A Genetic Algorithm with Neighborhood Search for the Resource-Constrained Project Scheduling Problem

Sepehr Proon,[1] Mingzhou Jin[2]

[1] *Industrial Engineering, University of Pittsburgh, Pittsburgh*

[2] *Industrial Engineering, Mississippi State University, Mississippi State, Mississippi 39762*

Received 6 August 2008; revised 8 October 2010; accepted 15 October 2010
DOI 10.1002/nav.20439
Published online 3 February 2011 in Wiley Online Library (wileyonlinelibrary.com).

Abstract: The resource-constrained project scheduling problem (RCPSP) consists of a set of non-preemptive activities that follow precedence relationship and consume resources. Under the limited amount of the resources, the objective of RCPSP is to find a schedule of the activities to minimize the project makespan. This article presents a new genetic algorithm (GA) by incorporating a local search strategy in GA operators. The local search strategy improves the efficiency of searching the solution space while keeping the randomness of the GA approach. Extensive numerical experiments show that the proposed GA with neighborhood search works well regarding solution quality and computational time compared with existing algorithms in the RCPSP literature, especially for the instances with a large number of activities. © 2011 Wiley Periodicals, Inc. Naval Research Logistics 58: 73–82, 2011

Keywords: resource-constrained project scheduling problem; genetic algorithm; neighborhood search

1. INTRODUCTION

The resource-constrained project scheduling problem (RCPSP) is to schedule a set of non-preemptive activities $A = \{1, 2, \ldots, n\}$ in a project under resource constraints [1]. Two additional dummy activities, 0 as a source node and $n + 1$ as a sink node, are added when the project is represented as an activity-on-node network $G = \{V, E\}$, where $V = A \cup \{0, n + 1\}$ and $E =$ the set of directed edges representing precedence relationship between activities. A deterministic duration d_j is associated with each activity $j \in A$ and $d_0 = d_{n+1} = 0$. Assume there are K types of resources (k is its index) and each resource type has R_k units in each time period t. Each activity j requires r_{jk} units of resource k during each period t in which the activity is performed. Also, $r_{0k} = r_{n+1,k} = 0$ for all $k \in \{1, 2, \ldots, K\}$. All parameters are assumed to be non-negative integers.

The objective of the RCPSP is to find a precedence- and resource-feasible schedule $S = (s_1, s_2, \ldots, s_n, s_{n+1})$ to minimize the makespan (i.e., s_{n+1}) of the project. Here, s_j is the start time of activity j and $s_0 = 0$. When scheduling the activities in the project, we need to consider two types of constraints:

- The precedence constraints: No activity can start before all its immediate predecessors have finished. In other words, $s_j \geq s_i + d_i, \forall (i, j) \in E$.
- The resource availability constraints: The total units of each resource type k required by all activities that are scheduled in a time period should not exceed the available amount of that resource type in that time period. In other words, $\sum_{i \in \{i | s_i \leq t \leq s_i + d_i - 1\}} r_{ik} \leq R_k, \forall t \in \{0, 1, \ldots, \text{UB}\}, \forall k \in \{i, \ldots, K\}$, where UB is an upper bound makespan for the project (e.g., $\text{UB} = \sum_{i \in A} d_i$).

Figure 1 illustrates an RCPSP example provided by Demeulemeester and Herroelen [2]. The example comprises $n = 6$ activities with two resource types, which have four and two units in each time period t, respectively. The arcs show the precedence relationship between activities. Three integer numbers $(d_j, r_{j,1}, r_{j,2})$, the duration and required Resource Type 1 and Resource Type 2 per time unit, are associated to each activity j. A feasible schedule with an optimal makespan of 15 periods is provided in Fig. 2 [2].

Being a generalized form of the Job Shop Scheduling problem, the RCPSP has been proven an NP-hard problem [2]. How to solve the large-scaled RCPSP has been a subject of interest for practitioners and researches over the past few decades. A huge number of heuristic and exact-solution

Correspondence to: M. Jin (mjin@ise.msstate.edu)

approaches have been developed in the RCPSP literature. We refer to the surveys provided by Herroelen et al. [4], Brucker [5], Kolisch and Padman [6], and Demeulemeester and Herroelen [2].

The exact-solution procedures developed for the RCPSP so far can only handle small-sized problems with fewer than 60 activities. To solve a large-sized RCPSP, which typically has hundreds or thousands of activities in practice, heuristic procedures are the main interest of research. Many heuristic procedures are based on priority rules and forward–backward methods [1]. The meta-heuristic algorithms proposed for the RCPSP include the genetic algorithm (GA), the Tabu search, the Simulated Annealing, the Ant Colony optimization, the path re-linking, and hybrid algorithms [1, 7, 8]. The numerical experiments [8] conducted on standard instance sets, namely J30, J60, and J120 generated by ProGen in the PSPLIB [9], showed that the meta-heuristic methods outperformed the heuristic methods based on priority rules and forward–backward methods.

This paper proposes a GA approach to solve the RCPSP. GA is a search heuristic that mimics biological evolution. For a general introduction to the GA approach, we refer to Goldberg [10]. The GA implementation does not necessarily need any specific mathematical requirements and is capable of optimizing the problem regardless of the functionality of the problem under study. Therefore, the GA approach is suitable for solving any problems that have any kind of discrete or continuous constraints over the solution space [10]. The flexibility of the GA approach makes it a good option to have other heuristics embedded and therefore lead to new and efficient approaches. There are quite a few implementations of pure and modified GA in the RCPSP literature. The modifications include adding new features such as path re-linking, forward–backward improvement, self-adapting mechanisms, non-standard crossover techniques, or even other meta-heuristics [1]. These modifications have resulted in better quality solutions. Kolisch and Hartmann [8] provided a classification and comparison of different heuristics

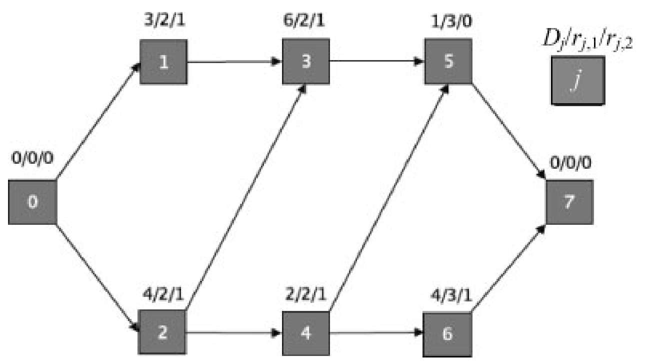


Figure 1. An RCPSP example with six activities and two resource types.

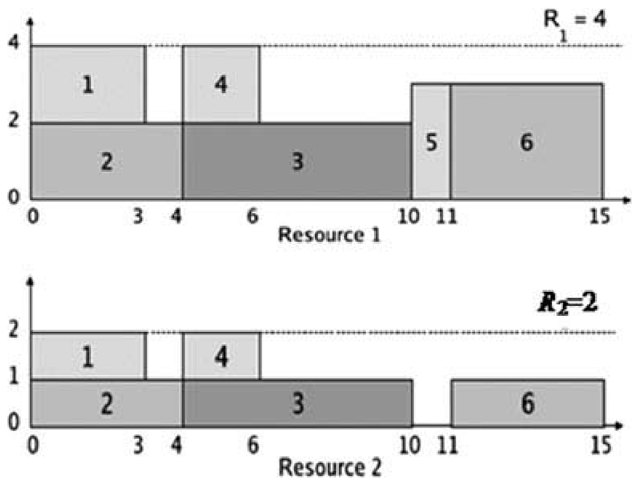


Figure 2. The optimal with a makespan of 15 periods for the RCPSP in Fig. 1.

for the RCPSP. Based on numerical experiment results [8], the GA approach in general outperforms other meta-heuristic approaches. Out of the four best heuristic procedures for the RCPSP in their comparison, three procedures are GA-based. However, all existing GA-based algorithms for the RCPSP do not utilize any directed search strategies. Although the main idea of the GA approach is to perform a random search based on the fitness in the feasible solution space, we believe it is desirable to conduct a directed (neighborhood) search to improve search efficiency for large-sized real-world project scheduling problems involving a huge number of activities. This article integrates the new directed search feature into the basic GA scheme by modifying crossover and mutation operators. This integration provides a better search direction while keeping the randomness of the GA approach. The numerical experiments demonstrate that the integrated strategy leads to a better GA to solve the RCPSP.

The remainder of the article is organized as follows. Section 2 describes the proposed genetic algorithm with neighborhood search (GANS). Section 3 presents numerical experiments to compare the GANS with other algorithms in the literature on standard instance sets in the PSPLIB (Kolisch and Sprecher, 1996). Section 4 concludes the article and provides future research directions.

2. GENETIC ALGORITHM WITH NEIGHBORHOOD SEARCH FOR THE RCPSP

In this section, we propose a new GA approach for solving the RCPSP by incorporating the neighborhood search method introduced by Palpant et al. [11]. The proposed GANS aims to (1) keep the randomness of the GA search and (2) improve solution quality by conducting a neighborhood search at each iteration of the GA. In the heuristics literature, the neighborhood search is widely used to solve large-scaled combinatorial optimization problems. Exact-solution approaches, at each iteration, such as the branch-and-bound method, can be used to solve a reduced problem of the original one obtained by some procedures. Palpant et al. [11] introduced the Local Search with Sub-problem Exact Resolution (LSSPER) method for the RCPSP. The authors investigated various strategies for generating sub-problems to be optimized at each iteration. The hybrid of local search and exact-solution method seems to be very promising [8]. However, the overall solution search across iterations is conducted in a random fashion in the LSSPER method. We conjecture that incorporating the neighborhood (local) search into a GA framework will result in a good numerical performance.

2.1. Neighborhood Search Operator

The neighborhood search (NS) operator is used to improve one feasible solution by fixing the start times of some activities and rescheduling other activities. Palpant et al. [11] proposed five selection methods to form the sub-problem, which is defined by the set of fixed activities. Their numerical experiments showed that the "Block" selection method clearly outperformed other methods. Therefore, this paper uses the Block selection method to form the sub-problem in the NS operator. To facilitate the overall GA approach, the NS operator in this article includes a given core activity j rather than creates a random core activity as Palpant et al. [11] did.

For a given feasible schedule $S = \{s_1, \dots, s_n, s_{n+1}\}$ and a core activity $j \in A$, the NS operator reschedules a set of activities, A_j^s, while keeping the start times of other activities. Let P a predetermined number of activities that will be rescheduled. The value of P influences the computational time to obtain a neighborhood solution by rescheduling. Smaller P usually means fewer activities to be rescheduled and less time to obtain a new schedule. The following Block selection method is used to create A_j^s [11].

```
Step 0: $A_j^s = \{j\}$; $b = 0$; create a random order for all
        activities $\in A/\{j\}$. Let $i$ = the first activity in the
        order.
Step 1: If $s_j - d_i - b \le s_i \le s_j + d_j + b, A_j^s = A_j^s \cup \{i\}$.
Step 2: If $|A_j^s| = P$, go to Step 5.
Step 3: If $i$ is the last activity among the ones not belonging
        to $A_j^s$ based on the order defined in Step 0, $b = b+1$.
Step 4: Let $i$ be the next activity among the ones not belong-
        ing to $A_j^s$ based on the order defined in Step 0. Go to
        Step 1.
Step 5: END
```

The Block selection method basically selects a set of P activities that are overlapped or close to activity j in a given feasible schedule. The rescheduling effort defines a sub-problem, which is formed by the following two steps.

I. Fix the start times of all the activities $\notin A_j^s$ and release resources used by all the activities $\in A_j^s$ in each time period t. The available amount of resource k for activities $\in A_j^s$ in period t is R_k minus the resource used by all the activities $\notin A_j^s$ in period t.
II. Derive an earliest start time (EST) and a latest finish time (LFT) for each activity $i \in A_j^s$ as

$$\mathrm{EST}_i = \max\{s_l + d_l, \forall l \notin A_j^s \text{ and } (l,i) \in E\} \text{ and}$$
$$\mathrm{LFT}_i = \max\{s_l, \forall l \notin A_j^s \text{ and } (l,i) \in E\}.$$

(EST_i, LFT_i) defines a time window for activity i that could be rescheduled in order to guarantee that the new schedule is still feasible for all activities that will not be rescheduled. The rescheduling problem is to reschedule all the activities $\in A_j^s$ to minimize their makespan while meeting the resource restriction of each period and time window constraints defined by $(\mathrm{EST}_i, \mathrm{LFT}_i)$. Palpant et al. [1] used a commercial integer linear programming solver to obtain the optimal solution for the rescheduling problem. Because of using the GA approach rather than a simple iterative local search method, this paper adopts the simple forward or backward serial scheduling generation schemes (SGS) to solve the sub-problem [1, 12]. In each iteration, a new random vector is produced for the activities $i \in A_j^s$ as a priority list. The vector is created iteratively by randomly picking the next activity among all unselected activities whose precedent activities in A_j^s have been selected. Following the priority list, one activity by one is moved to the earliest (latest) start time that is precedence- and resource-feasible and satisfies the time window $(\mathrm{EST}_i, \mathrm{LFT}_i)$. Once all the activities $i \in A_j^s$ are rescheduled, the activities that do not belong to A_j^s are added to form a complete feasible solution for the RCPSP. A global left shift is then performed on all the activities $\in A$ to possibly reduce the makespan [11]. The resulting new schedule is compared with the previous solution before applying the NS operator. If the makespan is improved, the resulting schedule replaces the previous schedule and the NS operator stops. If there is no improvement, as long as the number of iterations has not reached a predefined limit, I, the forward (or backward) serial SGS is applied on the schedule with a new random priority list as the next iteration. Each of the I iterations counts for 2 generated schedules, one for rescheduling activities in A_j^s and adding the activities that do not belong to

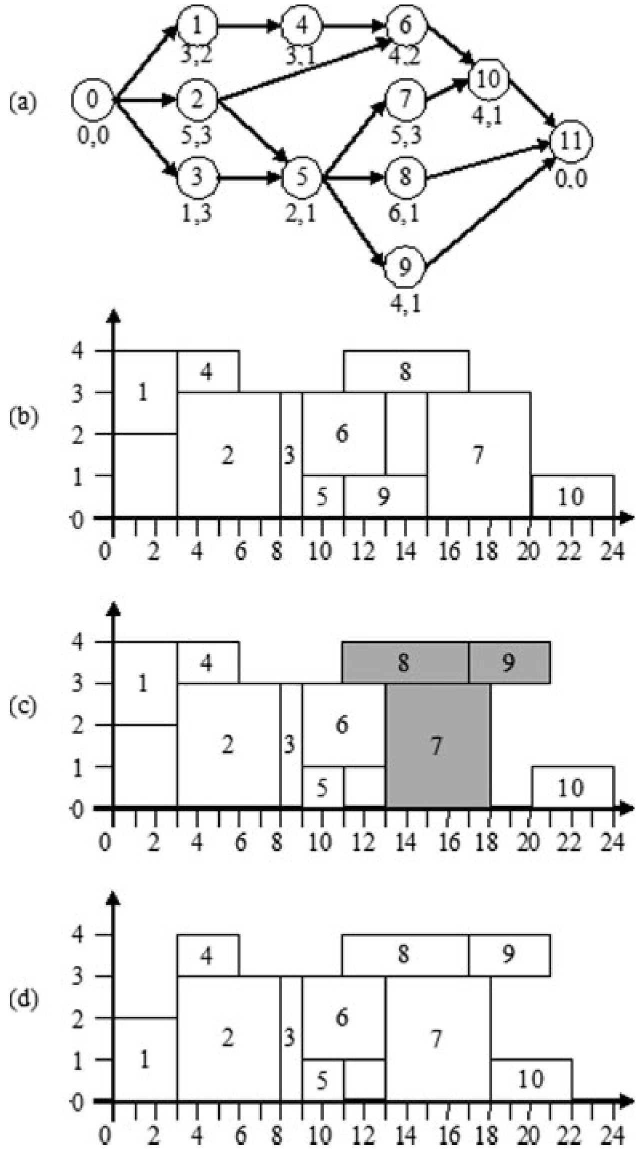


Figure 3. An NS operator example.

A_j^s to generate a new schedule, one for the global left shift.

Figure 3 shows one example of the application of the NS operator. The example RCPSP with 10 nondummy activities and a single resource of four units was provided by Klein and Scholl [13]. In part (a) of the figure, the precedence relationship is displayed. Under each node, the duration and the required resource amount are given, respectively. A feasible schedule of makespan 24 provided by Palpant et al. [11] is displayed in part (b) of the figure. An NS operator with activity 7 as the core activity and $P = 3$ is applied to the feasible schedule. Here, we assume $A_j^s = \{7, 8, 9\}$ and the priority list is also [7,8,9] where activity 7 has the highest priority and activity 9 has the lowest priority. Part (c) of the figure is the schedule obtained after a simple serial SGS is applied to A_j^s (gray boxes). A global left shift results in a new schedule of makespan 22 in part (d) of the figure.

In summary, the NS operator defined by a core activity j with a predefined P conducts up to I iterations of forward (or backward) serial SGS on a feasible schedule trying to find a better feasible schedule with a shorter makespan. In the following numerical experiments, we count all schedules created when performing an NS operator. The computational effort of an NS operator could be between 2 and 2I, measured by the number of created schedules, including the global left shift effort.

2.2. Genetic Algorithm with Neighborhood Search

In this subsection, we define the basic elements of the proposed GANS, including the chromosome representation, fitness functions, parent selection, crossover operators, mutation operators, and stop criteria.

2.2.1. *Chromosome Representation*

A feasible solution to the RCPSP is a vector of the start times of activities $S = (s_1, s_2, \ldots, s_{n+1})$, where s_i is the start time of activity i. Figure 4 shows the chromosome structure in the GANS.

The first gene X is the index of the core activity used in the NS operator that has been applied to obtain the chromosome (the feasible solution). The second gene Y is the contribution of this NS operator on the current chromosome. This contribution is the improvement (reduced makespan) achieved by applying the NS operator with a core activity X. Since any NS operator either has an improvement in the objective function of the main problem or keeps the original schedule unchanged in the case of not finding any better solution, the value of Y in any chromosome is always non-negative. s_1 through s_{n+1} are the start times of activities in the project in the current solution. The last gene, Z, is a binary variable indicating whether the forward or backward serial SGS has been used to solve the rescheduling sub-problem in the NS operator. $Z = 1$ means the forward serial SGS has been used.

2.2.2. *Fitness Functions*

The GANS uses two fitness functions. The first fitness function is the project makespan, which is the most common

X	Y	s_1	s_2	...	s_n	s_{n+1}	Z

Figure 4. Chromosome structure.

objective function of the RCPSP. The second fitness function is the improvement achieved in the objective function by applying a specific NS operator, which is Y and the second gene in the chromosome structure.

2.2.3. *Parent Selection*

Each generation of the GANS has M chromosomes. For creating the next generation, N_p pairs of father and mother chromosomes are selected in each generation. The parent selection uses both fitness functions. A simple roulette wheel selection method [10] is used to select the father and mother chromosomes. The father chromosome is first selected based on the makespan of each chromosome, such as $s^m_{n+1}, m = 1, 2, \dots, M$. The selection probability of each chromosome $m, m = 1, \dots, M$, is calculated based on its makespan s^m_{n+1} as $(s^m_{n+1})^{-1} / \sum_{i=1}^{M}[(S^l_{n+1})^{-1}]$. After selecting the father chromosome, the algorithm selects the mother chromosome by the same procedure but using $Y^m, m = 1, \dots, M$, as the fitness function to calculate the selection probability.

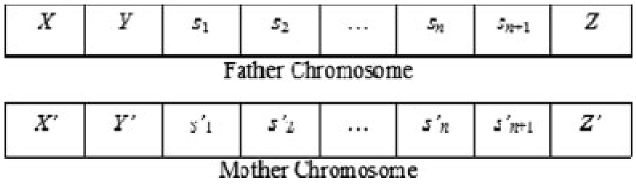
2.2.4. *Crossover Operator*

Once the father and mother chromosomes have been selected, the crossover operator will be applied on the chromosomes to create children. Figure 5 shows the father and mother chromosomes and Fig. 6 illustrates the crossover operator.

As illustrated by Figs. 5 and 6, the crossover operator applies the NS operator of the father chromosome, defined by X and Z, on the schedule of the mother chromosome $(s'_1, s'_2, \dots, s'_{n+1})$ and applies the NS operator of the mother chromosome, defined by X' and Z', on the schedule of the father chromosome $(s_1, s_2, \dots, s_{n+1})$ to produce two child chromosomes. Y and Y' are re-calculated based on the difference of the makespan before and after applying the NS operator. If there is no improvement by applying the NS operator, the existing solution remains and Y becomes 0.

2.2.5. *Mutation Operator*

After applying the crossover operators on the parents to obtain the child chromosomes, the GANS applies a modified



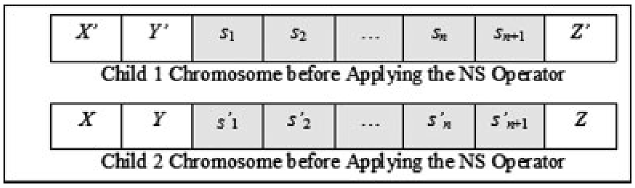
X	Y	s_1	s_2	...	s_n	s_{n+1}	Z

Father Chromosome

X'	Y'	s'_1	s'_2	...	s'_n	s'_{n+1}	Z'

Mother Chromosome

Figure 5. The selected parents.



X'	Y'	s_1	s_2	...	s_n	s_{n+1}	Z'

Child 1 Chromosome before Applying the NS Operator

X	Y	s'_1	s'_2	...	s'_n	s'_{n+1}	Z

Child 2 Chromosome before Applying the NS Operator

Figure 6. Children produced after the crossover operator.

mutation operator on chromosomes to avoid being trapped in local optimum solutions. The psudo-code for the mutation operator is as follows.

```
Step 1. For each child chromosome produced by applying the crossover operator, generate a random number $u$ between 0 and 1.
Step 2. If the produced random number $u$ is less than a predefined number $p_{\text{mut}}$, produce a random integer number $j$ between 1 and $n$ and produce a random number $w$ between 0 and 1. Otherwise, go to Step 5.
Step 3. If $w$ is less than another predefined number $p'_{\text{mut}}$, reverse the direction of the serial SGS in the NS operator. In other words, use the forward direction if $Z = 1$ or the backward direction if $Z = 0$.
Step 4. Apply the new NS operator defined by the core activity $j$ and the direction of the serial SGS on the schedule $(s_1, s_2, \dots, s_{n+1})$ of the chromosome. Update the value of $X$ with $j$, the schedule, and the value of $Z$ with the new direction. If there is an improvement, update the value of $Y$ with the improvement. Otherwise, update the value of $Y$ with 0.
Step 5. End.
```

The mutation operator basically adds more randomness into the GANS by applying a randomly created NS operator with some probability.

2.2.6. *Selection of Chromosomes for the Next Generation*

The GANS keeps M chromosomes at each generation. From each generation, N_p pairs of father and mother chromosomes are selected to create $2N_p$ child chromosomes by applying the crossover and mutation operators. M chromosomes are selected out of the $2N_p$ child chromosomes. Father and mother chromosomes are not direct candidates in this selection. Please note that some child chromosomes obtained after the crossover and mutation operators may be the same as some parent chromosome and could be candidates for the next generations. At first, the GANS chooses the top N_t chromosomes out of the $2N_p$ child chromosomes based on their makespan. For the rest of the chromosomes, a simple roulette wheel tournament considering the makespan for each chromosome as its fitness function is used to choose the remaining

Table 1. Parameter Settings of the GANS in Numerical experiments.

Parameter notation	Parameter meaning	Definition place	Value
n	The number of activities in the RCPSP	1	30/60/120
P	The number of activities to be rescheduled in each sub-problem	2.1	$0.15n$, $0.2n$ or $0.25n$
I	The maximum number of serial SGS iterations in the NS operator	2.1	5, 8, 10, 12 or 15
M	The number of chromosomes in each generation	2.2.3	40
N_p	The number of father and mother chromosomes pairs used in each generation	2.2.3	20
p_{mut}	The predefined number to decide whether to apply the mutation operator	2.2.5	0.02
p'_{mut}	The predefined number to decide whether to change the direction of the serial SGS in the mutation operator	2.2.5	0.05
N_t	The number of child chromosomes that are directly selected into the next generation	2.2.6	8 (20% of M)
λ	Maximum number of schedules to be produced	2.2.7	1000, 5000, 50,000 or ∞
γ	Maximum number of generations without an improvement	2.2.7	300

$M - N_t$ chromosomes for the next generation. The roulette wheel tournament is the same as that used in choosing the father chromosome in 2.2.3.

2.2.7. Stop Conditions for the GANS

There are two stop conditions for the proposed algorithm. The first stop condition stops the GANS once a predefined number of schedules (solutions), denoted by λ, have been produced. The GANS counts any schedule that is created during the outer GA loop and the inner NS loop. The number of created schedules created by one NS operator could be between 2 and $2I$. The second stop condition stops the GANS when there is no improvement in the solution quality for γ successive generations. The first stopping condition enables us to compare the GANS with other start-of-the-art algorithms no matter which computers are used to implement the algorithms.

3. NUMERICAL EXPERIMENTS

The GANS was coded in C and run on a 400 MHz computer. Three standard RCPSP test sets from the Project Scheduling Problem Library (PSPLIB) were used, including J30 (containing 30 activities), J60 (containing 60 activities), and J120 (containing 120 activities). Each test set includes 480 instances except set J120 which contains 600 instances. In other words, totally 1560 instances were used. Each instance considers four types of resources. Please see Kolisch et al. [14] for how the instances were created. The instances can be found in Kolisch and Sprecher [9] and are downloadable at http://129.187.106.231/psplib/data.html. The parameters of the GANS used in numerical experiments are summarized in Table 1.

To make a comparison between the GANS and other algorithms in the literature, the numerical experiments set λ, the limit of schedules, at 1000, 5000, 50000, or infinity. When finite λ is used, only the first stop condition is used. $\lambda = \infty$ means that only the second stop condition is applied. In the numerical experiments, all created schedules are counted. An NS operator may result in at least one and up to I schedules. The measure of solution quality is the average percent deviation (APD) from the optimal solutions (if known) or from the lower bounds obtained by the critical path algorithm [14] for instances in datasets J60 and J120 whose optimal solutions are unknown in the literature.

Before comparing the performance of the GANS with other algorithms, different values of I and P are tested in Table 2 to tune the parameters of the GANS. The value of I determines the allocation of computational effort between the inner NS operator and the outer GA loop. When there is a limit of total created schedules, a larger I in general means more schedules created by an NS operator but fewer generations for the GA loop. Table 2 shows that a balanced way of $I = 10$ could lead to the best performance no matter how many activities are involved and what the limit of schedule is. It is also interesting to see a larger value of P does not necessarily lead to a better result even though larger P usually means more computational time for the NS. A moderate value of $P = 0.2n$ could result in the lowest APD for any I and datasets. This phenomenon could be explained by the fact that a simple random sampling technique where the activities in A_j^s are randomly generated and scheduled is not a good strategy when there are many activities (i.e., $P = |A_j^s|$ is large). This fact has been shown in the studies of Hartmann and Kolisch [1, 7, 8]. The table also shows that the choices of I and P are relatively independent. No matter which dataset is used, $I = 10$ and $P = 0.2n$ always result in the lowest APD. In the later comparisons to other

Table 2. Average percent deviations (APD) with different values of I and P.

Parameters		Dataset J30			Dataset J60			Dataset J120		
		$\lambda = 1000$	$\lambda = 5000$	$\lambda = 50{,}000$	$\lambda = 1000$	$\lambda = 5000$	$\lambda = 50{,}000$	$\lambda = 1000$	$\lambda = 5000$	$\lambda = 50{,}000$
$P = 0.25n$	$I = 5$	2.32	1.80	1.08	11.56	10.89	10.70	33.94	31.87	30.54
	$I = 8$	2.11	1.65	0.92	11.47	10.64	10.63	33.93	31.85	30.51
	$I = 10$	1.85	1.31	0.78	11.37	10.56	10.54	33.47	31.79	30.48
	$I = 12$	1.85	1.48	0.81	11.61	10.78	10.69	33.67	31.82	30.52
	$I = 15$	1.93	1.77	1.11	11.71	10.94	10.72	33.69	31.85	30.52
$P = 0.20n$	$I = 5$	2.31	1.78	1.06	11.55	10.87	10.68	33.68	31.59	30.48
	$I = 8$	2.09	1.62	0.87	11.46	10.62	10.61	33.68	31.58	30.47
	$I = 10$	1.83	1.27	0.71	11.35	10.53	10.52	33.45	31.51	30.45
	$I = 12$	1.83	1.43	0.75	11.58	10.74	10.66	33.65	31.54	30.46
	$I = 15$	1.92	1.75	1.07	11.67	10.93	10.70	33.66	31.56	30.46
$P = 0.15n$	$I = 5$	2.31	1.79	1.07	11.56	10.87	10.69	33.93	31.86	30.53
	$I = 8$	2.10	1.64	0.90	11.46	10.63	10.62	33.93	31.83	30.50
	$I = 10$	1.85	1.31	0.77	11.36	10.56	10.55	33.46	31.78	30.50
	$I = 12$	1.85	1.47	0.82	11.60	10.77	10.68	33.66	31.81	30.51
	$I = 15$	1.92	1.76	1.10	11.71	10.93	10.71	33.67	31.84	30.52

Table 3. Average percent deviations (APD) for dataset J30 with finite λ.

Algorithm	SGS type	Reference	APD		
			$\lambda = 1000$	$\lambda = 5000$	$\lambda = 50,000$
GA, TS - Path re-linking	Both	Kochetov and Stoylar [15]	0.1	0.04	0.00
Scatter search - FBI	Serial	Debels et al. [16]	0.27	0.11	0.01
DBGA	Serial	Debels and Vanhouchke [17]	0.12	0.04	0.02
GA	Serial	Debels and Vanhouchke [17]	0.15	0.04	0.02
GA - Hybrid, FBI	Serial	Valls et al. [18]	0.27	0.06	0.02
GA – FBI	Serial	Valls et al. [19]	0.34	0.2	0.02
TS - Activity list	Serial	Nonobe and Ibaraki	0.46	0.16	0.05
Sampling - LFT, FBI	Both	Tormos and Lova [20]	0.3	0.16	0.07
GA - Self-adapting	Both	Hartmann [21]	0.38	0.22	0.08
GA - Activity list	Serial	Hartmann [22]	0.54	0.25	0.08
SA - Activity list	Serial	Bouleimen and Lecocq [23]	0.38	0.23	—
GA - Late join	Serial	Ceolho and Tavares [24]	0.74	0.33	0.16
Sampling - Adaptive	Both	Kolisch and Drexl [25]	0.65	0.44	—
GA - Priority rule	Serial	Hartmann [22]	1.38	1.12	0.23
GANS	Serial	This article	1.83	1.27	0.71
Sampling -WCS	Parallel	Kolisch [26]	1.4	1.28	—
Sampling - LFT	Parallel	Kolisch [12]	1.4	1.29	1.13
Sampling - Random	Parallel	Kolisch [27]	1.77	1.48	1.22
GA - Problem space	Parallel	Leon and Ramamoorthy [28]	2.08	1.59	—

Table 4. Average percent deviations (APD) for dataset J30 with infinite λ.

Algorithm	SGS type	Reference	APD	CPU time (s)		APD
				Average	Max	
Decomposition & local optimization	Serial	Palpant et al. [11]	0	10.26	123	2.3 GHz
Population-based	Serial	Valls et al. [29]	0.1	1.16	5.5	400 MHz
Network decomposition	—	Sprecher [30]	0.12	2.75	39.7	166 MHz
GANS	Serial	This article	1.12	7.39	37	400 MHz

Table 5. Average percent deviations (APD) for dataset J60 with finite λ.

Algorithm	SGS type	Reference	APD		
			$\lambda = 1000$	$\lambda = 5000$	$\lambda = 50,000$
GANS	Serial	This article	11.35	10.53	10.52
DBGA	Serial	Debels and Vanhouchke [17]	11.31	10.59	10.68
GA	Serial	Debels et and Vanhouchke [17]	11.45	10.59	10.68
Scatter search - FBI	Serial	Debels et al. [16]	11.73	11.1	10.71
GA - Hybrid, FBI	Serial	Valls et al. [18]	11.56	11.1	10.73
GA, TS - Path re-linking	Both	Kochetov and Stoylar [15]	11.71	11.17	10.74
GA - FBI	Serial	Valls et al. [19]	12.21	11.27	10.74
GA - Self-adapting	Both	Hartmann [21]	12.21	11.7	11.21
GA - Activity list	Serial	Hartmann [22]	12.68	11.89	11.23
Sampling - LFT, FBI	Both	Tormos and Lova [20]	12.81	11.87	11.54
SA - Activity list	Serial	Bouleimen and Lecocq [23]	12.75	11.9	—
TS - Activity list	Serial	Nonobe and Ibaraki	12.97	12.18	11.58
GA - Late join	Serial	Ceolho and Tavares [24]	13.28	12.68	11.94
GA - Priority rule	Serial	Hartmann [22]	13.3	12.74	12.26
Sampling - Adaptive	Both	Kolisch and Drexl [25]	13.51	13.06	—
Sampling -WCS	Parallel	Kolisch [12, 26]	13.66	13.21	—
Sampling - LFT	Parallel	Kolisch [12]	13.59	13.23	12.83
GA - Problem space	Parallel	Leon and Ramamoorthy [28]	14.33	13.49	—
Sampling - Random	Parallel	Kolisch [27]	14.89	14.3	13.66

algorithms in the literature, $I = 10$ and $P = 0.2n$ will be used in Tables 3–8.

For data set J30, Table 3 compares the GANS with other algorithms in the literature using a finite λ of 1000 or 5000 or 50,000. The second column shows whether the algorithm uses serial, parallel, or both SGS. The algorithms are ranked based on APD under $\lambda = 50,000$ from the lowest to the highest. In the literature, some algorithms do not limit the number of produced schedules. The GANS is compared with them for dataset J30 in Table 4, which includes average and maximum computational times of all instances, APD, and computer type. In Table 4, only the second stop condition is used in the GANS and the first stop condition regarding finite λ is ignored. For data set J30, Tables 3 and 4 show that the GANS is not competitive compared to other algorithms in the literature. When $\lambda = \infty$ and only the second stop condition is used, the GANS creates totally 40,800 schedules on average.

Tables 5–8 show the numerical experiment results for dataset J60 and dataset J120 with finite or infinite λ. The GANS is highlighted in each table. For the dataset J60, Table 5 shows that the GANS works well compared with almost all other algorithms when the first stop condition is used with $\lambda = 1000$ or 5000 or 50,000 schedules. Table 6 shows that the GANS yields better APD than all the other algorithms when $\lambda = \infty$ regarding solution quality. On average, the GANS creates totally 45,300 schedules when $\lambda = \infty$. For the dataset J120, Tables 7 and 8 indicate that the GANS yields better APD than all other algorithms no matter whether the first stop condition is applied or not. We note that the GANS yields better results for the dataset J120 with 50,000 schedule limit than with $\lambda = \infty$, because the GANS creates totally 47,800 schedules on average with $\lambda = \infty$ before the second stop condition takes effect. In summary, the numerical experiments demonstrate that the GANS yields better APD than existing algorithms for the RCPSP in the literature for large-sized instances. The GANS is not competitive on the small instances perhaps because the search space is in general small for those instances. Incorporating the neighborhood search in GA may waste time around local solutions and is not as efficient as exploring the relatively small solution space in a more random way.

Table 6. Average percent deviations (APD) for dataset J60 with Infinite λ.

Algorithm	SGS type	Reference	APD	CPU time (s)		Computer speed
				Average	Max	
GANS	Serial	This article	10.52	22	71	400 MHz
Decomposition & local optimization	Serial	Palpant et al. [11]	10.81	38.8	223	2.3 GHz
Population-based	Serial	Valls et al. [29]	10.89	3.7	22.6	400 MHz
Network decomposition	—	Sprecher [30]	11.61	460.2	4311.5	166 MHz

Table 7. Average percent deviations (APD) for dataset J120 with finite λ.

			APD		
Algorithm	SGS type	Reference	$\lambda = 1000$	$\lambda = 5000$	$\lambda = 50,000$
GANS	Serial	This article	33.45	31.51	30.45
DBGA	Serial	Debels and Vanhouchke [17]	33.55	32.18	30.69
GA	Serial	Debels and Vanhouchke [17]	34.19	32.34	30.82
GA - Hybrid, FBI	Serial	Valls et al. [18]	34.07	32.54	31.24
Scatter search - FBI	Serial	Debels et al. [16]	35.22	33.1	31.57
GA - FBI	Serial	Valls et al. [19]	35.39	33.24	31.58
GA, TS - Path re-linking	Both	Kochetov and Stoylar [15]	34.74	33.36	32.06
GA - Self-adapting	Both	Hartmann [21]	37.19	35.39	33.21
GA - Activity list	Serial	Hartmann [22]	39.37	36.74	34.04
Sampling - LFT, FBI	Both	Tormos and Lova [20]	36.49	35.81	35.01
TS - Activity list	Serial	Nonobe and Ibaraki	40.86	37.88	35.85
GA - Late join	Serial	Ceolho and Tavares [24]	39.97	38.41	36.44
SA - Activity list	Serial	Bouleimen and Lecocq [23]	42.81	37.68	—
GA - Priority rule	Serial	Hartmann [22]	39.93	38.49	36.51
Sampling - LFT	Parallel	Kolisch [12]	39.6	38.75	37.74
Sampling -WCS	Parallel	Kolisch [12, 26]	39.65	38.77	—
Sampling - Adaptive	Both	Kolisch and Drexl [25]	41.37	40.45	—
GA - Problem space	Parallel	Leon and Ramamoorthy [28]	42.91	40.69	—
Sampling - Random	Parallel	Kolisch [27]	44.46	43.05	41.44

Table 8. Average percent deviations (APD) for dataset J120 with infinite λ.

				CPU time (s)		
Algorithm	SGS type	Reference	APD	Average	Max	Computer speed
GANS	Serial	This article	30.78	37	175	400 MHz
GA - Hybrid, FBI	Serial	Valls et al. [18]	30.95[a]	39.51	69.26	400 MHz
Decomposition & local optimization	Serial	Palpant et al. [11]	31.58	59.4	264	2.3 GHz
Population-based	Serial	Valls et al. [29]	32.41	207.9	501	400 MHz
Network decomposition	—	Sprecher [30]	39.29	458.5	1511.3	166 MHz

[a]This result was obtained with $\lambda = 100,000$ rather than based on infinite λ.

4. CONCLUSION

This article proposes a GANS for the RCPSP. The GANS is developed based on the conjecture that incorporating the neighborhood search in a meta-heuristic framework (i.e., GA) may improve the algorithm performance. The NS operator, introduced by Palpant et al. [11], is defined by a core activity and the direction of the serial SGS. The NS operator is part of a chromosome and participates in the crossover and mutation operators in the genetic algorithm implementation. The NS operator can provide better search direction of the general genetic algorithm without hurting the randomness. Unlike Palpant et al. [11], the GANS only uses the simple serial SGS rather than any optimization or constrained programming to solve the rescheduling problem in the NS operator. The genetic algorithm framework rather than the intensive local search guarantees a good convergence of the overall algorithm. Numerical experiments based on three standard RCPSP test sets of J30, J60, and J120 demonstrate the GANS produces better APD than existing algorithms in the literature for large-sized instances. Since the large computational challenge lies on the large-sized problems, the GANS seems promising for solving the RCPSP in the practice. The numerical experiments also show that the GANS performance well when the computational effort assigned to the neighborhood search is not too much or too little and the number of activities in the rescheduling sub-problem is not too big or too small. In the future study, different heuristics will be tested to solve the rescheduling sub-problems in the NS operators.

REFERENCES

[1] R. Kolisch and S. Hartmann, "Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis," in Project scheduling: Recent models, algorithms and applications, J. Weglarz (Editor), Kluwer Academic Publishers, Berlin, 1999, pp. 147–178.

[2] E.L. Demeulemeester and W.S. Herroelen, Project scheduling: A research handbook, Kluwer Academic Publisher, Norwell, Massachusetts, 2002.

[3] J. Błazewicz, J. Lenstra, and A. Rinnooy Kan, Scheduling subject to resource constraints: Classification and complexity, Discrete Appl Math 5 (1983), 11–24.

[4] W. Herroelen, P. Van Dommelen, and E. Demeulemeester, Project network models with discounted cash flows: A guided tour through recent developments, Eur J Oper Res 100 (1997), 97–121.

[5] P. Brucker, Scheduling Algorithms, Springer, Berlin, 2004.

[6] R. Kolisch and R. Padman, An integrated survey of deterministic project scheduling, OMEGA Int J Manage Sci 29 (2001), 249–272.

[7] S. Hartmann and R. Kolisch, Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem, Eur J Oper Res 127 (2000), 394–340.

[8] R. Kolisch and S. Hartmann, Experimental investigation of heuristics for resource-constrained project scheduling: An update, Eur J Oper Res 174 (2006), 23–37.

[9] R. Kolisch and A. Sprecher, PSPLIB—A project scheduling problem library OR Software—ORSEP Operations Research Software Exchange Program, Eur J Oper Res 96 (1996), 205–216.

[10] D. Goldberg, Genetic algorithms in search, optimization and machine learning, Addison-Wesley Professional, Boston, Massachusetts, 1989.

[11] M. Palpant, C. Artigues, and P. Michelon, LSSPER: Solving the resource-constrained project scheduling problem with large neighborhood search, Ann Oper Res 131 (2004), 237–257.

[12] R. Kolisch, Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation, Eur J Oper Res 90 (1996), 320–333.

[13] R. Klein and A. Scholl, Computing lower bound by destructive improvement: an application to resource-constrained project scheduling problem, Eur J Oper Res 112 (1999), 322–346.

[14] R. Kolisch, A. Sprecher, and A. Drexl, Characterization and generation of a general class of resource-constrained project scheduling problems, Manage Sci 41 (1995), 1693–1703.

[15] Y. Kochetov and A. Stolyar, Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem, in Proceedings of the 3rd International Workshop of Computer Science and Information Technologies, Russia, 2003.

[16] D. Debels, B.R. De Reyck Leus, and M. Vanhoucke, A hybrid scatter search/Electromagnetism meta-heuristic for project scheduling, Eur J Oper Res 169 (2006), 638–653.

[17] D. Debels and M. Vanhoucke, Decomposition-based genetic algorithm for the resource-constrained project-scheduling problem, Oper Res 55 (2007), 457–469.

[18] V. Valls, F. Ballestin, and M.S. Quintanilla, A hybrid genetic algorithm for the resource-constrained project scheduling problem, Eur J Oper Res 185 (2008), 495–508.

[19] V. Valls, F. Ballestin, and M.S. Quintanilla, Justification and RCPSP: A technique that pays, Eur J Oper Res 165 (2005), 375–386.

[20] P. Tormos and A. Lova, A competitive heuristic solution technique for resource-constrained project scheduling, Ann Oper Res 102 (2001), 65–81.

[21] S. Hartmann, A self-adapting genetic algorithm for project scheduling under resource constraints, Nav Res Logist 49 (2002), 433–448.

[22] S. Hartmann, A competitive genetic algorithm for resource-constrained project scheduling, Nav Res Logist 45 (1998), 733–750.

[23] K. Bouleimen and H. Lecocq, A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple modes version, Eur J Oper Res 149 (2003), 268–281.

[24] J. Coelho and L. Tavares, Comparative analysis of metaheuristics for the resource constrained project scheduling problem, Technical report, Department of Civil Engineering, Instituto Superior Tecnico, Portugal, 2003.

[25] R. Kolisch and A. Drexl, Adaptive search for solving hard project scheduling problems, Nav Res Logist 43 (1996), 23–40.

[26] R. Kolisch, Efficient priority rules for the resource constrained project scheduling problem, J Oper Manage 14 (1996), 179–192.

[27] R. Kolisch, Project scheduling under resource constraints—Efficient heuristics for several problem classes, Physica, Heidelberg, 1995.

[28] V.J. Leon and B. Ramamoorthy, Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling, OR Spectrum 17 (1995), 173–182.

[29] V. Valls, F. Ballestin, and M.S. Quintanilla, A population based approach to the resource-constrained project scheduling problem, Ann Oper Res 131 (2004), 305–324.

[30] A. Sprecher, Network decomposition techniques for resource-constrained project scheduling, J Oper Res Soc 53 (2002), 405–414.