International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 6 (2014) pp. 364 - 372
© MIR Labs, www.mirlabs.net/ijcisim/index.html

Best-Order Crossover in an Evolutionary Approach to Multi-Mode Resource-Constrained Project Scheduling

Anca Andreica[1] and Camelia Chira[2]

[1] Department of Computer Science,
Babes-Bolyai University
Kogalniceanu 1, 400084 Cluj-Napoca, Romania
anca@cs.ubbcluj.ro

[2] Department of Computer Science and
Centre for the Study of Complexity
Babes-Bolyai University
Kogalniceanu 1, 400084 Cluj-Napoca, Romania
cchira@cs.ubbcluj.ro

***Abstract*: Multi-Mode Resource-Constrained Project Scheduling is an NP-hard optimization problem intensively studied due to its large area of applications. This paper concentrates on the evolutionary approaches to Multi-Mode Resource-Constrained Project Scheduling based on permutation encoded individuals. A new recombination operator is presented and a comparative analysis of several recombination operators is given based on computational experiments for several project instances. Numerical results emphasize a good performance of the proposed crossover scheme which takes into account information from the global best individual besides the genetic material from parents.**

***Keywords*:** multi-mode resource-constrained project scheduling, evolutionary algorithm, crossover, permutation based encoding, best individual**.**

I. Introduction

Resource-Constrained Project Scheduling Problem (RCPSP) requires the allocation of limited resources to dependent activities over time, such that the makespan of the project is minimized. RCPSP has been intensively studied due to its large area of applications in industrial engineering, construction engineering, civil engineering, production and service industry. Multi-mode Resource-Constrained Project Scheduling Problem (MRPSP) represents a generalization of RCPSP, and therefore even more difficult to solve, in which activities can be in various modes affecting the resources needed.

It has been proven that RCPSP belongs to the class of NP-hard optimization problems [3]. Since exact solutions cannot be found in polynomial time by any algorithm, there is a high interest in developing good approximation methods for solving RCPSP able to determine a near-optimal (or optimal) solution using reasonable resources. Evolutionary computation provides good approximate methods for solving problems belonging to this class.

There have been several bio-inspired approaches to this problem among which a genetic algorithm with a new permutation of priority-based encoding scheme in [13], a permutation-based elitist genetic algorithm in [7], a genetic algorithm with two population and a mode optimization procedure in [10], an algorithm based on a priority value encoding scheme [12], a new hybrid genetic algorithm [Li, 2011]. An extensive description of all bio-inspired methods for solving RCPSP and MRCPSP is beyond the scope of this paper.

The role of the recombination operator during the evolutionary search of the MRCPSP solution is emphasized in this paper. Moreover, a new recombination scheme that uses genetic material from the best individual obtained by the search process besides genetic information from the parents is presented in this paper, together with a proof that the precedence feasibility of solutions is preserved in offspring if both parents and global best represent feasible solutions. Computational experiments are performed for several RCPSP and MRCPSP instances and results support a good performance of the proposed crossover scheme in comparison with well-known recombination operators in an evolutionary search.

The paper is organized as follows: Section 2 describes the Resource-Constrained Project Scheduling Problem, the evolutionary approach with an emphasis on chromosome representation is presented in Section 3, several recombination operators for MRCPSP found in the literature are described in Section 4, the new crossover in Section 5, experimental results in Section 6 and conclusions and directions for further research in Section 7.

II. Multi-Mode Resource-Constrained Project Scheduling

MRCPSP considers a project with a set of activities and a set of available resources. The *multi-mode* feature of the problem, which actually induces a generalization to the Resource-Constrained Project Scheduling Problem, refers to the fact that each considered activity can be in one of a set of modes and requires different resources and has different durations when finding itself in different modes. The goal is to find a schedule of the activities with minimum makespan, but considering both precedence and resources constraints. The problem is detailed and formalized in what follows.

Let J be the number of activities (also called jobs) and $\{a_1,...,a_J\}$ the set of activities. The set of possible modes for the activity a_j is denoted by $m_j \in M_j$, where $M_j = \{1,...,|M_j|\}$.

Jobs have to follow certain precedence constraints which means that a job can not start before all its predecessors are finished. Let us denote by P_j the set of all predecessors of activity a_j and by S_j the set of all its successors. The precedence constraints are usually represented as an acyclic activity-on-node network. Two additional activities are also normally considered: an initial activity a_0 , which must precede all other activities of the project and a final activity a_{J+1} which must be preceded by all activities of the project.

Figure 1 depicts an acyclic activity-on-node network for a project instance that we consider in order to better illustrate the Multi-Mode Resource-Constrained Scheduling Problem.

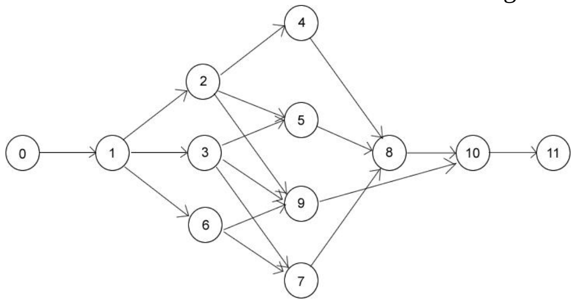

Figure 1. Acyclic activity-on-node network

If we denote by d_{jm_j} the duration of activity a_j in mode m_j , then the precedence relations can be written as:

$s_{j1} + d_{j1m_{j1}} \le s_{j2}, \ \forall a_{j2} \in S_{j1}$, (1)

where s_j represents the start time of activity a_j .

In addition to the set of activities, there is also a set of available resources. There are three types of resources:

- *renewable resources*, which are characterized by a constant per-period-availability
- *non-renewable resources*, whose availability covers the whole project duration
- *doubly-constrained resources*, which are limited both for each timestep and for the whole project

In order to be executed, each activity requires a certain amount of some of the resources. If we denote by R the set of renewable resources and by R_i the availability of resource $i \in \{1,...,|R|\}$, by N the set of non-renewable resources and by N_l the availability of resource $l \in \{1,...,|N|\}$, by DC the set of doubly-constrained resources and by DC_k the availability of resource $k \in \{1,...,|DC|\}$, then the limited availabilities of these types of resources could be written as:

$$\sum_{a_j \in A(t)} r_{ijm_j} \le R_i, \forall i \in R, \forall m_j \in M_j, \quad (2)$$

where r_{ijm_j} represent the quantity of resource R_i that activity a_j is using in mode m_j and $A(t)$ represent the set of activities taking place at timestep t;

$$\sum_{a_j} n_{ljm_j} \le N_l, \forall l \in N, \forall j \in \{1,...,J\}, \forall m_j \in M_j, \quad (3)$$

where n_{ljm_j} represent the quantity of resource N_l that activity a_j is using in mode m_j . As stated above, both types of constraints are applied to doubly-constrained resources *DC*.

It should be noted that both dummy activities (initial and final) require no time and no resources. The goal is therefore to minimize s_{J+1}. A feasible schedule is one that complies with both precedence and resource constraints. In order to illustrate a feasible schedule, let us consider the project of 10 activities described in Table 1. Each activity of the project can be in one of two modes, except the dummy activities which only have one mode characterized by 0 duration and no resources needed.

a_j	m_j	d_{jm_j}	r_{1jm_j}	r_{2jm_j}	n_{1jm_j}	n_{2jm_j}
0	1	0	0	0	0	0
1	1	6	0	10	1	1
	2	7	8	11	1	2
2	1	5	8	2	3	1
	2	4	8	0	2	2
3	1	4	4	9	3	0
	2	5	0	5	2	1
4	1	1	2	0	0	2
	2	4	5	4	1	1
5	1	3	8	7	2	3
	2	4	5	0	0	1
6	1	2	0	8	2	3
	2	4	0	8	1	1
7	1	3	6	6	3	0
	2	4	0	6	0	1
8	1	2	5	0	2	2
	2	2	6	3	1	0
9	1	10	5	9	1	1
	2	10	0	9	2	2
10	1	10	0	3	1	0
	2	11	4	4	0	2
11	1	0	0	0	0	0

Table 1. Project description

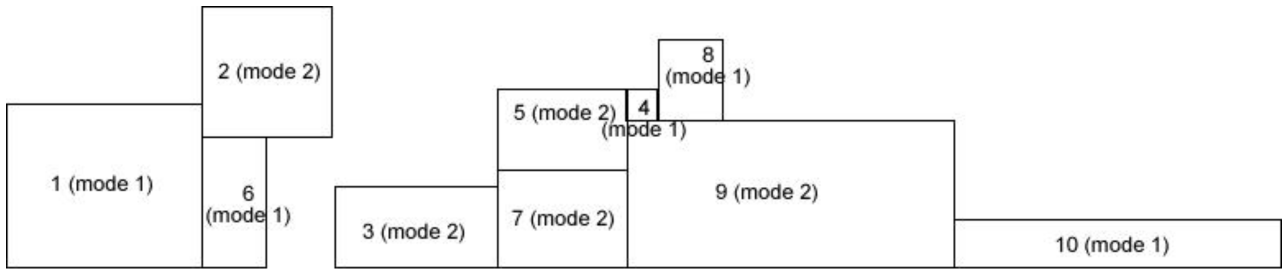

Figure 2. Project description

The precedence relations depicted in Figure 1 apply to the activities belonging to this project. Also considering the dummy activities results in the following list of activities: $\{0,1,...,10,11\}$. The time duration and amount of needed resources for each activity a_j in each of the possible modes are presented in Table 1.There are 2 renewable resources with the following per-period-availability: $R_1 = 8, R_2 = 11$ and 2 non-renewable resources with the following availabilities: $N_1 = 12, N_2 = 15$. A feasible schedule for this project is depicted in Figure 2.

III. Evolutionary Approach to Resource-Constrained Project Scheduling

Inspired by the process of natural evolution, evolutionary algorithms (EAs) are search heuristics widely used to generate useful solutions to optimization problems. A population of individuals (also called chromosomes) is used to represent the search space of the problem. Each individual encodes a candidate solution to the problem and an evaluation (fitness) function is used to assess its quality. EAs evolve a population of individuals towards better solutions based on mechanisms of selection, recombination and mutation [4].

A standard EA (SEA) is used to address the Multi-Mode Resource-Constrained Project Scheduling problem. Some of the key aspects of the algorithm will be described in this section.

A. Codification

The first thing to consider in an evolutionary approach is how a solution of the problem is to be encoded in a chromosome. There have been several different codifications proposed in the literature for the Resource-Constrained Project Scheduling problem, among which: activity list representation, random key representation, priority rule representation, shift vector representation and schedule scheme representation.

Our work focuses on the activity list representation which means that a solution of the problem is encoded as a list of the activities which represent their execution order. Moreover, if one activity a_2 appears after another activity a_1 in the activity list, it means that start time of activity a_2 is higher or equal to start time of activity a_1:

$s_{a_1} \le s_{a_2}$. (4)

The list of activities must be precedence feasible, meaning that each activity must have a higher index than each of its predecessors.

This representation is suitable for RCPSP but can be easily extended for MRCPSP [1]. The extension is done by also considering a list of modes, where each activity will have a corresponding mode (see Figure 3).

a_0	a_1	……	……	……	……	……	a_J	a_{J+1}
m_{a_0}	m_{a_1}	……	……	……	……	……	m_{a_J}	$m_{a_{J+1}}$

Figure 3. Chromosome codification (list of activities and corresponding modes)

When translating an activity list into a schedule, each activity is assigned the smallest possible start time, thus obtaining an active schedule [9]. This means that no activity can be left shifted without violating the constraints. In order to illustrate the activity list representation, let us consider the project described in Table 2. The list of activities and corresponding modes presented in Figure 4 is precedence feasible and can be translated into the active schedule depicted in Figure 2.

0	1	6	2	3	7	5	9	4	8	10	11
1	1	1	2	2	2	2	2	1	1	1	1

Figure 4. List of activities and corresponding modes for project described in Table 2

B. Initial population

Initial population is randomly generated in such a way that only precedence feasible individuals are obtained. Therefore, the first chromosome gene in the activity list is randomly selected from the entire activities set, and any other gene is randomly generated from the remaining set of activities only if all the predecessors of the chosen gene already exist in the chromosome configuration obtained at that point. A mode is randomly generated for each activity in the obtained list from all possible modes of the corresponding activity.

This mechanism does not ensure a resource feasibility of the chromosome. We are referring of course to non-renewable resources, because the renewable resources constraints can be always ensured by considering only non-overlapping activities.

Each non-feasible chromosome with respect to non-renewable resources is subject to a repair procedure presented in the next subsection.

C. Repair operator

Let us denote by *ERR (N)* the number of requested non-renewable resources that exceeded the capacity $N_l, l \in \{1,...,|N|\}$, the so—called *excess of resource request* [10]. *ERR (N)* is defined as in equation (5).

$$ERR(N) = \sum_{l=1}^{|N|}(\min(0, N_l - \sum_{j=0}^{J+1} n_{ljm_j})) . \quad (5)$$

A chromosome which contains a list of activities and corresponding modes with $ERR(N) > 0$ is not feasible with respect to non-renewable resources.

Repairing such a chromosome means taking each activity from the list and searching for a new mode that would improve *ERR(N)*. If such a mode is found, then it will replace the old mode. This mechanism ensures a possible improvement of the resource excess, but it does not ensure that only feasible solutions will be obtained. However, from our experiments, more than 50% of the chromosomes become feasible solutions after being subject to this reparation operator.

At the first glance, it might not be a good idea to keep unfeasible solutions in the population. However, these chromosomes will be penalized when computing their fitness, and therefore they will not be considered as relevant genetic material for the search process. Nevertheless, they will contribute to the increase of the population diversity.

D. Fitness function

There are two categories of potential solutions for the problem:

- feasible solutions with respect to non-renewable resources
- non-feasible solutions with respect to non-renewable solutions.

As stated before, the problem of feasibility with respect to renewable resources does not exist. Also, we only keep in the population solutions that are feasible with respect to precedence constraints. Therefore, from this point onwards, the term non-feasible chromosomes will refer to non-feasible solutions from the non-renewable resources point of view.

For the feasible solutions the fitness is given by the start time of the last dummy activity, which actually represents the makespan of the corresponding schedule. It is computed as the total duration of all activities, each activity acting in the mode given by the chromosome.

If we consider the chromosome *c* depicted in Figure 3, the fitness of such a feasible chromosome is given by:

$$Fitness(feasible\_c) = \sum_{j=0}^{J+1} d_{a_j m_{a_j}} . \ (6)$$

For the non-feasible chromosomes the fitness is computed as the maximum duration of all activities (by choosing the modes that ensure the maximum duration of each activity) plus the excess of resource request:

$$Fitness(non\_feasible\_c) = \sum_{j=0}^{J+1} \max(d_j) + ERR(N)\_c \ , \ (7)$$

where *ERR(N)_c* represents the excess of resource request for chromosome *c* and $\max(d_j)$ represents the maximum duration of activity *j*, considering all possible modes m_j.

E. Selection

Roulette selection is used for choosing which individuals should enter the mating pool. The best individual obtained in one generation will always replace one randomly generated individual from the next generation. This mechanism ensures that the best individual obtained in the last generation is actually the best individual obtained in all generations of the algorithm.

F. Recombination

Several different recombination operators will be considered for performing comparisons with the one proposed in this paper. These operators will be described in detail in section IV.

However, one feature that is common to all considered crossover operators is the fact that they do not change the modes of the activities when performing recombination between parents chromosomes. They do change the order in which activities appear in the activity list, but the mode of each activity is kept the same.

Another common feature of all considered recombination operators is that they all create precedence feasible solutions of the problem if the parents represent feasible solutions of the problem. Also, each obtained offspring will be subject to the repairing operator if the schedule it is encoding exceeds the available non-renewable resources.

G. Mutation

A simple mutation scheme is applied with a certain mutation probability. This operator swaps two genes of a chromosome only if the obtained permutation still represents a precedence feasible solution of the problem. Mutation obtains new individuals that recombination could not obtain.

Mutation is also responsible for changing the modes of the activities in the chromosome structure. Therefore, with the same probability of mutation, the mode of each activity can be changed to another mode from all possible modes of the corresponding activity.

The chromosome obtained after mutation will be subject to the repairing operator if it exceeds the available non-renewable resources.

IV. Crossover Operators

In order to test the efficiency of our proposed recombination operator, we consider some other crossover schemes proposed in the literature for permutation based encoding that preserves precedence relations between activities, for Resource-Constrained Project Scheduling. Three of the most popular crossover operators for permutation based encoding are described in what follows. They are similar to the general crossover operators described by [11] for permutation based encoding, but they have been slightly adapted so that they take precedence relations into account.

In the paper where these operators are proposed [6] the authors also present a proof that they all create feasible offspring if the parents represent feasible solutions of the problem.

A. One-Point Crossover

The first crossover operator is called one-point crossover. Let us denote by *P1* and *P2* the parents considered for recombination:

$$P1 = \{j_1^{P1}, \ldots, j_J^{P1}\},$$

$$P2 = \{j_1^{P2}, \ldots, j_J^{P2}\}.$$

An integer q is randomly chosen so that $1 \leq q < J$. The first offspring is obtained by taking from the first parent the entire sequence $\{j_1^{P1}, \ldots, j_q^{P1}\}$ and the rest of the genes are taken from the second parent so that the jobs that have already been taken from the first parent may not be considered again. This way the relative positions of the activities in the parents configuration is kept.

Let us consider the following example in order to illustrate the one-point crossover:

$$P1 = \{1,2,6,4,3,5,7\}$$

$$P2 = \{1,5,2,3,6,4,7\}$$

For *q=3* we obtain the following first offspring:

$$O1 = \{1,2,6,5,3,4,7\}$$

The second offspring is obtained is a similar way, by changing the role of the parents.

B. Two-Point Crossover

The two-point crossover represents an extension of the previously described recombination. Two integer numbers *q1,* q2 are randomly chosen so that $1 \leq g1 < q2 < J$.

The first offspring is obtained by taking from the first parent the sequence $\{j_1^{P1}, \ldots, j_{q1}^{P1}\}$, taking the next *q2-q1* positions from the second parent so that the jobs that have already been taken from the first parent may not be considered again and the rest of the positions $\{j_{q2}^{O1}, \ldots, j_J^{O1}\}$ again from the first parent, without considering the activities already taken at the first two steps.

For the two parents considered before:

$$P1 = \{1,2,6,4,3,5,7\}$$

$$P2 = \{1,5,2,3,6,4,7\}$$

For *q1=3* and *q2=5,* the first offspring would be:

$$O1 = \{1,2,6,5,3,4,7\}.$$

The other offspring is obtained in a similar way, by considering the second parent for the first and for the last sequence, and the first parent for the second sequence in offspring.

C. Uniform Crossover

For obtaining the first offspring by uniform crossover, a zero or a one is randomly generated for each position in the chromosome. If the position is 0, then the activity with the lowest index that has not been considered yet is copied from the first parent. Analogously, if the position is 1, then the activity with the lowest index that has not been considered yet is copied from the second parent.

For the two parents considered before:

$$P1 = \{1,2,6,4,3,5,7\}$$

$$P2 = \{1,5,2,3,6,4,7\}$$

and array {0,1,1,0,1,0,0}, the first offspring is:

$$O1 = \{1,5,2,6,3,4,7\}.$$

The second offspring is obtained in a similar way, by swapping the role of the two parents.

V. Proposed Best Order Recombination Scheme

An initial variant of the best-order crossover has been proposed by the authors in [5]. The obtained results proved the efficiency of the proposed crossover for the well known Traveling Salesman Problem. The best-order operator has been adapted for Resource-Constrained Project Scheduling, ensuring that obtained offspring represent feasible solutions of the problem. The proposed operator is described in what follows, together with a proof that feasible parents create feasible offspring.

A. Best-Order Crossover

The proposed Best-Order Crossover (BOX) is using information from the best individual obtained by the search process, besides using genetic material from the parents. We use this information in order to accelerate the search process by orienting it towards promising regions of the search space.

In order to be able to differentiate between good and bad genetic material, we have to decide what relevant genetic material means when considering permutation based encoding. In TSP for example, it is not important that a certain city has been visited at a certain moment of time, but rather the succession of visited cities, because the edges of a tour can be seen as the carriers of the genetic information. In RCPSP, the order of the activities is also important because of the precedence constraints.

The proposed BOX also exploits the fact that the order of the activities is important, not their positions. The main new feature of the proposed crossover operator is the use of genetic material belonging to the best individual together with genetic information from the two parents that are subject to recombination.

Several cutting points are randomly chosen. The number of cutting points is randomly selected and can be even zero. Every two consecutive cutting points (including the beginning and the end of the chromosome array) will generate a sequence of alleles; when the number of cutting points is 0, we will only have one sequence containing the whole chromosome. One of the following values will be assigned to each resulting sequence: -1, -2, -3. These values identify the source used for creating the offspring. A sequence identified by -1 means that the alleles will be taken from the main parent. A sequence identified by -2 means that the alleles will be taken from the other parent and when -3 is assigned to a sequence, the alleles will be taken from GlobalBest.

For example, in order to create the first offspring, we consider the first parent as the main parent. When we have a -1 sequence in offspring, we take the corresponding positions in the same order from the main parent. For a -2 sequence, we take the corresponding positions from the main parent but the order is given by the other parent. For a sequence identified by -3, we take the corresponding positions from the main parent but in the order imposed by GlobalBest.

In Figure 5, the sequence 4-5 is taken from Parent 1 is the same order, sequence 1-2-3 from the first parent is copied in Offspring 1 but the order is given by GlobalBest and the sequence 6-7-8 is also taken from the first parent but the order is given by the second parent.

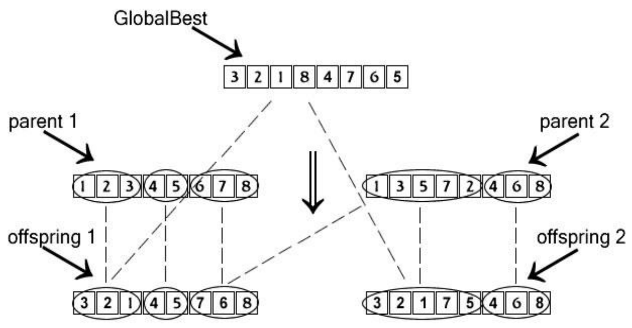

Figure 5. Best Order Crossover Operator

B. Feasible Solutions

THEOREM 1. The offspring obtained by BOX represent feasible solutions of the problem if the parents are also feasible solutions.

Proof: Let P1 ad P2 be two feasible parents:

$$P1 = \{ j_1^{P1}, ..., j_J^{P1} \},$$

$$P2 = \{ j_1^{P2}, ..., j_J^{P2} \}$$

and let $G = \{ j_1^{G}, ..., j_J^{G} \}$ be the global best obtained by the search process up to that moment, also a feasible solution.

Let us assume that the first obtained offspring O1 is not a feasible solution:

$$O1 = \{ j_1^{O1}, ..., j_J^{O1} \},$$

$$\exists j_i^{O1}, j_k^{O1} \text{ so that } 1 \le i < k \le J \text{ and } j_i^{O1} \in S_{j_k^{O1}} .$$

The following cases may appear:

i) both j_i^{O1} and j_k^{O1} belong to the same parent or both of them belong to the global best. This means that activity j_i^{O1} is before activity j_k^{O1} in that chromosome, which is a contradiction of the parents/global best feasibility.

ii) j_i^{O1} and j_k^{O1} belong to different parents or to one parent and to the global best. This means that activity j_i^{O1} is before activity j_k^{O1} in P1, because we take sequences of activities from P1 in an order given by P1, P2 or G, which is a contradiction of P1 feasibility.

VI. Experimental Results

The four recombination operators described in Sections 4 and 5 - One-Point Crossover (1PX), Two-Point Crossover (2PX), Uniform Crossover (UX), Best Order Crossover (BOX) - are engaged in a standard evolutionary approach to MRCPSP in order to see how the results are affected by the application of different crossover schemes. The parameters used by the evolutionary algorithm are given in Table 2.

Parameter	**Value**
Population size	100
Number of generations	100
Mutation probability	0.05

Table 2. Evolutionary algorithm parameters

For testing the performance of the proposed recombination scheme, several ProGen project instances have been considered [8]. ProGen is a Project Scheduling Problem Instance Generator of controlled difficulty, which is focused on precedence constrained and resource constrained project scheduling problems. Instances for both single-mode and multi-mode resource constrained scheduling problems are thus obtained.

Some results obtained for the single-mode variant of the problem (RCPSP) for projects with 30 activities have been presented in [2]. The experiments performed in this paper will include both more complex single-mode scheduling problems (the complexity being given by the increased number of activities), and multi-mode scheduling problems with a large number of activities.

For the single-mode variant of the problem we have considered 80 ProGen instances having between 30 and 120 activities. Table 3 presents the best and the average results over 10 runs of the standard evolutionary algorithm with all four recombination operators for each of 20 randomly chosen instances.

For the multi-mode variant, 400 ProGen instances with a number of activities varying between 10 and 30, have been considered. The results obtained for 20 randomly chosen instances are presented in Table 4. These results refer to the best and average results over 10 runs of the evolutionary algorithm with all four recombination operators for each considered problem instance.

The bold values in both Tables 3 and 4 represent the best solutions obtained for each instance. When using BOX, the algorithm is able to find the known solution of the problem in the highest percentage, compared to the other recombination operators. Also, if we take a look at the mean values obtained after 10 runs, it can be noticed that in most of the runs the result obtained by the algorithm using BOX is closer to the known solution of the problem compared to the other ones.

The convergence curve obtained in one run of the algorithm for a single-mode instance and one run for a multi-mode instance are depicted in Figures 6 and 7. It can be noticed that not only the final results are better when using BOX, but the solution is detected in fewer generations, which reduces the time and resources needed for detecting it. This behavior has been observed for most of the considered instances.

Therefore, the test results indicate the acceleration of the search process when using the proposed recombination operator.

	1PX		**2PX**		**UX**		**BOX**	
	Best	**Avg**	**Best**	**Avg**	**Best**	**Avg**	**Best**	**Avg**
j301_1	48	51	50	52	51	52	**43**	**47**
j301_8	55	**56**	55	**56**	56	59	**53**	**56**
j601_2	80	83	77	82	85	88	**74**	**80**
j601_4	93	**95**	93	96	93	98	**91**	96
j601_8	84	91	86	90	91	96	**82**	**87**
j601_9	90	**96**	94	97	97	98	**85**	**96**
j602_5	58	61	57	**60**	57	**60**	**55**	**60**
j901_1	95	99	100	103	100	103	**91**	**97**
j901_2	109	113	104	**112**	110	114	**101**	**112**
j901_4	99	**104**	101	105	109	110	**97**	105
j901_9	88	92	88	92	90	94	**86**	**89**
j902_2	119	125	119	123	119	122	**114**	**120**
j902_8	90	96	92	95	89	**92**	**86**	93
j1201_1	150	159	153	159	153	159	**141**	**151**
j1201_3	151	161	152	159	159	165	**147**	**154**
j1201_5	149	161	154	159	159	166	**139**	**153**
j1201_10	155	161	149	160	159	165	**139**	**154**
j1202_1	113	120	110	117	115	118	**107**	**112**
j1202_5	141	148	137	143	142	146	**133**	**140**
j1202_7	119	127	116	124	119	124	**112**	**120**
j301_1	48	51	50	52	51	52	**43**	**47**

Table 3. Best and average makespan obtained after 10 runs for single-mode instances

	1PX		**2PX**		**UX**		**BOX**	
	Best	**Avg**	**Best**	**Avg**	**Best**	**Avg**	**Best**	**Avg**
c1510_7	20	22	20	22	21	22	**18**	**21**
c2111_8	23	**24**	23	25	23	**24**	**22**	26
j1010_5	**24**	25	**24**	25	**24**	25	**24**	**24**
j1211_1	20	21	20	21	19	21	**18**	**20**
j1211_10	**22**	**22**	**22**	23	**22**	**22**	**22**	**22**
j1410_5	**20**	**21**	**20**	22	**20**	**21**	**20**	**21**
j1810_9	**39**	**42**	**39**	**42**	40	**42**	40	**42**
j2011_7	27	29	27	29	27	**27**	**26**	29
j3010_3	36	40	36	**38**	36	**38**	**33**	39
j3011_1	42	44	41	44	43	44	**39**	**41**
j3011_5	42	**43**	42	44	43	**43**	**40**	**43**
m411_9	22	**23**	22	**23**	22	**23**	**20**	**23**
m510_9	21	**22**	21	**22**	21	**22**	**19**	**22**
n310_7	28	30	28	30	28	**29**	**26**	30
n310_10	27	30	26	**29**	26	**29**	**24**	**29**
r110_10	11	13	12	13	11	**12**	**9**	**12**
r311_3	26	29	25	**27**	26	28	**24**	28
r510_7	30	32	30	32	29	32	**26**	**31**
r511_9	**29**	**30**	30	31	**29**	31	30	31

r511_5	27	**28**	27	**28**	27	**28**	**25**	**28**
c1510_7	20	22	20	22	21	22	**18**	**21**

Table 4. Best and average makespan obtained after 10 runs for multi-mode instances

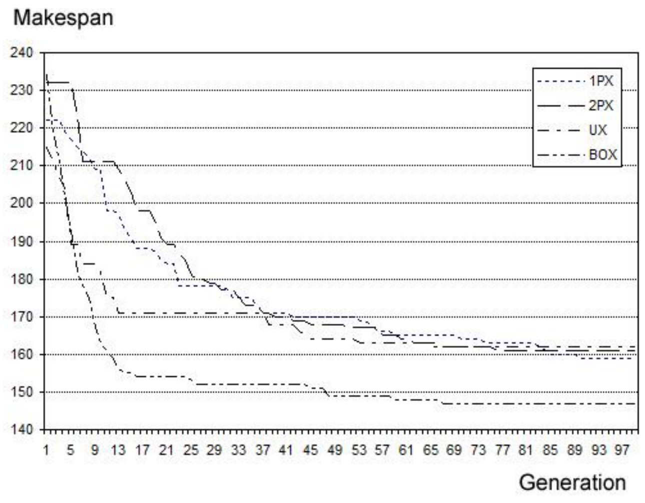

Figure 6. One run of the evolutionary algorithm for instance j1201_1. The results for each recombination operator are depicted over generations.

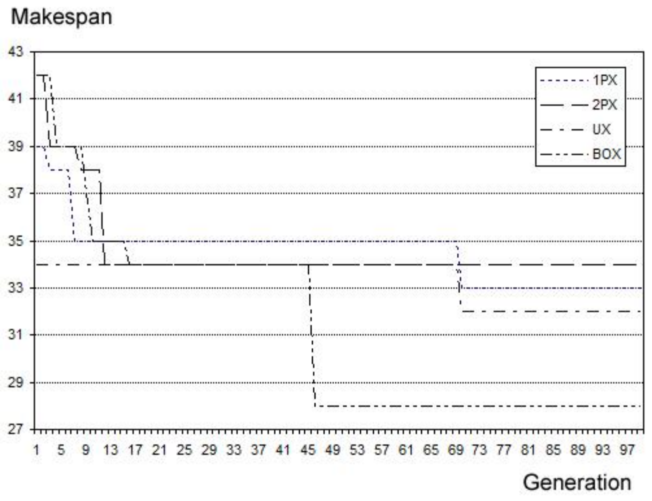

Figure 7. One run of the evolutionary algorithm for instance 510_7. The results for each recombination operator are depicted over generations.

The proposed operator does not require extra resources because an evolutionary algorithm usually keeps track of the best individual obtained by the search process at each time step.

VII. Conclusions

A new recombination operator for permutation based encoding has been proposed in this paper. This operator is suitable to Resource-Constrained Project Scheduling Problem and Multi-Mode Resource-Constrained Project Scheduling Problem because it preserves the precedence constraints when obtaining the offspring from feasible parents. The main feature of the proposed operator is the use of genetic information from the best individual besides the two parents considered for recombination.

Experimental results performed on ProGen project instances indicate a superior performance of the proposed operator, thus emphasizing the role that recombination has in accelerating the search in an evolutionary process. An extensive study of all recombination operators used for RCPSP and MRCPSP will be performed and the experiments will be extended to more instances with more activities.

Acknowledgments

This research is supported by Grant PN II TE 320, Emergence, auto-organization and evolution: New computational models in the study of complex systems, funded by UEFISCDI, Romania.

References

[1] Alcaraz, J., Maroto, C., A New Genetic Algorithm for the Multi-Mode Resource-Constrained Project Scheduling Problem, 2002.

[2] Andreica, A., Chira, C., The Role of Crossover in Evolutionary Approaches to Resource-Constrained Project Scheduling, Proceedings of the International Conference on Intelligent Systems Design and Applications (ISDA 2012), Kochi, India, pp. 200-205, 2012.

[3] Blazewicz, J., Lenstra, J., Rinnooy Kan, A., Scheduling subject to resource constraints: Classification and complexity. Discrete Applied Mathematics 5, pp. 11-24, 1983.

[4] Dumitrescu, D., Lazzerini, B., Jain, L.C, Dumitrescu, A., Evolutionary Computation, CRC Press, Boca Raton, FL., 2000.

[5] Gog, A., Chira, C., Recombination operators in permutation-based evolutionary algorithms for the travelling salesman problem, Chapter 10 in Logistics Management and Optimization through Hybrid Artificial Intelligence Systems, IGI Global, pp. 268-285, 2012.

[6] Hartmann, S., A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling, Naval Research Logistics 45, pp. 733-750, 1998.

[7] Kim, J.-L., Ellis, R.D., Permutation-Based Elitist Genetic Algorithm for Optimization of Large-Sized Resource-Constrained Project Scheduling, J. Constr. Eng. Manage. 134, 904, 2008.

[8] Kolisch, R., Schwindt, C., Sprecher, A., Benchmark instances for project scheduling problems; Kluwer; Weglarz, J. (Hrsg.): Handbook on recent advances in project scheduling, pp. 197-212, 1999.

[9] Kolisch, R., Serial and parallel resource-constrained project scheduling methods revisited: Theory and

computation. European Journal of Operational Research, 90, 320–333, 1996.

[10] Peteghem, V., Vanhoucke, M., A genetic algorithm for the Multi-Mode Resource-Constrained Project Sceduling Problem, working paper, 2008.

[11] Reeves, C.R., Genetic algorithms and combinatorial optimization. In V. J. Rayward-Smith, editor, Applications of modern heuristic methods, pages 111–125. Alfred Waller Ltd., Henley-on-Thames, 1995.

[12] Ren, Y.H., Kong, D.C., Peng, W.L., A Genetic Algorithm Based Solution with Schedule Mode for RCPSP, Advanced Materials Research, vol 268-270, pp. 1802-1805, 2011.

[13] Zhang, H., A Genetic Algorithm for Solving RCPSP, Computer Science and Computational Technology, 2008. ISCSCT '08. International Symposium on, pp. 246 – 249, 2008.

Author Biographies

Anca Andreica received the BSc degree in 2001, MSc degree in 2002 and PhD degree in 2007, all degrees in computer science from Babes-Bolyai University, Cluj-Napoca, Romania. Anca is currently assistant lecturer within the Department of Computer Science, Babes-Bolyai University, Romania. Her main research interests include evolutionary computing, nature-inspired metaheuristics and complex systems. Anca has published three books and over 50 papers.

Camelia Chira received the BSc degree in computer science from Babes-Bolyai University, Cluj-Napoca, Romania in 1998. From 2000 to 2005 she was engaged in full-time research at Galway-Mayo Institute of Technology (GMIT), Galway, Ireland focusing in the area of agent-based systems and ontologies for distributed collaborative design environments. She received the MSc and PhD degrees from GMIT Ireland in 2002 and 2005, respectively. Since 2006, Camelia is a researcher within the Department of Computer Science and Centre for the Study of Complexity, Babes-Bolyai University, Romania, currently working in the study and analysis of complex systems using cellular automata, complex networks and supporting artificial intelligence models. Her main research interests include computational intelligence, swarm techniques, complex systems and networks, machine learning, multi-agent systems and ontologies with applications ranging from distributed cooperative work, optimization, planning, scheduling, routing and logistics to social network analysis, data mining and bioinformatics. Camelia Chira has published over 100 scientific papers on these topics in various journals and conference proceedings, 5 book chapters and 2 books.