11e CONGRES INTERNATIONAL DE GENIE INDUSTRIEL – CIGI2015
Québec, Canada
26-28 octobre 2015

Resource Constrained Project Scheduling Problem with sequence dependent Transfer Times: The single mode case

Roubila Lilia KADRI and Fayez F. BOCTOR

Centre Interuniversitaire de Recherche sur le Réseau d'Entreprises, la Logistique et le Transport (CIRRELT)
Faculté des sciences de l'administration, Université Laval
2325 rue de la Terrasse, Québec, Canada, G1V0A6

Abstract: In this paper we address the Resource-Constrained Project Scheduling Problem with sequence dependent Transfer Times (RCPSPTT). We assume that pre-emption is not allowed, and precedence relations are zero-lag, finish-to-start relations. Also, we assume that activity durations and resource transfer times are known and deterministic. The objective is to choose a start time for each activity of the project such that the project duration is minimized while satisfying precedence relations, resource availabilities and resource-transfer time constraints. This paper proposes a new genetic algorithm using a modified magnet-based crossover operator (MMBX). Experiment conducted on a large number of instances shows that the proposed algorithm produces better results than the previously published solution methods.

Keywords - project scheduling, resource transfer time, genetic algorithms.

1 INTRODUCTION :

The standard version of the *Resource-constrained, project scheduling problem* (RCPSP) consists of scheduling a set of activities (j=1, .., J) linked by zero-lag, finish-to-start precedence relations while pre-emption is not allowed. Resources required to perform activities are renewable and available in limited amounts at each period. Each activity has a unique and known duration and uses a constant amount of the resources for each period of its execution. The objective of the RCPSP is to minimize the project duration (make-span) while respecting precedence and resource constraints. This problem is known to be NP-hard as shown by Blazewicz et al. (1983).

To solve the RCPSP to optimality, various branch and bound procedures have been developed (Patterson et al. 1990; Carlier and Latapie. 1991; Demeulemeester and Herroelen.1996; Brucker et al. 1998). However, none of this technique is capable of solving large size realistic problem in reasonable computational time.

Hence, several authors propose heuristic approaches to solve the RCPSP. Most of these methods belong to two categories, priority rule based heuristics and neighbourhood search heuristics. The first category are construction heuristics which start with an empty schedule and add activities one by one until all activities are scheduled. This operation is controlled by a scheduling scheme and a priority rule. Two different scheduling schemes called serial and parallel (Kelley, 1963; Brooks and White, 1965) are widely used to develop most heuristic techniques published in the literature (Boctor, 1990; Kolisch, 1996a; Kolisch and Drexl, 1997). In order to improve a feasible schedule obtained by some constructive heuristic, various neighbourhood search methods are developed like Simulated annealing (Boctor, 1996; Bouleimen and Lecocq, 2003), Tabu search (Baar et al. 1999; Nonobe and Ibaraki, 2002), Scatter search (Debels et al. 2006; Mobini et al. 2009) and Genetic algorithm (Alcaraz et al. 2003; Hartmann, 2001; Valls et al. 2004; Mendes et al. 2009; Zamani, 2013).

The standard RCPSP as defined above assume that all activities are performed in the same location (site), so no transfer times are needed to transfer resources from one site to another. In practice, however, it happens that activities have to be performed in several locations. Consequently, project resources have to be transferred between these locations. For example, heavy construction equipment might be requested at different construction sites and moving them from one site to another may take a significant time.

In such a case, modelling and solving the project scheduling problem without taking into consideration transfer times might produce unfeasible schedules. Recently Krüger and Scholl (2009) developed methods to solve the *resource-constrained project scheduling problem with transfer times* (RCPSPTT) for the single and multi-project cases. The authors formulate both problems as integer linear models and developed a priority rule based heuristic solution approach. Then Krüger (2009) developed a genetic algorithm for this problem. Recently, a flow based tabu search algorithm for the RCPSP with transfer times was developed by Poppenborg and Knust (2014). Hereafter we

propose a new and efficient genetic algorithm to solve the RCPSPTT.

The remainder of this paper is organized as follows. Section 2 describes the Resource constraint project scheduling problem with transfer times and section 3 presents a mathematical formulation for it. Section 4 describes the proposed genetic approach and section 5 reports and discusses the results of the conducted evaluation experiment. Section 6 concludes the paper and opens on possible extensions.

2 PROBLEM DESCRIPTION

In the RCPSPTT we have a set J of activities, a set P of zero-lag, finish-to-start precedence relations between these activities, a set of renewable resources K. The limits on resources are denoted Q_k; $k \in K$. Each activity i has a set of immediate predecessors $P_i \subseteq J$, a set of immediate and indirect predecessors, denoted π_i, a set of immediate successors $Z_i \subseteq J$ and a set of immediate and indirect successors, denoted σ_i. Each activity has a unique and known duration d_i and uses a constant amount of resources q_{ik}; $\forall k \in K$, for each period of its execution. We assume that activities are to be performed in a number of different locations (sites). We also define $I_i = [EST_i, LST_i]$, the time window between the earliest start time and the latest start time of activity i. These times are calculated using the critical path method while using feasible project duration denoted T. The transfer of a unit of renewable resource k from the execution location of activity i to the execution location of activity j requires a transfer time denoted Δ_{ijk}. All transfer times are assumed to fulfil the triangular inequality; i.e., $\forall i,j,l \in J$, $\Delta_{ijk} \leq \Delta_{ilk} + \Delta_{ljk}$.

Without loss of generality we assume that project activities include a dummy start activity, activity 1, and a dummy finish activity, activity F. These two activities have duration zero and require the entire available amount Q_k of the every resource k; (i.e., $q_{1k} = q_{Fk} = Q_k$). This is because the dummy start activity should provide resources to the other activities while the dummy finish activity should collect them back. Transfer times from the dummy start activity and to the dummy finish activity are assumed nil (i.e., $\Delta_{1ik} = \Delta_{iFk} = 0$; $\forall i \in J$, $k \in K$).

The objective in RCPSPTT is to construct a schedule that minimizes the project duration while satisfying precedence, resource and transfer time constraints.

Figure 1 shows an example of a project comprising 9 activities which have to be scheduled using one resource with eight available units and we assume that project activities have to be performed in different locations (sites). Consequently, transfer times Δ_{ij} are needed to transfer units of the resource from the location of i to the location of j. The transfer time matrix is also given in Figure 1.

The optimal schedule of this example with a make-span of 13 time-periods is shown in Figure 2 where arrows indicate transfer of resource units.

3 MATHEMATICAL FORMULATION

In addition to the notation presented above, we need to use three decision variables, first we use a binary variables x_{it} that take the value 1 if and only if activity i starts at the beginning of time period t. we also use another binary variable, y_{ijk}, that takes the value 1 if and only if at least one unit of resource k is transferred from activity i to activity j. Finley, f_{ijk} gives the number of units of resource k to transfer from activity i to activity j.

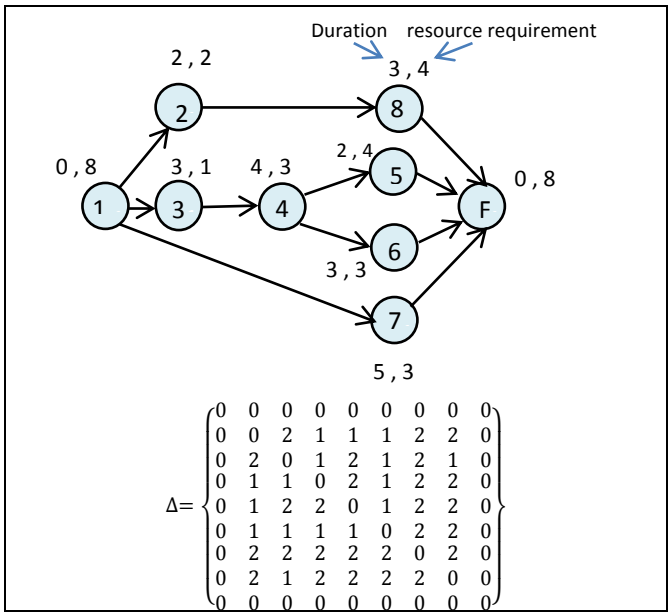

Figure 1 : a RCPSPTT example.

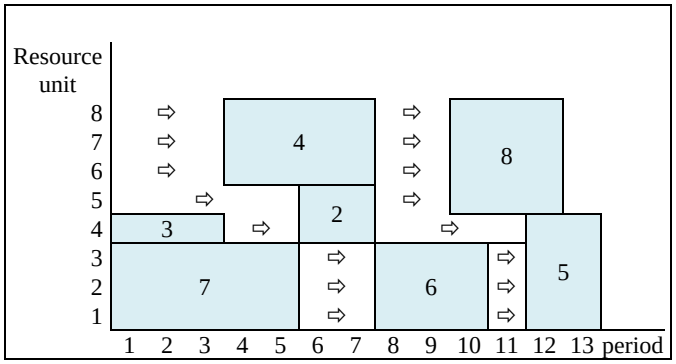

Figure 2: Optimal solution of the RCPSPTT example

Using this notation the mathematical model of the RCPSPTT can be written as follows (also see Krüger and Scholl, 2009):

$$Minimize: \sum_{t \in I_F} t\, x_{Ft} \quad (1)$$

Subject to:

$$\sum_{t \in I_i} x_{it} = 1 \quad ; \quad \forall i \in J \quad (2)$$

$$\sum_{t \in I_j} t.x_{jt} - \sum_{t \in I_i} (t + d_i) x_{it} \geq 0 \quad ; \forall (i,j) \in P \quad (3)$$

$$\sum_{t \in I_j} t.x_{jt} - \sum_{t \in I_i} (t + d_i) x_{it} - (T + \Delta_{ijk}).y_{ijk} \geq -T \quad ; \forall i \in J - \{F\}, \forall j \in J - \pi_i \quad (4)$$

$$f_{ijk} \leq \min(q_{ik}, q_{jk}).y_{ijk} \quad ; \forall i \in J - \{F\}, \ \forall j \in J - \pi \quad (5)$$

$$\sum_{i \in J - \{F\}} f_{ijk} = q_{jk} \quad ; \forall j \in J - \pi_i \quad (6)$$

$$\sum_{j \in J - \pi_i} f_{ijk} = q_{ik} \quad ; \forall i \in J - \{F\} \quad (7)$$

$$y_{ijk} \in \{0,1\} \quad ; \forall k \in K, \forall i \in J - \{F\}, j \in J - \pi_i \quad (8)$$

$$x_{it} \in \{0,1\} \quad ; \forall i \in J, \forall t \in I_i \quad (9)$$

The objective function (1) minimizes the project duration. Constraints (2) ensure that each activity is executed without pre-emption, and started within the time window between its earliest and latest start time. Constraints (3) are the precedence constraints and constraints (4) ensure that the transfer time of the resource k is taken into consideration when some of its units are transferred from activity i to activity j. Constraint (5) imply that if y_{ijk}=0 then f_{ijk}=0. Otherwise f_{ijk} takes a value less than or equal to both the number of units required to execute the activity j and the number of units available at i. Constraints (6) and (7) are the resources flow conservation constraints. Constraints (8); (9) and (10) define the decision variables of the model.

4 THE PROPOSED GENETIC ALGORITHM

In this section we develop a new genetic algorithm approach to solve the RCPSPTT. Our GA starts with an initial population generated through a modified version of the regret-based biased random sampling mechanism proposed by Kolisch and Drexl (1997). The size of the population is constant and equal to G. The genetic algorithm then determines the fitness value of each individual, randomly selects a number of pairs of individuals (solutions), and uses a Modified version of the Magnet-Based Crossover operator (called hereafter MMBX) to generate two new individuals (offspring solutions) from each selected pair (see Zamani; 2013, for the details of the original version of the magnet based crossover operator). Afterwards a mutation operation is applied to some of the new individuals.

After computing the fitness of each new individual, we add offspring individuals to the current population. Then among the G parent individuals and G new individuals, the best G individuals are selected and placed in the next generation. This process is repeated until the total number of solutions generated by the algorithm reaches a pre-selected number L. The pseudo-code of the algorithm is presented in Figure 3 below.

1. Generate an initial population of G individuals using regret-based biased random sampling
2. $N_{solutions} = G$
3. WHILE ($n_{solutions} \leq L$):
- Generate G offspring individuals from current population using MMBX.
- Apply mutation to some of the generated individuals.
- Determine the fitness (project duration) of each children.
- $N_{solutions} = N_{solutions} + G$
- Add the new G solutions to the previous generation of G solutions.
- Select the best G individuals to be the next generation.

END WHILE
4. Retain the best obtained solution

Figure 3: The developed GA.

4.1 Solution coding

The genetic algorithm described in this paper uses an extended activity list representation, where each individual is represented (coded) by |J|+2 genes as shown in Figure 4. The first |J| genes give a precedence feasible activity list (sequence) where each activity is placed after all its predecessors. After that, two extra binary genes indexed |J|+1 and |J|+2 are added. The first one indicates the scheduling generation scheme used to generate the corresponding feasible schedule. In this work the serial (S) and parallel (P) scheduling schemes adapted by Krüger and Scholl (2009) to the RCPSP with transfer times are used to construct the schedule (these two algorithms are presented in subsection 4.3). The second binary gene indicates the scheduling direction (forward or backward F/B) to be used while generating the schedule.

1	i_1	i_2	…	i_j	…	F	binary	binary
← Precedence feasible activity list →							0=Serial or 1=Parallel	0=Forward or 1=Backward

Figure 4: Solution code (representation).

4.2 Generation of the initial population

The first step of the genetic algorithm consists of generating an initial population of G individuals. As mentioned before, each individual is composed of an ordered activity list (activity sequence), a binary indicating the type of heuristic (serial or parallel) to be used to translate the list into a feasible schedule, and another binary indicating the direction (forward or backward) to be followed while building this schedule. The ordered activity list is obtained by a modified version of the regret-based biased sampling heuristic described in Figure 5 below. Within this heuristic, three priority rules are used: the LFT (latest finish time), LST (latest start time) and the MINSLK (minimum total slack) rules. This biased sampling method is an iterative heuristic that iteratively determines the set of eligible activities (whose predecessors are already included in the sequence under construction) and randomly selects and adds one of them to the sequence. The probability of selecting an activity j equals $\psi_j = \frac{r_j}{\sum_{i \in S} r_i}$, where r_j is called the regret value of j and depends on its priority. This process is repeated until a complete precedence-feasible sequence is obtained. Afterwards, two binaries are randomly generated and added to complete the code representing the individual.

1. Initialize the activity sequence : $\sigma = \{1\}$
2. Randomly choose a priority rule among the three rules (LFT, LST and MINSLK)
3. Let S be the set of eligible activities; $S = \{\forall i \in J \mid P_i = \{1\}\}$
4. WHILE $S \neq \emptyset$
- Calculate p_j, $\forall j \in S$, the priority of each eligible activity according to the selected rule
- For each eligible activity j, calculate r_j, its regret value, and ψ_j its selection probability, where:
  $r_j = 1 + (\max_{i \in S} p_i) - p_j$ and $\psi_j = \frac{r_j}{\sum_{i \in S} r_i}$
- Based on the probability ψ_j and using the simple roulette method, select an activity among the eligible ones. Let k be the selected activity. Add k to the sequence; $\sigma = \sigma + \{k\}$
- Update S, the set of eligible activities; $S = \{\forall i \in J | P_i \subseteq \sigma\}$

END WHILE

Figure 5: Modified-regret based biased random sampling procedure.

4.3 Transforming a sequence into a schedule

Once an individual (coded by a sequence of activities and the two additional binaries as shown in Figure 4) is generated, we need to deduce a corresponding feasible schedule and to calculate its fitness value. The fitness value of a solution is given by the duration of the corresponding schedule.

To construct a feasible schedule we use either the serial or the parallel scheduling scheme (Kelley 1963) as adapted from Krüger and Scholl (2009). Before we present these two scheduling schemes let us introduce the following notation:

x_{ijk}	Quantity of resource k transferred from activity i to activity j
y_{ik}	Number of units of k available at the site of activity i
D	Set of activities scheduled to start by the beginning of t
C	Set of activities scheduled to finish by the beginning of t
A	Set of activities in progress at period t
FT_i	Finish time of activity i
Z_{jk}	Set of activities that can provide units of k to activity j
Γ_{jk}	Set of pair of activities (i,l) such that i provide units of k to l and that activity j can be placed in between and uses some of these units of k before passing them to l

The serial scheduling scheme

The serial scheduling scheme is presented in Figure 6. This procedure sequentially adds activities to the schedule until a complete feasible schedule is obtained. First we arrange the list of activities using a selected priority rule (e.g., MINSLK, LFT or LST). Then, in each iteration, the next activity j in the arranged list of activities is selected and scheduled at the earliest possible time. All activities i which precede activity j in the list which can deliver at least one unit of resource type k to activity j at time t or earlier are considered and added to a set denoted Z_{jk}.

Also, if activity j is inserted between two already scheduled activities, say activities i and l, the resource flow between these two activities can be broken to deliver some resource units from i to j which transfer them later to l. However, it must be ensured that the breaking flow between i and l will not cause any delay for activity l. So, within the procedure, all breakable flows are identified and the corresponding pairs of activities i and l are added to a set denoted Γ_{jk}. Thus providing the required units of resource k to j at time t can be either from activities belonging to Z_{jk} or by breaking some flows between pairs of activities belonging to $\Gamma_{jk.}$ The amount of resource k that can be provided to j at period t, denoted s^{jk}, is $s^{jk} = \sum_{i \in Z_{jk}} y_{ik} + \sum_{(i,l) \in \Gamma_{jk}} x_{ilk}$. If enough resource units are available for activity j; i.e., when $s^{ik} \geq q_{jk}; \forall k \in K$, then activity j can be scheduled to start at the beginning of period t.

Activities belonging to Z_{jk} are arranged in the ascending order of transfer time to activity j while pairs of activities (i,l) belonging to Γ_{jk} are ordered in the descending order of x_{ilk} . To satisfy the demand of activity j for resource k, the delivering activity with minimum transfer time Δ_{ijk} is selected first and the receiving activities with maximum flow x_{ilk} are prioritized. Then, the current breakable flow x_{ilk} from i to l is broken up and activity j receives the maximum quantity from activity i and, once finished, delivers the same amounts back to activity l.

If s^{jk} the transferable amount of resource k is not sufficient to allow starting activity j at the beginning of time period t then t is increased until sufficient resources become available. The procedure stops when a complete schedule is constructed.

The parallel scheduling scheme

The parallel scheduling scheme is presented in Figure 7. It is a straight forward adaptation of parallel scheduling scheme for the standard resource-constrained project scheduling problem proposed by Kelley (1963) and presented by Brooks and White (1965). This iterative procedure identify the candidate activities, arrange them using a preselected priority rule and schedule the activity having the highest priority that can be scheduled with the available amount of resources. Then it updates the list of candidate activities and if the list is not empty proceeds to schedule another activity. In case where the list is empty but there is still some non-scheduled activities, the procedure moves the earliest date where some activities can be scheduled and select one of them. The procedure stops once all activities are scheduled.

4.4 Modified Magnet-Based Crossover operator (MMBX)

The uniform crossover (Bierwirth et al. 1996) and the k-point crossover operators (Djerid et al. 1996) were suggested to handle scheduling problems. However, the one point crossover (Davis 1985) and two-point crossover (Hartmann 1998) are the usually used crossover operators in solving the standard RCPSP. Recently Zamani (2013) introduced new operator and called it Magnet-Based Crossover operator or MBX. The genetic algorithm developed in this paper modifies the MBX and uses this modified version, called MMBX, to generate offspring solutions.

The MBX operator suggested by Zamani works as follows. First a contiguous block of genes is selected from the first parent, called the donator, by randomly drawing two numbers n_1 and n_2 ($1<n_1<n_2<|J|$) which become the bounding positions in the donator sequence specifying the required contiguous block. Then we determine the position of the first element belonging to the block appearing in the sequence of the second parent, called receiver, as well as the position of the last element of the block appearing in the sequence of the receiver. Let these position be denoted n_0 and n_3 respectively. The offspring sequence is built by concatenating: activities in positions 1 to n_0-1 in the receiver sequence; activities in positions between n_0 and n_3 in the receiver sequence which are predecessors to any activity in the block; activities in the block; activities in positions between n_0 and n_3 in the receiver sequence who are successors to any activity in the block; and activities in positions between n_3+1 and $|J|$ in the receiver sequence. However, some of the activities in the positions between n_0 and n_3 in the receiver sequence may neither be a predecessor or a successor to any activity in the block. These activities are called free activities and are randomly placed either before or after the activities of the block in the offspring sequence. Let q be the number of free activities, Zamani suggest placing the first p free activities before the block activities and the remaining (q-p) free activities after; where p is randomly determined.

Initialize: $D = \{1\}$
$FT_i = 0;\ \forall i \in J$
$x_{ijk} = 0;\ \forall i \in J - \{F\}, \forall j \in J - \pi_i$
$y_{1k} = Q_k;\ \forall k \in K$
$y_{ik} = 0;\ \forall i \in J - \{1, F\},\ \forall k \in K$
Using the selected priority rule (MINSLK, LFT, or LST), generate a feasible activity list (*Sequence*)
FOR λ=2 to $|J|$
Let j be the activity in position λ in the sequence, calculate:

$$t = max\left(EST_j, max_{i \in P_j} FT_i\right)$$

WHILE $j \notin D$
FOR k =1 **to** $|K|$
$Z_{jk} = \varnothing$
FOR l =1 **to** λ-1
Let i be the activity in position l in the sequence
$\Gamma_{jk} = \varnothing$
IF $y_{ik} > 0$ **AND** $FT_i + \Delta_{ijk} \le t$ **THEN** $Z_{jk} = Z_{jk} \cup \{i\}$
FOR m = 1 to $|J|$
IF $x_{imk} > 0$ **AND** $FT_i + \Delta_{ijk} \le t \le FT_m - d_j + d_m - \Delta_{jmk}$
THEN $\Gamma_{jk} = \Gamma_{jk} \cup \{(i,m)\}$
NEXT m
NEXT l
$s^{jk} = \sum_{i \in Z_{jk}} y_{ik} + \sum_{(i,l) \in \Gamma_{jk}} x_{ilk}$
NEXT k
IF $s^{jk} \ge q_{jk},\ \forall k \in K$ **THEN**
For k =1 **to** $|K|$
Arrange all breakable-flow activities pairs $(i,l) \in \Gamma_{jk}$ in the descending order of x_{ilk}
Arrange all potential delivering activities $i \in Z_{jk}$ in the ascending order of their transfer time Δ_{ijk}
$u_k = 0$
WHILE $u_k < q_{jk}$ **AND** $\Gamma_{jk} \neq \varnothing$
Starting from the first down to last pair $(i,l) \in \Gamma_{jk}$
$x = min\ (x_{ilk}\ ,\ q_{jk} - u_k)$
$x_{ilk} = x_{ilk} - x$
$x_{ijk} = x_{ijk} + x$
$x_{jlk} = x_{jlk} + x$
$u_k = u_k + x$
END WHILE
WHILE $u_k < q_{jk}$
Starting from the first down to last activity $i \in Z_{jk}$
$x = min\ (y_{ik}\ ,\ q_{jk} - u_k)$
$y_{ik} = y_{ik} - x$
$x_{ijk} = x_{ijk} + x$
$u_k = u_k + x$
END WHILE
NEXT k
$D = D \cup \{\ j\ \}$, $FT_j = t + d_j$,
ELSE
$t = t + 1$
END IF
END WHILE
NEXT λ

Figure 6: Serial Scheduling procedure for the RCPSPTT.

The two additional binaries (indicating if a serial or parallel scheme is to be used and the direction of applying it; Forward or backward) are copied from the code of the donator solution.

In our MMBX operator, free activities are randomly placed either before or after the activities of the block as long as this does not violate any precedence relation. This is implemented as follows. For every free activity we determine all its possible positions that do not lead to violating precedence relations and we randomly chose one of these positions. This modification of the MBX operator makes it slightly more disruptive than the original one and thus introduces more diversification in the process. Our experiment shows that the MMBX operator produces slightly better results than the original MBX.

To illustrate the application of the MMBX operator described here, we use the numerical example presented in Figure 1. Two feasible solutions are presented in top of Figure 8 and are considered as the donator and receiver parents respectively. Assuming that we randomly draw n_1=6 and n_2=8 then the contiguous block to transfer from the donator parent to the offspring is composed respectively from activities 4, 5 and 6. Consequently, the positions n_0 and n_3 in the receiver parent are respectively position 3 (where we have activity 4) and 8 (where we have activity 6). Now we can start building the activity sequence for the offspring as follows. We copy from the receiver all activities between positions 1 and 2 (i.e., between 1 and position n_0-1) as well as the activity in position 9 (i.e., position n_3+1 which is also the last position in the sequence). Activities of the block are then placed in between. The three remaining activities {2, 8 and 7} are free activities as they are neither predecessors nor successors to any activity in the block. These free activities are then added to offspring sequence one by one as follows. Activity 2 is an immediate predecessor of activity 9 and immediate successor to activity 1. Consequently activity 2 can be placed in the offspring sequence either between activity 3 and activity 4 (the first activity in the block) or between activity 6 (the last activity in the block) and activity 9. Assume that we randomly decided to place it between activities 3 and 4.

Now, as activity 8 is an immediate successor of activity 2 and immediate predecessor of activity 9, it can be located either between activities 2 and 4 or between 6 and 9. Assume that we randomly placed it between activities 2 and 4. Four positions are now possible for activity 7 as shown in Figure 8. If we randomly place activity 7 between 6 and 9 and we copy the two last binaries from the donator we obtain the offspring code {1-3-2-8-4-5-6-7-9-S-F} as shown in the figure.

4.5 Mutation

Once the new population has replaced the previous one, a mutation operator is applied to every individual as follows. We randomly select a position, say i, in the activity sequence (i.e., $1<i<|J|-2$) and a random number x_1 from a uniform distribution between 0 and 1. Then we permute the jobs in positions i and i+1 if the activity in position i is not an immediate predecessor of the activity in position i+1 and $x_1<\mu$, the mutation probability. In our implementation of the GA we used μ = 0.05. We may also mute one or both of the last two binaries in the code with a mutation

probability equals μ. Thus we draw a random number x_2 and flip the binary indicating whether we use the serial or the parallel scheme to construct the project schedule if $x_2<\mu$ and we draw another random variable x_3 and change the last binary of the code if $x_3<\mu$.

Initialize: $t = 1$
$C = \{1\}$ (set of activities completed by period t)
$FT_i = 0$; $\forall i \in J$
$y_{1k} = Q_k$; $\forall k \in K$
$y_{ik} = 0$; $\forall i \in J - \{1, F\}$, $\forall k \in K$
$S = \{\forall i \in J \mid P_i = C\}$ (set of candidate activities)
$A = \varnothing$ (set of activities in progress at period t)
$a_{kt} = Q_k$; $\forall k \in K$, t=1, .., T (units of k still available at t)
Arrange the list S according to selected priority rule (e.g. MINSLK, MINLFT, etc.)
For j =1 **to** $|S|$
  For k =1 **to** $|K|$
    $Z_{jk} = \varnothing$
    For l =1 **to** $|C|$
      Let i be the activity in position l in the set C
      IF $y_{ik} > 0$ **AND** $FT_i + \Delta_{ijk} < t$ **THEN** $Z_{jk} = Z_{jk} + \{i\}$
    NEXT l
  NEXT k
  For k =1 **to** $|K|$
    Arrange the activities $i \in Z_{jk}$ in the ascending order of their transfer time Δ_{ijk}
    $u_k = 0$
    WHILE $u_k < q_{jk}$
      Let i be the next activity in the ordered list Z_{jk}
      $x = min(y_{ik}, q_{jk} - u_k)$
      $y_{jk} = y_{jk} + x$
      $u_k = u_k + x$
    END WHILE
  NEXT k
  $A = A \cup \{j\}$
  $FT_j = t + d_j$,
  FOR $\tau = t$ to FT_j
    FOR k =1 to $|K|$
      $a_{kt} = a_{kt} - q_{jk}$
    NEXT k
  NEXT τ
  (Update the list of candidate activities S)
  $S = \{\forall j \in J - C - A \mid P_j \subseteq C; q_{jk} \le a_{k\tau}; \forall \tau = t,.., t+d_j, \forall k \in K\}$
  IF $S = \varnothing$ **AND** $(C \cup A) \neq J$ **THEN**
    t = the earliest period where there is at least one candidate activity
    Update C the list of activities completed before or at t
      $C = \{\forall j \in J \mid FT_j \le t\}$
    Update A the list of activities in progress at t
      $A = \{\forall j \in A \mid FT_j > t\}$
    Update S the list of candidate activities
      $S = \{\forall j \in J - C - A \mid P_j \subseteq C; q_{jk} \le a_{k\tau}; \forall \tau = t,.., t+d_j, \forall k \in K\}$
  END IF
  IF $S = \varnothing$ **AND** $(C \cup A) = J$ **THEN** Stop
  Arrange the list S according to selected priority rule (e.g. MINSLK, MINLFT, etc.)
NEXT j

Figure 7: Parallel Scheduling scheme for the RCPSPTT.

Notice that from every pair of parent solutions we generate two offspring solutions. To generate the first offspring, we use the first parent solution as donator and the second as receiver while to generate the second offspring we consider the second parent as donator and the first one as receiver.

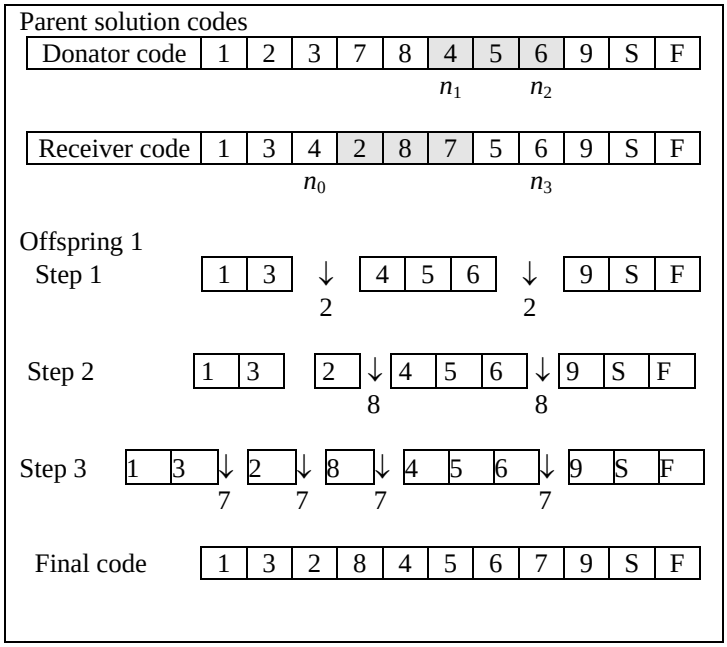

Figure 8: Application of the MMBX.

5 COMPUTATIONAL RESULTS

5.1 Test instances

To assess the performance of the proposed genetic algorithm we use two instance-sets composed of a total of 869 instances. These instances are the 480 instances of the set *J*30 and 389 instances among the 480 instances of the set *J*60 of the resource-constrained project scheduling instances generated by the instance generator PROGEN (Kolisch et al. 1995) and provided by the project scheduling problem library PSPLIB (http://www.om-db.wi.tum.de/psplib/main.html), to which we added transfer times.

These instances are those for which we were able to obtain their optimal solution. Table 1 gives the characteristics of these instances. For all these instances the network complexity (*NC*) is between 1.5 and 2.1 while the resource factor (*RF*) is between 0.25 and 1 and resource strength (*RS*) is between 0.25 and 1. The network complexity *NC* is measured by the average number of immediate predecessors in the network while the resource factor *RF* indicates the portion of resources consumed. The table also indicates the resources strength which is a scaling parameter expressing the resource availability as a convex combination of the sum of the minimum amount of the resources that allows to schedule the project and the amount required to schedule activities at their earliest start time. The added transfer times were determined in a way that makes optimal solutions of these instances without transfer times remain optimal for the corresponding instances with transfer times.

Set	*NC*	*RF*	*RS*	Number of instances
*J*30	[1.5,2.1]	[0.25,1.0]	0.25	120
			0.50	120
			0.75	120
			1.00	120
	All			480
*J*60	[1.5,2.1]	[0.25,1.0]	0.25	35
			0.50	114
			0.75	120
			1.00	120
	All			389

Table 1: Instances for which the optimal solution is found.

This way of generating transfer times is similar to what has been done by Krüger and Scholl (2009); Krüger (2009) and Poppenborg and Knust (2014) to generate their test-instance.

To generate test instance with transfer times that preserve the optimality of the solution for the same instance where there is no transfer times, we first solve to optimality the instance without transfer times, determine for each activity the set of activities from which we can transfer its required resources, and then add transfer times less than or equal to the difference between the finish time of the activities providing the required resources and the start time of the activity requiring these resources. We used the commercial MIP code GUROBI 5.0.1 to solve the test instances. This MIP code was able to solve all the *J*30 instances but not all the *J*60 instances. So we used only 389 instances for which the optimal solution was found. Krüger and Scholl (2009) indicated that they were able to get the optimal solution for 400 among the 480 instances of the *J*60 set. However, their solutions are not provided anywhere.

5.2 Results obtained by the proposed genetic algorithm

The proposed genetic algorithm is implemented in Matlab release 2014 and all calculations were run on a computer with i7 core processor, 2 GHz and 6 GB of internal memory. To solve each instance, the limit of 5000 generated schedules has been set for the procedure. Table 2 gives the percentage deviation from the optimal solution for the *J*30 and *J*60 instances. From this table we can see that the proposed GA obtained an average deviation from the optimum of 0.12% and 0.36% for the *J*30 and *J*60 instances respectively.

Set	*NC*	*RF*	*RS*	Average deviation from optimum
*J*30	[1.5,2.1]	[0.25,1.0]	0.25	0.39%
			0.50	0.11%
			0.75	0.00%
			1.00	0.00%
	All			0.12%
*J*60	[1.5,2.1]	[0.25,1.0]	0.25	1.13%
			0.50	0.88%
			0.75	0.00%
			1.00	0.00%
	All			0.36%

Table 2: Average percentage deviation from the optimum as obtained by the proposed genetic algorithm.

Table 3 compare the results of the proposed GA with those of the genetic algorithm of Krüger (2009) and of the flow based Tabu search algorithm of Poppenborg and Knust (2014). These results show that the proposed genetic algorithm developed here gives the best results outperforming the two other heuristics.

	*J*30	*J*60
GA Proposed in this paper	0.12%	0.36%
Krüger's GA (2009)	0.16%	0.38%
Popenborg and Knust's TS (2014)	1.27%	0.98%

Table 3: Average percentage deviation from the optimum: a comparison with the published solution methods.

6 CONCLUSION

In this paper, the Resource Constrained Project Scheduling Problem with sequence dependent Transfer Times (RCPSPTT) is considered, where sequence dependent transfer time is the time needed to transfer required resources between different activity execution sites. A genetic algorithm using a new crossover operator is proposed. Experimental results show that this GA is able to efficiently solve this problem. This GA has been compared to methods published in the literature using a set of instances constructed by the ProGen project generator to which we added transfer times. The obtained results show that the proposed genetic algorithm outperforms the genetic algorithm developed by Krüger (2009) as well as the Tabou search algorithm of Popenborg and Knust (2014).

Further research is obviously needed to develop more efficient optimal solution procedures for the RCPSPTT. The development of an optimal solution procedure capable of solving all the used test instances will allow us to provide a more complete assessment of the performance of the proposed GA algorithm.

Acknowledgement

This research work was partially supported by Grant OPG0036509 from the Canadian Natural Sciences and Engineering Research Council (NSERC). This support is gratefully acknowledged.

REFERENCE

Alcaraz, J., Maroto, C., Ruiz, R., 2003. Solving the multi-mode resource- constrained project scheduling problem with genetic algorithms. Journal of the Operational Research Society 54 (6), 614–626.

Baar, T., Brucker, P., Knust, S., 1999. Tabu search algorithms and lower bounds for the resource-constrained project scheduling problem. In: Vo, S., Martello, S., Osman, I., Roucairol, C. (Eds.), Meta-Heuristics. Springer US, pp. 1–18. Blazewicz, J., Lenstra, J. K., Kan, A., 1983. Scheduling subject to resource constraints: classification and complexity. Discrete Applied Mathematics 5 (1), 11–24.

Bierwirth, C., Mattfeld, D., Kopfer, H., 1996. On permutation representations for scheduling problems. Parallel Problem Solving from Nature IV, 310-318.

Boctor, F. F., 1990. Some efficient multi-heuristic procedures for resource- constrained project scheduling. European Journal of Operational Research 49 (1), 3–13.

Boctor, F. F., 1996. Resource-constrained project scheduling by simulated annealing. International Journal of Production Research 34 (8), 2335–2351.

Bouleimen, K., Lecocq, H., 2003. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. European Journal of Operational Research 149 (2), 268–281.

Brooks, G., White, C., 1965. An algorithm for finding optimal or near optimal solution to the production scheduling problem. Journal of Industrial Engineering, 34 – 40.

Brucker, P., Knust, S., Schoo, A., Thiele, O., 1998. A branch and bound algorithm for the resource-constrained project scheduling problem1. European Journal of Operational Research 107 (2), 272 – 288

Carlier, J., Latapie, B., 1991. Une méthode arborescente pour résoudre les problèmes cumulatifs. RAIRO. Recherche opérationnelle 25 (3), 311–340.

Davis, J., 1985. Job shop scheduling with genetic algorithms. Proceedings of the first international conference on genetic algorithms. 136-140.

Debels, D., De Reyck, B., Leus, R., Vanhoucke, M., 2006. A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. European Journal of Operational Research 169 (2), 638–653.

Demeulemeester, E. L., Herroelen, W. S., 1996. An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem. European Journal of Operational Research 90 (2), 334–348.

Djerid, L., Portman, M. C., Villon, P., 1996. Performance analysis of permutation cross-over genetic operators. Journal of Decision Systems 4, 131-140.

Hartmann, S., 1998. A competitive genetic algorithm for resource constrained project scheduling. Naval Research logistics 45, 733-750.

Hartmann, S., 2001. Project scheduling with multiple modes: a genetic algorithm. Annals of Operations Research 102 (1-4), 111–135.

Kelley, J., 1963. The critical-path method : Resources planning and scheduling. In : Muth, J. F., Thomson, G. L. (Eds.), Industrial Planning Scheduling. Prentice-Hall, 347 – 365.

Kolisch, R., 1996a. Efficient priority rules for the resource-constrained project scheduling problem. Journal of Operations Management 14 (3), 179 – 192.

Kolisch, R., 1996b. Serial and parallel resource-constrained project schedul- ing methods revisited: Theory and computation. European Journal of Operational Research 90 (2), 320 – 333.

Kolisch, R., Drexl, A., 1997. Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE transactions 29 (11), 987–999. Krüger, D., 2009. Multi-project scheduling with transfers. PhD dissertation, University of Jena.

Krüger, D., Scholl, A., 2009. A heuristic solution framework for the resource constrained (multi-) project scheduling problem with sequence-dependent transfer times. European Journal of Operational Research 197 (2), 492– 508.

Mendes, J. J. d. M., Goncalves, J. F., Resende, M. G., 2009. A random key based genetic algorithm for the resource constrained project scheduling problem. Computers & Operations Research 36 (1), 92–109.

Mobini, M. M., Rabbani, M., Amalnik, M., Razmi, J., Rahimi-Vahed, A., 2009. Using an enhanced scatter search algorithm for a resource- constrained project scheduling problem. Soft Computing 13 (6), 597–610.

Nonobe, K., Ibaraki, T., 2002. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In: Essays and surveys in metaheuristics. Springer, pp. 557–588.

Patterson, J. H., Brian Talbot, F., Slowinski, R., Wegelarz, J., 1990. Com- putational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. European Journal of Operational Research 49 (1), 68–79.

Poppenborg, J., Knust, S., 2014. A flow-based tabu search algorithm for the rcpsp with transfer times. At the 14th International Conference on Project Management and Scheduling, March 30th April 2nd 2014, 181 – 184.

Valls, V., Ballestın, F., Quintanilla, S., 2004. A population-based approach to the resource-constrained project scheduling problem. Annals of Operations Research 131 (1-4), 305–324.

Zamani, R., 2013. A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem. European Journal of Operational Research. 20