. *Computers and Operations Research*, Vol. 34 (11), pp. 3458-3470, 2007.

[21] http://www.lania.mx/~ccoello/EMOO.

[22] B. Aghezzaf and M. Naimi. Un algorithme génétique pour le problème de sac à dos multidimensionnel multiobjectif (in french). *META*'06 *Conference*, Hammamet (Tunisia), November 2006.

[23] B. Aghezzaf and M. Naimi. Un algorithme hybride pour la résolution du problème de sac à dos multidimensionnel multiobjectif (in french). *Proceedings of the International Colloquium on Computer Science and Applications* (*IA* 2006), pp. 207-212. Oujda (Morocco), November 2006.

[24] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, Vol. 6, pp. 182-197, 2002.

[25] M. Ehrgott. *Multicriteria optimization*. Springer, second edition, 2005.
[26] E. L. Ulungu and J. Teghem. Multi-Objective Combinatorial Optimization Problems: A Survey. *Journal of Multi-Criteria Decision Analysis*, Vol. 3, pp. 83-104, 1994.
[27] H. Ishibuchi and T. Murata. A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, Vol. 28 (3), pp. 392-403, 1998.
[28] N. Srinivas and K. Deb. Multiobjective optimization using non dominated sorting in genetic algorithms. *Evolutionary Computation*, Vol. 2(3), pp. 221-248, 1994.
[29] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. Ph. D. thesis, Swiss Federal Institute of Technology, 1999.
[30] http://www.tik.ee.ethz.ch/~zitzler/testdata.html/.
[31] http://www-idss.cs.put.poznan.pl/~jaszkiewicz/.
[32] A. Jaszkiewicz. On the computational effectiveness of multiple objective metaheuristics. *Proceedings of the Fourth International Conference on the Multi-Objective Programming* (*MOPGP*'00). Theory & Applications, Springer-Verlag, June 2000.