World Applied Sciences Journal 7 (8): 987-997, 2009
ISSN 1818-4952
© IDOSI Publications, 2009

A New Efficient Genetic Algorithm for Project Scheduling under Resource Constraints

Mehdi Deiranlou and Fariborz Jolai

Department of Industrial Engineering, Faculty of Engineering, University of Tehran, Tehran, Iran

Abstract: This paper considers a new method for solving resource-constrained project scheduling problem (RCPSP). We propose a new genetic algorithm, called DHGA, to solve RCPSP whereas the objective is to minimize the makespan of project. Our new approach for solving the problem uses a new crossover based on combination of order crossover and partially mapped crossover. The employed swap mutation selects two activities and then swaps their contents. Auto-tuning is considered to automatically adjust rates of crossover and mutation operators. Computational experiments show effectivity of the proposed method.

Key words: Resource Constrained Project Scheduling · Genetic Algorithm · Auto-tuning rates

INTRODUCTION

The resource-constrained project scheduling problem (RCPSP) still lingers to represents an important practice and research problem. Not only exact solution procedures, but also many heuristic methods have been proposed to solve RCPSP. Being an NP-hard problem, Alcaraz and Maroto [1] mentioned that the optimal solution can be achieved by exact solution procedures but only in small projects, usually with less than 60 activities, which are not highly resource constrained. Therefore, heuristic methods are designed to solve large and highly resource-constrained projects. Möhring and. [2] mentioned that RCPSP is one of the most intractable problems in operations research and many optimization techniques have been applied to solve it. Many effective heuristic and metaheuristic algorithms are also proposed for RCPSP.

There are many exact algorithms proposed for RCPSP, which are mainly based on the branch-and-bound strategy. For example, Demeulemeester and Herroelen [3, 4] developed a depth-first branching scheme with dominance criteria and the bounding rules. Brucker and. [5] presented a branch-and-bound algorithm where its branching scheme is applied on a set of conjunctions and disjunctions to pairs of activities. Due to limitations of exact algorithms, some authors proposed heuristic algorithms for RCPSP. For example, Möhring *and.* [2] proposed a heuristic based on the Lagrangian relaxation and minimum cut computations. The heuristic methods which are based on priority rules can be divided into two classes: single-pass methods and multi-pass methods. For example, [6, 7, 8] used single-pass methods while multi-pass method was presented in [9]. The forward–backward methods were also proposed in [10].

There are also numerous metaheuristics proposed for RCPSP. They include Genetic Algorithm (GA), Simulated Annealing, Tabu Search and Ant Colony Optimization. The investigation of Hartmann and Kolisch [11] and its updated version [12] conducted an elaborate study on state-of-the-art heuristic and metaheuristic methods. They presented performance comparisons among heuristic and metaheuristic methods in their study by applying these methods to different standard instance sets, namely J30, J60 and J120, generated by ProGen in the PSPLIB. As shown by the latest experimental evaluation [12], metaheuristic methods outperform heuristic methods.

The use of Tabu Search for RCPSP has had high quality achievements in recent years. Klein [13] developed a so-called reactive Tabu Search method for RCPSP with time-varying resource constraints. It was based on the activity list representation and the serial schedule generation scheme (SGS). The neighborhood was given by swap moves which included the shifting of predecessors or successors of the swapped activities, if the resulting list would otherwise not be precedence feasible. Nonobe and Ibaraki [14] suggested a Tabu Search approach for a generalized variant of RCPSP. Considering only the features that are relevant for the standard RCPSP, the heuristic employed the activity list

Corresponding Author: Mehdi Deiranlou, Department of Industrial Engineering, Faculty of Engineering, University of Tehran, P. O. Box: 11365/4563, Tehran, Iran

representation, the serial SGS, shift moves and a specific neighborhood reduction mechanism. Thomas and Salhi [15] introduced a Tabu Search method which operated directly on schedules. They defined three different moves. Since the resulting neighbor schedules may be infeasible, they employed a repair procedure to turn an infeasible schedule into a feasible one. There are also many instances of the application of Simulated Annealing on RCPSP. For example, Fayer [16] reported fairly good performance by a Simulated Annealing approach on the scheduling problems. Valls and. [17] tested a Simulated Annealing method in a paper that focuses on forward–backward improvement. The first application of Ant Colony to RCPSP was proposed by Merkle *et al.* [18]. In their approach, a single ant corresponds to one application of the serial SGS. The eligible activity to be scheduled next is then selected using a weighted evaluation of the latest start time (LST) priority rule and so-called pheromones, which represent the learning effect of previous ants. Further features of the approach included separate ants for forward and backward scheduling and a 2-opt-based local search phase at the end of the heuristic.

One of the first attempts to apply GA in the scheduling problem was made by Davis [19]. The main idea of his approach was to encode the representation of a schedule in a meaningful and legal way. Alcaraz and Maroto [1] developed a GA based on the activity list representation and the serial SGS. An additional gene was used to decide whether forward or backward scheduling is employed when computing a schedule from an activity list. The crossover operator for activity lists was extended such that a child's activity list could be built up either in forward or in backward direction. Alcaraz and. [20] extended the genetic algorithm of Alcaraz and Maroto [1] by adding two features from the literature. First, they took the additional gene that determines the SGS to be used from Hartmann [21]. Second, they employed the forward–backward improvement of Tormos and Lova [22]. Mendes and. [23] used a random key representation and a modified parallel SGS. The modified parallel SGS determined all activities to be eligible which can be started up to the schedule time plus a delay time. The random key had twice the length of the number of activities and each entry was a random number. The first half of the entries biased the activity selection and the second half biased the delay time of the SGS. They also presented a new genetic algorithm for multi-project scheduling problem [24]. Hindi and. [25] suggested a genetic algorithm based on the activity list representation, the serial SGS and the related order-preserving crossover strategy. Toklu [26] developed a GA which operated directly on schedules (i.e., a vector of start times). Since the genetic operators may produce infeasible offspring schedules, a penalty function was used to evaluate the constraint violations. Valls and. [27] extended the activity list-based GA with forward–backward improvement of Valls and. [17] to what they called a hybrid GA. This is more developed in [28, 29].

There are also some review papers on RCPSP. Kolisch [30] provided an extensive comparison of the parallel and serial scheduling schemes. Computational results of six priority rules under the two scheduling schemes were tested. Herroelen and. [31] conducted surveys of various branch-and-bound algorithms for RCPSP and illustrated extensions to a variety of related problems. Brucker and. [32] and Kolisch and Padman [33] surveyed some exact and heuristic methods for classes of project scheduling problems. See also [34].

The purpose of this paper is to introduce a new GA approach for solving RCPSP and to compare it to existing GA techniques for this problem class. The design of this approach is based on Hartmann [21] and the algorithm focuses on crossover and mutation operators. In fact, the main difference between our approach and [21] is the type of crossover and mutation operators employed. Our main contributions are designing a new crossover based on combination of order crossover and partially mapped crossover, employing the swap mutation by selecting two activities and swapping their contents and auto-tuning for adjusting the rates of crossover and mutation operators. We call our new heuristic as "Development of Hartmann Genetic Algorithm" (DHGA).

The remainder of this paper is organized as follows. We begin with a description of RCPSP in Section 2. Subsequently, DHGA approach is presented in Section 3. Section 4 summarizes the results of our computational investigation. Moreover, we compare it with several project scheduling heuristics from the literature. Finally, Section 5 draws conclusions from this study.

Problem Description: The definition of resource-constrained project-scheduling problem has been given by Wall [35], as follows:

- A project has a number of distinct activities that have known durations.

- A set of precedence constraints are predefined.
- A project has multiple resource types and multiple units of each resource type are required to execute an activity. If activity *j* requires r_{jk} units of resource *k*, then this requirement is constant throughout the duration of *j* and the resource is "shared" so that r_{jk} units of resource *k* are removed from a single pool of available units of resource *k* when activity *j* starts and are returned to this pool when activity *j* finishes.
- A feasible schedule must satisfy both precedence constraints and resource constraints.
- The goal is to find the minimum makespan.
- Activities cannot be interrupted (pre-empted) during execution.

A typical problem instance in resource-constrained project scheduling can be represented on a graph $G = (N,P)$ where *N*, the set of nodes, corresponds to the activities and *P*, the set of arcs, corresponds to the precedence relations. In addition, the duration of each activity and resource requirements for each activity must be specified.

The objective is to minimize the duration of the project that is represented by the finish time of the last activity in the project. That is, the problem is to establish a set of feasible start times for all activities such that the entire project is completed in a minimum makespan. The feasibility of the schedule is established by two sets of constraints: precedence constraints and resource constraints.

Design of Dhga

Basic Scheme: GA is a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. Continuing improvements have made GA attractive for many types of optimization problems. GA is inspired by Darwin's theory of evolution of survival of the fittest. Simply stated, problems are solved by an evolutionary process resulting in the best (fittest) solution (survivor). In other words, the solution is evolved.

Our GA framework starts with the computation of an initial population, i.e., the first generation, which is described in Subsection 3.2. The GA then determines the fitness values of the individuals of the initial population. Afterwards, the population is randomly partitioned into pairs of individuals. To each resulting pair (parent) of individuals, we apply the crossover operator to produce two new individuals (Subsections 3.3 and 3.4). Subsequently, we apply the mutation operator (Subsection 3.5) to the genotypes of the newly produced children. After computing the fitness of each child individual, we may add the children to the current population. Then we apply the selection operator (Subsection 3.6) to reduce the population to its former size. Finally, in subsection 3.7, we use auto-turning to adjust the rates of crossover and mutation operators. The general structure of DHGA is described in the Figure 1.

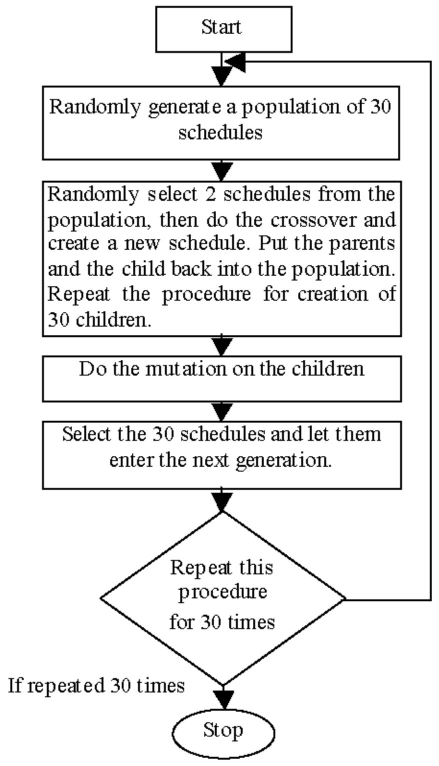
Start
Randomly generate a population of 30 schedules
Randomly select 2 schedules from the population, then do the crossover and create a new schedule. Put the parents and the child back into the population. Repeat the procedure for creation of 30 children.
Do the mutation on the children
Select the 30 schedules and let them enter the next generation.
Repeat this procedure for 30 times
If repeated 30 times
Stop

Fig. 1: Structure of DHGA

Chromosomes Representation and Initial Schedule Generation Scheme: For many optimization problems, GAs do not operate directly on the solutions of the problems. Instead, they make use of problem-specific representations of the solutions. The genetic operators modify the representation which is then transformed into a solution by means of a so-called decoding procedure. During the last few decades, there have been two main chromosome encoding methods for representing the sequence of a set of numbers including Adjacency and Path representation [36]. Each of these chromosome-encoding methods has its own "genetic" operators.

Adjacency representation is designed to facilitate the manipulation of edges. The crossover operators based on this representation generate offspring that inherit most of their edges from the parent chromosomes. The important disadvantage of this representation is that Adjacency representation does not support the classical crossover operators, such as one point or two point crossover, while Path representation supports these operators. Based on the above discussion, we chose the Path representation for presentation of the chromosome of DHGA.

After discussing how to represent the sequence of a set of numbers in the chromosome, we need to discuss how to allocate the resources and start time to the each activity. Therefore, we need a SGS, either a serial or a parallel one, depending on whether it is based on activity incrimination or time incrimination.

Hartmann also pointed out that the activity list representation together with the serial SGS as decoding procedure leads to better results than other representations for the RCPSP [21]. Based on this research, we chose the serial SGS in DHGA.

In short, the encoding and presentation of the chromosome of DHGA can be divided into two stages. In the first stage, we generate chromosomes that do not break the precedence constraints in the form of path representation. Each time DHGA generates an individual allele, it checks whether adding this allele into the chromosome will break the precedence constraints. If using the new allele as the next activity in the schedule breaks the precedence constraints, DHGA will abort this allele and continue to generate new alleles until using the new allele as the next activity in the schedule does not break the precedence constraints.

While DHGA guarantees the precedence constraints in the first stage, in the second stage it considers the resources constraints. In fact, we begin from the first activity in the chromosomes and allocate the resources and start time to each activity in order to complete a feasible schedule and achieve its makespan. If the individual activity has enough resources to begin, we let this activity begin as soon as possible. Otherwise, we delay it until some previous activities finish and free enough resources.

Selection: For crossover, we should determine how the individuals of current population must be selected. There are several variants of selection operator such as ranking method, proportional selection and tournament selection. From the computational studies by researchers, ranking method gives better results than the other alternatives. Therefore, we set the selection component to the ranking approach.

Crossover: For RCPSP, a simple crossover reproduction scheme does not work since it makes the chromosomes inconsistent (some activities may be repeated while others are missed out and hence solutions cannot meet the precedence constraints). The traditional crossover operators like one point crossover are regarded as inappropriate in the study of scheduling problems. In addition to the traditional one or two point crossover, recently two crossover operators were developed for the sequence problem: partially-mapped (PMX) and order (OX) crossovers (For detailed expressions see [36]). However, they can only guarantee that no activities are missed out or visited twice. Sometimes they still break the precedence constraints. The crossover operator used in this paper is derived from them and inherits their merits.

Now, we describe the construction of DHGA crossover operator. Let us assume that two individuals of the current population have been selected for crossover. We set the following two parents to show how this crossover operator performs under the precedence constraints: P1=[1 2 4 | 5 3 6 | 7 8] and P2=[1 3 2 | 6 5 4 | 7 8]. Obviously these two parents obey the precedence constraints. Let's set two crossover points $x = 3$ and $y = 6$ as shown above ("|" means the crossover points). Then we set an empty offspring O where the symbol "X" can be interpreted as "at present unknown"). So we set O= [X X X X X X X X]. First, from 1 to x, we copy P1 to O: O= [1 2 4 | X X X | X X]. Then from y to n, we copy P1 to O: O= [1 2 4 | X X X | 7 8]. In the next step we can set the activities from x to y for offspring O under the precedence constraints. Obviously the job sequences in parents P1 and P2 are valid. So the job sequence in the first part and last sections of the offspring O are valid. For the middle section of offspring O, we can borrow the job sequence from P2. That means if any activities i in P2 have not been in O and all activities before i have been finished at its position in P2, we can set it to the offspring O. Thus we can complete the whole offspring O. We begin to set the first X in the middle section of offspring O. In parent P2, first element 1 has already been copied to offspring O. So we skip the first element of P2 and check the second element 3 of P2. This element is not in O. Since every activity before 3 has been finished, we can set the first X

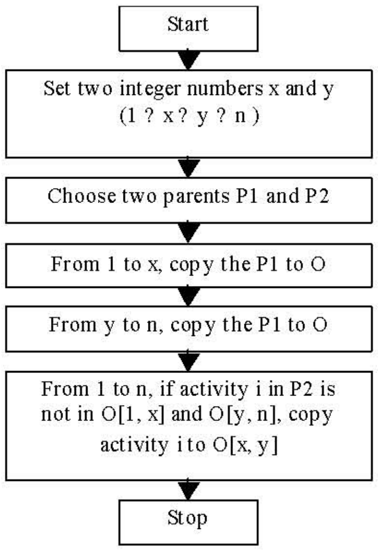
Start
Set two integer numbers x and y (1 ? x? y ? n)
Choose two parents P1 and P2
From 1 to x, copy the P1 to O
From y to n, copy the P1 to O
From 1 to n, if activity i in P2 is not in O[1, x] and O[y, n], copy activity i to O[x, y]
Stop

Fig. 2: Structure of DHGA crossover operator

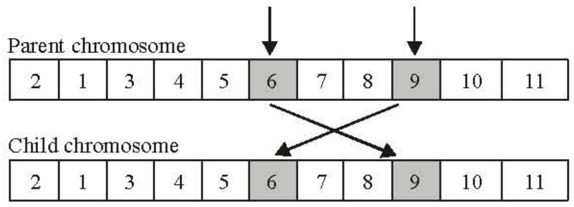

Fig. 3: The swap mutation operator

in the middle section of offspring O to 3. So we can set O= [1 2 4 | 3 X X | 7 8]. We continue to set the second element of X for O. The element after 3 in P2 is 6, because 6 is not in O and all activities before 6 have been finished. We set the second element of X of O to 6. So we can set O= [1 2 4 | 3 6 X | 7 8]. For the same reason we set the last X to 5.

There are three main differences between this operator and PMX and OX. Firstly, both PMX and OX focus on missed or replicated activities. However they do not consider the precedence constraints. The crossover operator used in this paper can guarantee the precedence constraints. Secondly, the crossover operator used in this paper only produces one offspring rather than two and finally, this crossover focuses on the change between the two cut points rather than outside the two cut points.

The crossover of DHGA is similar to Hartmann [21]. However, Hartmann's crossover can create two children. The crossover of DHGA creates only one child. Another important issue is that DHGA only considers the changes in the middle part of the schedules rather than the whole part of solution. Figure 2 depicts the structure of DHGA crossover operator.

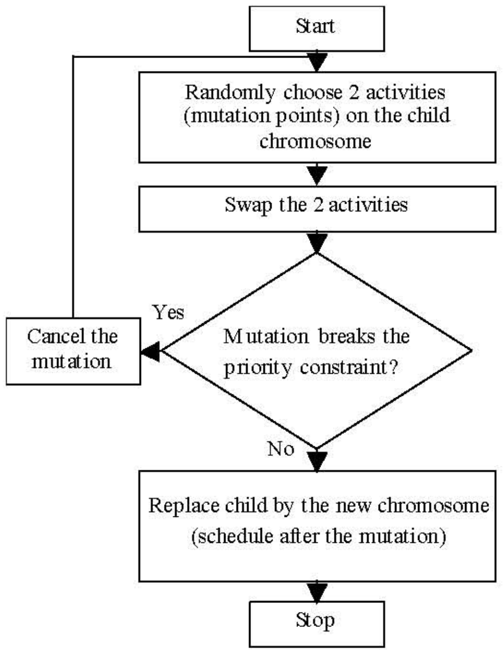
Start
Randomly choose 2 activities (mutation points) on the child chromosome
Swap the 2 activities
Mutation breaks the priority constraint?
Yes
Cancel the mutation
No
Replace child by the new chromosome (schedule after the mutation)
Stop

Fig. 4: Structure of DHGA mutation operator

Mutation: The classic mutation operator is also inappropriate in scheduling problems in terms of precedence constraints as well as crossover operators. Mutation used in this research will first randomly choose two activities and if possible, it swaps them as shown in Figure 3. The reason for using the word "possible" is that the offspring after mutation may be invalid and DHGA checks the precedence constraints. This means that if the swap breaks the precedence constraints, the procedure abandons this switch. Figure 4 shows how DHGA mutation operator works.

Construction of the next Generation: There are several methods for construction of the next generation. Hartmann [37] considered four alternative selection operators which follow a survival-of-the-fittest strategy. The first one is a simple *ranking method*. In this method, chromosomes are sorted based on non-decreasing function of their fitness value. Then keep the *POP* best individuals and remove the remaining ones from the population. The second variant, the *proportional selection*, can be viewed as a randomized version of the previously described ranking technique. Let $f(I)$ be the fitness of an individual I and let P denote the current population, that is, a list containing the individuals. Note that we use a list of individuals instead of a set because we explicitly allow two (or more) distinct individuals with the same genotype in a population. We restore the original population size by successively removing individuals from the population until *POP* individuals are left, using the following probability: Denoting with $f_{\text{best}} = \min\{f(I) | I \in P\}$ the best fitness in the current population, the probability to die for an individual I is given by

$$p_{death}(I) = \frac{(f(I) - f_{best} + 1)^2}{\sum_{I' \in P} (f(I') - f_{best} + 1)^2}$$

The next method is tournament technique. There are two versions of the tournament technique. In the *2-tournament selection*, two different randomly chosen individuals I_1 and I_2 compete for (temporary) survival. If individual I_i is not better than individual I_2, i.e. $f(I_1) \leq f(I_2)$, then it dies and is removed from the population (again, ties are broken arbitrarily). This process is repeated until *POP* individuals are left. Recall that a lower fitness value implies a better quality of the individual.

Finally, the *3-tournament selection* extends the previously described method by randomly selecting three individuals I_1, I_2 and I_3. If we have $f(I_1) \geq f(I_2)$ and $f(I_1) \geq f(I_3)$, individual I_1 is removed from the population. Again, this step is repeated until *POP* individuals are left.

Hartmann [21] denote that the ranking method gave better results than the other alternatives. So we choose this method for the construction of the next generation.

Auto-Tuning the Rates of Crossover and Mutation Operators: For this heuristic, we use the concept of Mak and. [38]. They employed the fitness values of parents and offspring at each generation in order to construct adaptive crossover and mutation operators: this scheme increases the occurrence rates of the crossover and mutation operators, if it consistently produces a better offspring during the genetic search process; however, it also reduces the occurrence rates of the operators, if it produces a poorer offspring. This scheme encourages the well-performing crossover and mutation operators to produce more offspring, while reduces the chance for the poorly performing operators to destroy the potential individuals during the genetic search process. This is the main scheme for constructing DHGA. We employed the 0.05, 0.1, 0.15 and 0.2 for increasing or decreasing the occurrence rates of the crossover and 0.01, 0.015 and 0.02 for mutation. The results show that the amount of 0.05 for crossover and 0.015 for mutation is the best.

The rate of crossover and mutation operators are determined as follows: If t denotes the current generation, $P_C(t)$ and $P_M(t)$ represent the rates of crossover and mutation operators at generation t and $f_{par}(t)$ and $f_{off}(t)$ are the average fitness values of parents and offspring at generation t, the probability of crossover and mutation operators are defined as:

$$If\ f_{par}(t) / f_{off}(t) - 1 \geq 0.1\ Then\ P_C(t+1) = P_C(t) + 0.05, P_M(t+1) = P_M(t) + 0.015 \tag{I}$$

$$If\ f_{par}(t) / f_{off}(t) - 1 \geq 0.1\ Than\ P_C(t+1) = P_C(t) - 0.05, P_M(t+1) = P_M(t) - 0.015 \tag{II}$$

$$If\ -0.1 \leq f_{par}(t) / f_{off}(t) - 1 \leq 0.1\ Then\ P_C(t+1) = P_C(t), P_M(t+1) = P_M(t) \tag{III}$$

In the cases (I) and (II), the adjusted rates should not exceed the range 0.5 to 1.0 for $P_C(t+1)$ and the range 0.00 to 0.10 for $P_M(t+1)$.

The scheme of the procedure defined above is evaluated in all generations during the genetic search process and the occurrence rates of crossover and mutation operators are adaptively regulated according to the results of the above procedure.

Computational Results

Test Design: In this section we present the results of the computational studies. The experiments have been performed on a Pentium-based Intel-compatible personal computer with 2.4 GHz clock-pulse and 128 MB RAM. DHGA for RCPSP has been coded in C++ and tested under Windows XP.

To evaluate the performance of DHGA, we have taken three standard sets of RCPSP instances from the literature, which were constructed by the project generator (ProGen) of Kolisch and. [11, 12]. These instance sets are available from the web-based project scheduling problem library PSPLIB (cf. Kolisch and Sprecher [39]). The first two sets contain 480 instances with 30 and 60 activities per project, respectively. The third one consists of 600 instances with 120 activities. DHGA computed 1000, 5000 and 50000 schedules for each project (with parameter settings *POP* = 30, *GEN* = 30 for 30 and 60 activities per project and *POP* = 90, *GEN* = 55 for 120 activities per project). This test design allowed us to compare our results with those obtained for several RCPSP heuristics from the literature which were tested for the evaluation study of Hartmann and Kolisch [11]. In that study, also 1000, 5000 and 50000 schedules were computed by each heuristic for each instance. This allows

Table 1: Average deviations (%) from optimal makespan—ProGen set J = 30

			Iteration		
Algorithm	SGS	Reference	1000	5000	50000
GA-Hybrid, FBI	Serial	Valls and. [27]	0.27	0.06	0.02
GA-FBI	Serial	Valls and. [17]	0.34	0.20	0.02
GA- FBI	Both	Alcaraz and. [20]	0.25	0.06	0.03
GA-FBI	Serial	Alcaraz and Maroto [1]	0.33	0.12	
Sampling-LFT, FBI	Both	Tormos and Lova [22]	0.25	0.13	0.05
TS-Activity List	Serial	Nonobe and Ibaraki [14]	0.46	0.16	0.05
GA- Self-adapting	Both	Hartmann [21]	0.38	0.22	0.08
GA-Activity List	Serial	Hartmann [37]	0.54	0.25	0.08
GA-Activity List	Serial	Klein [13]	0.42	0.17	
Sampling-Random, FBI	Serial	Valls and. [17]	0.46	0.28	0.11
GA-Activity List (DHGA)	*Serial*	*This Study*	*0.33*	*0.19*	*0.12*
SA-Activity List	Serial	Bouleimen and Lecocq [40]	0.38	0.23	
GA-Late Join	Serial	Coelho and Tavares [41]	0.74	0.33	0.16
Sampling-Adaptive	Both	Schirmer [42]	0.65	0.44	
TS-Schedule Scheme		Baar and. [43]	0.86	0.44	
Sampling-Adaptive	Both	Kolisch and Drexl [44]	0.74	0.52	
GA-Random Key	Serial	Hartmann [37]	1.03	0.56	0.23
Sampling-LFT	Serial	Kolisch [30]	0.83	0.53	0.27
Sampling-Global	Serial	Coelho and Tavares [41]	0.81	0.54	0.28
Sampling-Random	Serial	Kolisch [45]	1.44	1.00	0.54
GA-Priority Rule	Serial	Hartmann [37]	1.38	1.12	0.88
Sampling-WCS	Parallel	Kolisch [30]	1.40	1.28	
Sampling-LFT	Parallel	Kolisch [30]	1.40	1.29	1.13
Sampling-Random	Parallel	Kolisch [45]	1.77	1.48	1.22

Table 2: Average deviations (%) from critical path lower bound—ProGen set J = 60

			Iteration		
Algorithm	SGS	Reference	1000	5000	50000
GA-Hybrid, FBI	Serial	Valls and. [27]	11.73	11.10	10.71
GA-FBI	Serial	Valls and. [17]	12.21	11.27	10.74
GA- FBI	Both	Alcaraz and. [20]	11.89	11.19	10.84
GA- Self-adapting	Both	Alcaraz and Maroto [1]	12.21	11.70	11.21
GA-Activity List	Serial	Tormos and Lova [22]	12.68	11.89	11.23
Sampling-LFT, FBI	Both	Nonobe and Ibaraki [14]	11.88	11.62	11.36
GA-FBI	Serial	Hartmann [21]	12.57	11.86	
SA-Activity List	Serial	Hartmann [37]	12.75	11.90	
GA-Activity List	Serial	Klein [13]	12.77	12.03	
GA-Activity List (DHGA)	*Serial*	Valls and. [17]	*12.19*	*11.88*	*11.47*
TS-Activity List	Serial	*This Study*	12.97	12.18	11.58
Sampling-Random, FBI	Serial	Bouleimen and Lecocq [40]	12.73	12.35	11.94
Sampling-Adaptive	Both	Coelho and Tavares [41]	12.94	12.58	
GA-Late Join	Serial	Schirmer [42]	13.28	12.63	11.94
GA-Random Key	Serial	Baar and. [43]	14.68	13.32	12.25
GA-Priority Rule	Serial	Kolisch and Drexl [44]	13.30	12.74	12.26
Sampling-Adaptive	Both	Hartmann [37]	13.51	13.06	
Sampling-WCS	Parallel	Kolisch [30]	13.66	13.21	
Sampling-Global	Serial	Coelho and Tavares [41]	13.80	13.31	12.83
Sampling-LFT	Parallel	Kolisch [45]	13.59	13.23	12.85
TS-Schedule Scheme		Hartmann [37]	13.80	13.48	
Sampling-LFT	Serial	Kolisch [30]	13.96	13.53	12.97
Sampling-Random	Parallel	Kolisch [30]	14.89	14.30	13.66
Sampling-Random	Serial	Kolisch [45]	15.94	15.17	14.22

Table 3: Average deviations (%) from critical path lower bound—ProGen set J = 120

			Iteration		
Algorithm	SGS	Reference	1000	5000	50000
GA-Hybrid, FBI	Serial	Valls and. [27]	34.07	32.54	31.24
GA- FBI	Both	Valls and. [17]	36.53	33.91	31.49
GA-FBI	Serial	Alcaraz and. [20]	35.39	33.24	31.58
Population Based- FBI	Serial	Alcaraz and Maroto [1]	35.18	34.02	32.81
GA- Self-adapting	Both	Tormos and Lova [22]	37.19	35.39	33.21
Sampling-LFT, FBI	Both	Nonobe and Ibaraki [14]	35.01	34.41	33.71
GA-Activity List (DHGA)	*Serial*	Hartmann [21]	*35.27*	*34.64*	*33.88*
Ant System	Serial	Hartmann [37]		35.43	
GA-Activity List	Serial	Klein [13]	39.37	36.74	34.03
GA-FBI	Serial	Valls and. [17]	39.36	36.57	
TS-Activity List	Serial	*This Study*	40.86	37.88	35.85
GA-Late Join	Serial	Bouleimen and Lecocq [40]	39.97	38.41	36.44
Sampling-Random, FBI	Serial	Coelho and Tavares [41]	38.21	37.47	36.46
SA-Activity List	Serial	Schirmer [42]	42.81	37.68	
GA-Priority Rule	Serial	Baar and. [43]	39.93	38.49	36.51
Sampling-Adaptive	Both	Kolisch and Drexl [44]	39.85	38.70	
Sampling-LFT	Parallel	Hartmann [37]	39.60	38.75	37.74
Sampling-WCS	Parallel	Kolisch [30]	39.65	38.77	
GA-Random Key	Serial	Coelho and Tavares [41]	45.82	42.25	38.83
Sampling-Adaptive	Both	Kolisch [45]	41.37	40.45	
Sampling-Global	Serial	Hartmann [37]	41.36	40.46	39.41
Sampling-LFT	Serial	Kolisch [30]	42.84	41.84	40.63
Sampling-Random	Parallel	Kolisch [30]	44.46	43.05	41.44
Sampling-Random	Serial	Kolisch [45]	49.25	47.61	45.60

Table 4: Average deviations (%) from best solution and lower bound currently known

	J=60			J=120		
Iterations	1000	5000	50000	1000	5000	50000
From best solution currently known	1.31	1.06	0.73	3.76	3.33	2.83
From best lower bound currently known	3.32	3.06	2.72	8.14	7.68	7.16

evaluating the heuristics both in short and medium term optimization. The authors of the heuristics tested their approaches themselves such that they were able to adjust the parameters in order to obtain the best possible results. As the computational effort for constructing one schedule can be assumed to be similar in all of the tested heuristics, this test design should allow for a fair comparison.

Comparison with Other Heuristics for the RCPSP: The results of our experimental study on the ProGen instance sets are summarized in Tables 1–3. They compare DHGA with several RCPSP heuristics from the literature. The information of these tables has been provided by Kolisch and Hartmann [12]. In these tables, each heuristic is briefly described by a few keywords, the SGS employed and the reference. For the J30 set, the results are given in terms of average deviation from the optimal solution. For the other two sets, some of the optimal solutions are unknown. Thus, the average deviation from the well-known critical path-based lower bound is reported. In each table, the heuristics are sorted according to descending performance with respect to 50000 iterations. For DHGA, Table 4 displays the average percentage deviation from the best solutions and lower bounds currently known. The results show that the solution gap is rather small. The bounds are frequently updated in the library PSPLIB of Kolisch and Sprecher [39] (the results of Table 4 are based on the bounds reported there in 2006). These tables clearly demonstrate the superiority of DHGA in comparison with majority of GA based heuristics proposed in the literature.

CONCLUSION

In this paper, we presented a new GA based heuristic called DHGA for the classical resource-constrained project scheduling problem. The building blocks of DHGA were procedures such as its new crossover which is based on combination of order crossover and partially mapped crossover, swap mutation which is realized by selecting two activities and swapping their contents and the auto-tuned rates of crossover and mutation operators. The computational experiments on a large set of standard test instances showed that the proposed heuristic leads to competitive results compared to several heuristics from the literature. For solving RCPSP, DHGA is the 4th best for J30, J60 and J120 in the comparison lists where comparisons are based on evaluating 1000 schedules. Also, DHGA is the 8th best for J30, 7th best for J60 and 6th best for J120 in 5000 schedules.

From this Study, it Is Observed That:

- The representation of the chromosome, the crossover and the mutation operator in DHGA obey the precedence and resource constraints. This means that after crossover, the offspring solutions satisfy precedence constraints. This is a very critical issue for RCPSP as many other alternative GA methods cannot handle it.
- The large population size does not significantly improve the performance of GA. When we set the population size and other parameter settings of GA, we need to consider the number of tasks in the project and the size of chromosomes. It's unnecessary to use large population sizes when the number of tasks in the project is small. This can avoid unnecessary computational work.

The findings of this research can be readily employed to develop more efficient GA based methods. They also can be used to create more effective crossover operators for other scheduling instances like job-shop and flow-shop problems.

REFERENCES

1. Alcaraz, J. and C. Maroto, 2001. A robust genetic algorithm for resource allocation in project scheduling. Annals of Operations Research, 102: 83-109.
2. Möhring, R.H., A.S. Schulz, F. Stork and M. Uetz, 2003. Solving project scheduling problems by minimum cut computations. Management Science, 49(3): 330-350.
3. Demeulemeester, E. and W. Herroelen, 1992. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Management Science, 38(12): 1803-1818.
4. Demeulemeester, E. and W. Herroelen, 1997. New benchmark results for the multiple resource-constrained project scheduling problem. Management Science, 43(11): 1485-1492.
5. Brucker, P., S. Knust, A. Schoo and O. Thiele, 1998. A branch-and-bound algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research, 107: 272-288.
6. Kolisch, R., 1996. Efficient priority rules for the resource-constrained project scheduling problem. Journal of Operations Management, 14: 179-192.
7. Özdamar, L. and G. Ulusoy, 1996. An iterative local constraint based analysis for solving the resource constrained project scheduling problem. Journal of Operations Management, 14(3): 193-208.
8. Ranjbar, M., 2008. Solving the resource-constrained project scheduling problem using filter-and-fan approach, Applied Mathematics and Computation, 201: 313-318.
9. Boctor, F.F., 1990. Some efficient multi-heuristic procedures for resource-constrained project scheduling. European Journal of Operational Research, 49: 3-13.
10. Li, K.Y. and R.J. Willis, 1992. An iterative scheduling technique for resource-constrained project scheduling. European Journal of Operational Research, 56: 370-379.
11. Kolisch, R. and S. Hartmann, 2004. Experimental investigation of heuristics for resource-constrained project scheduling: An update. working paper, Technical University of Munich, Germany.
12. Kolisch, R. and S. Hartmann, 2006. Experimental investigation of heuristics for resource-constrained project scheduling: An update. European Journal of Operational Research, 174: 23-37.
13. Klein, R., 2000. Project scheduling with time-varying resource constraints. International Journal of Production Research, 38(16): 3937-3952.
14. Nonobe, K. and T. Ibaraki, 2002. Formulation and Tabu Search algorithm for the resource constrained project scheduling problem, in: Ribeiro, C.C. and P. Hansen (Eds.), 2002. Essays and Surveys in Metaheuristics. Kluwer Academic Publishers.

15. Thomas, P.R. and S. Salhi, 1998. A Tabu search approach for the resource constrained project scheduling problem, 1998. Journal of Heuristics, 4: 123-139.
16. Fayer, F.B., 1990. Some efficient multi-heuristic procedures for resource-constrained project scheduling. European Journal of Operational Research, 49(1): 3-13.
17. Valls, V., F. Ballestin and M.S. Quintanilla, 2005. Justification and RCPSP: A technique that pays. European Journal of Operational Research, 165: 375-386.
18. Merkle, D., M. Middendorf and H. Schmeck, 2002. Ant colony optimization for resource-constrained project scheduling. IEEE Transactions on Evolutionary Computation, 6: 333-346.
19. Davis, L., 1985. Job shop scheduling with genetic algorithms. Proceeding of the First International Conference on genetic algorithms, Carne-Mellon University, Pittsburg, PA.
20. Alcaraz, J., C. Maroto and R. Ruiz, 2004. Improving the performance of genetic algorithms for the RCPS problem. in: Proceedings of the Ninth International Workshop on Project Management and Scheduling, pp: 40-43.
21. Hartmann, S., 2002. A self-adapting genetic algorithm for project scheduling under resource constraints. Naval Research Logistics, 49: 433-448.
22. Tormos, P. and A. Lova, 2001. A competitive heuristic solution technique for resource-constrained project scheduling. Annals of Operations Research, 102: 65-81.
23. Mendes, J.J.M., J.F. Gonçalves and M.G.C. Resende, 2009. A random key based genetic algorithm for the resource constrained project scheduling problem. Computers and Operations Research, 36(1): 92-109.
24. Mendes, J.J.M. and J.F. Gonçalves, 2008. A genetic algorithm for the resource constrained multi-project scheduling problem. European Journal of Operational Research, 189: 1171-1190.
25. Hindi, K.S., H. Yang and K. Fleszar, 2002. An evolutionary algorithm for resource-constrained project scheduling. IEEE Transactions on Evolutionary Computation, 6: 512-518.
26. Toklu, Y.C., 2002. Application of genetic algorithms to construction scheduling with or without resource constraints. Canadian Journal of Civil Engineering, 29: 421-429.
27. Valls, V., F. Ballestin and M.S. Quintanilla, 2003. A hybrid genetic algorithm for the RCPSP. Technical report, Department of Statistics and Operations Research, University of Valencia.
28. Valls, V., F. Ballestin and M.S. Quintanilla, 2008. A hybrid genetic algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research, 185: 495-508.
29. Tseng L., S-C. Chen, 2006. A hybrid metaheuristic for the resource-constrained project scheduling problem. European Journal of Operational Research, 175(2): 707-721.
30. Kolisch, R., 1996. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. European Journal of Operational Research, 90: 320-333.
31. Herroelen, W., B. De Reyck and E. Demeulemeester, 1998. Resource-constrained project scheduling: A survey of recent developments. Computers and Operations Research, 25(4): 279-302.
32. Brucker, P., A. Drexl, R. Möhring, K. Neumann and E. Pesch, 1999. Resource-constrained project scheduling: Notation, classification, models and methods. European Journal of Operational Research, 112: 3-41.
33. Kolisch, R. and R. Padman, 2001. An integrated survey of deterministic project scheduling. OMEGA, 29: 249-272.
34. Kis T., 2005. Project scheduling: a review of recent books. Operations Research Letters, 33(1): 105-110.
35. Wall, M.B., 1996. A Genetic Algorithm for Resource-Constrained Scheduling, Ph.D. paper, Massachusetts Institute of Technology.
36. Goldberg, D.E. and R. Lingle, Jr., 1985. Alleles, Loci and the Traveling Salesman Problem. Proceeding of the First International Conference on Genetic Algorithms, Carne-Mellon University, Pittsburg, PA.
37. Hartmann, S., 1998. A competitive genetic algorithm for resource-constrained project scheduling. Naval Research Logistics, 45: 733-750.
38. Mak, K.L., Y.S. Wong and X.X. Wang, 2000. An adaptive genetic algorithm for manufacturing cell formation, International Journal of Manufacturing Technology, 16: 491-497.
39. Kolisch, R. and A. Sprecher, 1996. PSPLIB–A project scheduling problem library. European Journal of Operational Research, 96: 205-216.

40. Bouleimen, K. and H. Lecocq, 2003. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple modes version. European Journal of Operational Research, 149: 268-281.
41. Coelho, J. and L. Tavares, 2003. Comparative analysis of metaheuristics for the resource constrained project scheduling problem. Technical report, Department of Civil Engineering, Instituto Superior Tecnico, Portugal.
42. Schirmer, A., 2000. Case-based reasoning and improved adaptive search for project scheduling. Naval Research Logistics, 47: 201-222.
43. Baar, T., P. Brucker and S. Knust, 1998. Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem. in: Voss, S., S. Martello, I. Osman and C. Roucairol (Eds.), 1998. Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization. Kluwer Academic Publishers.
44. Kolisch, R. and A. Drexl, 1996. Adaptive search for solving hard project scheduling problems. Naval Research Logistics, 43: 23-40.
45. Kolisch, R., 1995. Project scheduling under resource constraints: efficient heuristics for several problem classes. Physica-Verlag.