Complexity comparison of integer programming and genetic algorithms for resource constrained scheduling problems

Rebeka Čorić, Mateja Đumić
Department of Mathematics, University of Osijek
Osijek, Croatia
Email: rcoric/mdjumic@mathos.hr

Domagoj Jakobović
Faculty of Electrical Engineering and Computing
Zagreb, Croatia
Email: domagoj.jakobovic@fer.hr

***Abstract*—Resource constrained project scheduling problem (RCPSP) is one of the most intractable combinatorial optimization problems. RCPSP belongs to the class of NP hard problems. Integer Programming (IP) is one of the exact solving methods that can be used for solving RCPSP. IP formulation uses binary decision variables for generating a feasible solution and with different boundaries eliminates some of solutions to reduce the solution space size. All exact methods, including IP, search through entire solution space so they are impractical for very large problem instances. Due to the fact that exact methods are not applicable to all problem instances, many heuristic approaches are developed, such as genetic algorithms. In this paper we compare the time complexity of IP formulations and genetic algorithms when solving the RCPSP. We present two different solution representations for genetic algorithms, permutation vector and vector of floating point numbers. Two formulations of IP and and their time and convergence results are compared for the aforementioned approaches.**

I. Introduction

Scheduling is process which all of us use on everyday basis. In scheduling the goal is to allocate resources to tasks over given time periods. Usually the resources that we have have some limitations. For example we have limited project budget, limited number of employees, vehicle capacity etc.

This work will focus on one of the problems with constraints known as resource constrained project scheduling problem (RCPSP). RCPSP is a problem in which one has to deal with two kinds of constraints - precedence and resource. In this scheduling process we define starting point of some activity taking care of its precedence and amount of resources that are available.

RCPSP belongs to class of NP hard problems [1] and that, with fact that it is common problem in everyday world, results with numerous methods for solving it. Which method to use is a decision that has to be made upon the problem characteristics. In literature two different groups of solving methods can be found - exact methods and heuristics [2].

Exact methods search the entire space of feasible solutions and guarantee the optimality [3], but because of size of solution space they are impractical and almost useless for problems with large instance size [4]. Because of that many heuristic methods were developed. Heuristics do not search the entire solution space and do not guarantee the optimality but they are fast and can result with good enough solutions for the specific problem.

Heuristics can be divided into two groups: priority rule-based methods (or constructive heuristics) and metaheuristic-based approaches (or improvement heuristics). Methods in the first group start with empty schedule, and iteratively put in each step one activity based on it's priority, until unscheduled activities exists. The second group starts with initial complete solution and in each iteration tries to improve it.

In this work we will present two IP formulations as representatives of exact methods, and Genetic algorithm (GA) as representative of heuristics methods. In GA approach two different solution representations will be used. Result comparison and time execution comparison will be given for these methods.

This paper is organized as follows: in section 2 a short overview of solving methods for RCPSP is given. The definition of RCPSP can be found in section 3. In section 4 two different IP formulations are given, and GA adaption for RCPSP is explained in section 5. Experimental results together with implementaion details of this methods are given in section 6. Finally in section 7 a short conclusion is given.

II. Related work

Many researchers investigate RCPSP and a lot of work for exact and heuristic methods can be found in literature.

Exact methods can be divided in four different groups:

1) Integer Programming - using large number of binary variables [5], [6]
2) Implicit Enumeration - using enumeration tree and bounds to reduce the space of feasible solutions [7], [8]
3) Branch-and-bound methods - using trees and lower bounds to eliminate nodes in tree that cannot lead to optimal solution [9], [10]
4) Dynamic programming - divide problem on problems with smaller size and solve that problems, and combine the solution [11].

Some of heuristic methods that can be found in literature are: Genetic algorithms [12], [13], Simulated annealing

[14], Ant colony [15]. For achieving better results there are attempts of combining different solution methods like hybrid algorithms, memetic algorithms etc. Usually these approaches combine some evolutionary algorithm and local search methods. One of newer work shows that combining local search method with GA in Genetic-Local Search Algorithm gives good results [16].

III. RCPSP

RCPSP is a problem in which we need to schedule activities which have their duration and resource demands, while all resources have limited amount that can be used in one time period. Beside of the limited resource amount, there are precedence constraints that have to be satisfied, meaning that every activity has predecessors that need to be finished before it starts. The goal of scheduling is to find a feasible schedule that minimizes some objective, by assigning start times to each activity that satisfies the precedence and resource constraints.

Formally, RCPSP can be defined as a combinatorial optimization problem as tuple [17]:

$$RCPSP = (A, E, p, R, B, D, c). \tag{1}$$

In this tuple A denotes the set of all activities $A = \{A_0, A_1, \ldots, A_{n+1}\}$, where activities A_0 and A_{n+1} are dummy activities that represent the start and the end of a schedule. Vector $p \in \mathbb{N}_0^{n+2}$ is vector of durations, in which component p_i is duration of activity A_i. Dummy activities have duration 0.

The precedence relations between activities are given by set E. Set E consists of pairs of activities (A_i, A_j) which mean that activity A_i precedes activity A_j. The assumption for dummy activities is that activity A_0 precedes all other activities and that A_{n+1} succeeds all other activities. So, by determining the start point of A_{n+1} the total project duration is found.

Resources that are needed for activities are given by set $R = \{R_1, \ldots, R_r\}$, and amount of each resource that is available in all time period is given by vector $B \in \mathbb{N}^r$. Demands on resources for each activity are given by matrix $D \in \mathbb{N}_0^{(n+1)\times r}$.

The objective function is $c : \chi \subseteq \mathbb{R}^{n+2} \to \mathbb{R}$, where χ is a feasible schedule. Sometimes this tuple member is omitted, in which case the objective function which needs to be minimized is project makespan (total project duration). The solution is a schedule $S \in \chi$, where component S_i of vector S represents the start time of activity A_i. When vector S is known it is easy to find vector of completion times of activities denoted by C, by computing component $C_i = S_i + p_i$.

If we mark the set of all activities which are active at a time t with $A_t = \{A_i \in A : S_i \leq t \leq S_i + p_i\}$ RCPSP can be defined as a problem of finding a feasible schedule S with minimal makespan S_{n+1} for which following constraints are satisfied:

$$S_j - S_i \geq p_i \quad \forall (A_i, A_j) \in E \tag{2}$$

$$\sum_{A_i \in A_t} D_{ij} \leq B_j \quad \forall t \geq 0, \forall R_j \in R. \tag{3}$$

For further problem analysis we also need to define set $E(t)$ - the set of eligible activities at time t. For definition of that set we need to define the following properties of each activity:

- ES_j - the earliest start time of activity A_j
- EF_j - the earliest finish time of activity A_j
- LS_j - the latest start time of activity A_j
- LF_j - the latest finish time of activity A_j.

To calculate mentioned properties the horizon (the total project time) T, must be known. If it is not given, it can be calculated by some heuristic or using the formula

$$T = \sum_{j=1}^{n} p_j. \tag{4}$$

With setting $ES_0 = EF_0 = 0$ for calculating ES_j and EF_j we can use the following formulas:

$$ES_j = max\{EF_i : i \in P_j\} \tag{5}$$

$$EF_j = ES_j + p_j, j \in \{1, \ldots, n\}, \tag{6}$$

where P_j is set of activities that directly precede the activity A_j.

Similarly, for calculating LS_j and LF_j we set $LF_n = LS_n = T$ and use formulas:

$$LF_j = min\{LS_i : i \in F_j\} \tag{7}$$

$$LS_j = LF_j - p_j, j \in \{n-1, \ldots, 1\}, \tag{8}$$

where F_j is set of activities that directly succeed the activity A_j.

Now we can define $E(t)$ as:

$$E(t) = \{A_j : A_j \in A, ES_j + 1 \leq t \leq LF_j\}. \tag{9}$$

IV. IP FORMULATION OF RCPSP

IP is a method in which number of binary variables are introduced and used for creating a schedule. In this section two different IP formulations of RCPSP will be given.

A. Formulation 1

The first IP formulation was given by Pritisker et al. [6] in 1969. In this formulation, the solution of RCPSP is represented with series of zero-one variables x_{jt}, $j \in J$ (J is set of activity indexes), $t \in [EF_j, LF_j]$. Variables are defined as:

$$x_{jt} = \begin{cases} 1, & \text{if activity } A_j \text{ is completed at the end of period t} \\ 0, & \text{otherwise.} \end{cases}$$

The variable x_{jt} can have the value 1 just in a time period in which the activity A_j can be completed so it is introduced just for $t \in [EF_j, LF_j]$. Time in which activity A_j finished is given by:

$$\sum_{t=EF_j}^{LF_j} t \cdot x_{jt} \tag{10}$$

IP formulation for RCPSP can be now given as:

$$\sum_{t=EF_n}^{LF_n} t \cdot x_{nt} \rightarrow \min \tag{11}$$

subject to:

$$\sum_{t=EF_j}^{LF_j} x_{jt} = 1, \;\; j \in J \tag{12}$$

$$\sum_{t=EF_j}^{LF_j} (t - p_j)\, x_{jt} - \sum_{t=EF_i}^{LF_i} t \cdot x_{it} \geq 0, \;\; j \in J, i \in P_j \tag{13}$$

$$\sum_{j \in E(t)} u_{jr} \cdot \sum_{q=\max\{t,EF_j\}}^{\min\{t+p_j-1,LF_j\}} x_{jq} \leq a_r, \;\; r \in R, t \in [1, \ldots, T] \tag{14}$$

$$x_{jt} \in \{0,1\}, \;\; j \in J, t \in [EF_j, LF_j] \tag{15}$$

From (11) it is clear that in this formulation one minimizes the completion time of the last activity (makespan). If the completion time of the last activity is minimal then the completion time for all activities is minimal too. Constraints (12) and (15) ensure that the activity can be completed exactly in one time period. Constraint (13) ensures that all predecessors of activity A_j are completed before it starts with execution, and constraint (14) ensures that in each time period the amount of resources being used is smaller or equal to the available amount of resources.

B. Formulation 2

RCPSP can be formulated as IP using variables y_{jt}, which can also only become 0 or 1 for $j \in J$, $t \in [ES_j + 1 - p_j, LF_j]$ and are defined in the following way:

$$y_{jt} = \begin{cases} 1, & \text{if activity } A_j \text{ is completed at the beginning} \\ & \text{of period t or earlier} \\ 0, & \text{otherwise.} \end{cases}$$

For activity A_j there is vector with $LF_j - ES_j - 1 + p_j$ components with two blocks: the first consists of zeroes, and the second consists of ones. Time period in which change between these two blocks happens is the time point in which activity A_j started. In accordance with that, the time point in which activity A_j starts is given with:

$$LF_j - \sum_{t=ES_j+1}^{LF_j} y_{jt} \tag{16}$$

IP formulation of RCPSP with introduced variables is:

$$T - \sum_{t=ES_n+1}^{LS_n} y_{nt} \rightarrow \min \tag{17}$$

subject to:

$$y_{j,t+1} - y_{jt} \geq 0, \;\; j \in J, t \in \{ES_j + 1, \ldots LS_j - 1\} \tag{18}$$

$$y_{i,t-p_i} - y_{jt} \geq 0, \;\; j \in J, i \in P_j, t \in \{ES_j + 1, \ldots LF_i\} \tag{19}$$

$$\sum_{j \in E(t)} u_{jr} \cdot \left(y_{jt} - y_{j,t-p_j}\right) \leq a_r, \;\; r \in R, t \in [1, \ldots, T] \tag{20}$$

$$y_{jt} = 0, \;\; j \in J, t \in [ES_j + 1 - p_j, ES_j] \tag{21}$$

$$y_{jt} = 1, \;\; j \in J, t \in [LS_j + 1, LF_j] \tag{22}$$

$$y_{jt} \in \{0,1\}, \;\; j \in J, t \in [EF_j, LF_j] \tag{23}$$

Objective function is given with (17) and minimizes makespan. Constraints (18) and (21-23) ensure that activity A_j starts in one time point only. Constraint (19) ensures that all predecessors $A_i \in P_j$ of activity A_j start at least p_i time units earlier. Constraint (20) ensures that resource usage in every time period satisfies the amount of resource which can be used.

V. GA FOR RCPSP

In this section, genetic algorithms and how they work will briefly be explained.

The idea for genetic algorithms (GAs) came from the principles of natural selection and genetics. Because of that, certain terms in GAs are equal to the terms one can see in genetics. In GAs decision variables are coded as finite dimensional vectors which come from an alphabet of some cardinality. Those vectors are called chromosomes, parts of the alphabet are called genes, and the values of the genes are called alleles. E.g. for the Traveling Salesman Problem (TSP), one chromosome represents a route and one gene represents a city in a route [18]. In GAs solutions can be represented in various ways. Some of them are [19]: array of bits, permutation array, matrix representation, floating point vector, integer vector etc.

In order to get better solutions in the process of natural selection, there should exist some measure which will determine which solutions are better than the others and then lead to better solutions in GAs. Genetic algorithms use population of solutions rather than only one solution to find optimum of some problem. It is necessary to determine the optimal size of population, which is an optimization problem by itself. After determining the size of population and chromosome coding as well as some fitness measure for solution candidates, algorithm can start to evolve solutions for a given problem using the following steps [18]:

1) Initialization: initial population of solutions is generated randomly, or if some additional information about solution space is known, that knowledge is incorporated in generating better initial population
2) Evaluation: after initialization, the fitness of each population member is evaluated

3) Selection: selection chooses few of the better solutions and that way one can secure survival of the fittest
4) Crossover: crossover operators combine parts of two or more parents in order to create one or more children solutions
5) Mutation: mutation modifies existing solution in order to keep diversity in population
6) Replacement: population of children obtained in selection, crossover and mutation, replaces old population of parents

After that, evaluation, selection, mutation and replacement are repeated until some of stopping criteria is met.

GAs can be sequential and parallel [19]. Sequential GAs can be steady-state GA or generational GA. Steady-state GA chooses two parents from population, makes one child, mutates it and evaluates it. Child can then be inserted in population (in that case some other solution candidate gets thrown out) or rejected. In generational GA in every step whole new population of children is created which replaces old parent generation. If in that process one does not want to lose best found solution, elitism can be introduced, i.e. the best current known solution is always put in child population. In GA implementation used for this work, a sequential steady-state GA with elitism is used.

VI. Experimental Results

A. Integer Programming

For solving IP problems it's common to use cutting plane methods and variants of the branch and bound method. In this work Gurobi [20] was used for solving IPs. The solvers in the Gurobi Optimizer use parallelism while solving problems. The process of solving can be divided in two phases: presolving and solving. Presolving phase is a phase in which problem is simplified which can significantly reduce the time needed for solving problem. Methods that are used in this phase are: removal of empty rows and columns, finding and removing rows dominated by linear combination of other rows etc. In the solving phase Gurobi uses branch and bound and cutting plane methods for solving new reduced problem created in presolve phase.

B. Genetic Algorithm

In GA implementation for this problem two solution representations were used: permutation representation (as proposed in [21]) and floating point vector representation. In permutation representation, every candidate solution is represented as integer vector which contains numbers from 0 to $n+1$, where n denotes number of jobs. E.g. for $n = 5$, $\{0, 2, 3, 5, 1, 4, 6\}$ means that job 2 is executed first, after that job 3 is executed etc. In floating point vector representation, every candidate solution is presented as vector which contains numbers between 0 and 1. The higher the number, the more important it is to start the job earlier. E.g. $(0.3, 0.95, 0.17, 0.82, 0.5)$ means that jobs should be executed in the following order: $(2, 4, 5, 1, 3)$. In order to get feasible solutions in the initial population (solutions that meet precedence constraints and resource constraints) for both representations, serial schedule generation scheme has been used which schedules a job in earliest possible time in which both precedence constraints and resource constraints for given job are met. Due to the lack of space, serial schedule generation scheme pseudocode is omitted but can be found in [22].

For GA tests, Evolutionary Computation Framework[1] (ECF) was used with the following setup: 3-tournament selection was used, where in each iteration three random individuals are chosen and the worst of the three is eliminated. A new individual is created using crossover from the remaining two, and is subjected to mutation with a given probability. In the permutation encoding, order-based crossover operators OBX and OX2 as well as cycle crossover where used as crossover operators. The mutation operator randomly chooses two genes in an individual and swaps their positions. For the floating point vector a discrete crossover was used and a simple mutation that randomly recreates a single vector element. More information on the genetic operators is available at the framework website.

A short tuning phase was conducted for both representations, which resulted in using the population size of 1000 individuals and a mutation rate of 0.7 (i.e. on average 7 out of 10 new individuals will get mutated). Rather than using a maximum number of generations or evaluations, the stopping criterion is set to a common time limit (of 8 hours) for the purpose of comparison with the IP solver.

C. Benchmark

Implementation of methods mentioned in this work were tested on problems from PSBLIB Library. This library consists of different sets of RCPSP with their so far known optimal or heuristic results. Problems in PSBLIB Library are divided in 4 groups based on number of jobs. The first group consists of problems with 30 activities, the second with 60 activities, the third one has problems with 90 activities and last group problems with 120 activities. More about this library and generation of problem instances can be found in [23].

D. Result and execution time comparison

Tests were conducted on 48 problem instances from group of 30 activities and on 48 problem instances from group of 90 activities. Results are given in tables I and II.

For IP formulations tables show execution times if solution was found in under 8 hours, otherwise the x is put in corresponding row. To make the comparison fair, for the genetic algorithm the stopping criterion was set at the same time limit or achieving the optimal solution (if it is known in advance). Furthermore, the GA was automatically restarted (under the time limit) if no improvement was achieved in the last 500 generations. The GA results in Tables I and II present the best found solution in the

[1] http://ecf.zemris.fer.hr/

TABLE I
PROBLEMS WITH 30 ACTIVITIES

Instance #	Opt. solution	Formulation 1		Formulation 2		GA - permutation		GA - floating point	
		Presolve phase (in s)	Solve phase (in s)	Presolve phase (in s)	Solve phase (in s)	Solution	Time (in s)	Solution	Time (in s)
1	43	0.07	0.47	0.14	0.31	43	0.41	43	1
2	51	0.08	0.37	0.34	0.63	51	0.48	51	1
3	57	0.10	0.13	0.21	0.31	57	0.44	57	1
4	45	0.03	0.04	0.16	0.24	45	0.41	45	1
5	82	0.16	176.50	0.62	144.45	82	0.44	82	1
6	67	0.19	2.81	0.62	5.01	67	0.45	67	1
7	44	0.08	0.33	0.21	0.33	44	0.42	44	1
8	53	0.11	0.12	0.27	0.37	53	0.43	53	1
9	63	0.15	2470.00	0.48	2155.83	63	5.65	63	20
10	58	0.21	3.99	0.18	3.22	58	2.90	**59**	**125**
11	62	0.26	0.64	0.21	0.90	62	0.45	62	1
12	53	0.25	0.26	0.18	0.32	53	0.42	53	1
13	77	x		x		77	5.43	77	1
14	50	0.19	12.22	0.32	32.43	50	0.69	50	1
15	47	0.24	0.27	0.36	0.48	47	0.45	47	1
16	51	0.32	0.33	0.39	0.45	51	0.44	51	1
17	47	0.07	2.17	0.35	1.92	47	0.44	47	1
18	56	0.08	0.24	0.53	0.65	56	0.45	56	1
19	49	0.04	0.10	0.41	0.57	49	0.41	49	1
20	41	0.03	0.03	0.17	0.21	41	0.41	41	1
21	69	0.14	99.00	0.50	172.20	69	0.38	69	1
22	42	0.06	0.65	0.28	0.54	42	0.75	42	1
23	65	0.12	0.85	0.73	1.01	65	1.14	65	2
24	58	0.15	0.16	0.34	0.60	58	0.45	58	1
25	72	0.16	2159.31	1542.38	0.66	72	1.33	72	3
26	58	0.15	0.27	0.43	1.59	58	0.41	58	1
27	64	0.21	0.50	0.49	0.88	64	0.40	64	1
28	62	0.32	0.33	0.47	0.75	62	0.46	62	1
29	78	x		x		78	3.47	78	18
30	53	0.25	3.46	0.43	15.09	53	1.73	53	13
31	63	0.26	0.34	0.54	0.71	63	0.43	63	1
32	54	0.36	0.37	0.46	0.61	54	0.43	54	1
33	60	0.12	0.66	0.37	0.73	60	0.43	60	1
34	44	0.05	0.15	0.25	0.46	44	0.40	44	1
35	57	0.08	0.16	0.29	0.49	57	0.45	57	1
36	46	0.04	0.05	0.22	0.30	46	0.42	46	1
37	81	0.17	62.41	0.95	229.26	81	0.40	81	1
38	63	0.13	0.70	0.42	1.59	63	0.44	63	1
39	54	0.10	0.17	0.35	0.56	54	0.44	54	1
40	51	0.14	0.15	0.30	0.48	51	0.42	51	1
41	88	0.23	1954.62	1.04	2385.91	88	1.41	88	9
42	66	0.24	6.13	0.54	6.24	66	1.48	66	8
43	43	0.12	0.17	0.30	0.42	43	1.10	43	1
44	57	0.29	0.30	0.46	0.59	57	0.44	57	1
45	92	0.26	274.07	1.44	3819.38	92	2.46	92	15
46	67	0.38	4.75	1.14	117.24	67	0.77	67	2
47	49	0.13	1.00	0.44	0.99	49	0.42	49	2
48	50	0.28	0.29	0.46	0.63	50	0.44	50	1

8 hours alloted to the algorithm, regardless of the number of restarts.

It can be seen that when using IP for problems with 90 activities the number of instances that are not solved in under 8 hours is bigger than in problems with 30 activities, which supports the fact that instances of larger size are more difficult to solve. If we compare formulation 1 and formulation 2, we can see that formulation 1 gives results in less time than formulation 2. However, it is worth noticing that there is no problem instance which is solved by one formulation and not by the other.

The tables also show results and execution times for both solution representations in GA. The GA was always able to reach the known optimum solution for 30 activity problems, whereas it did not reach the optimum in ten of the problem instances with 90 activities.

If we compare the two solution representations we can see that the permutation representation obtains slightly better results. This is evident from the deviation of the obtained results from the optimum solutions: for the cases where the optimum was not obtained, the permutation encoding exhibits an average deviation of 7.16%, while for the floating point it averages 8.8%.

To better illustrate the average performance of the two GA representations, Table III shows the results of both variants on the test instances from Table II in which the GA did not manage to obtain the optimal solution. The data in the table indicates the standard statistical properties obtained with running both variants in 20 repetitions with the stopping criterion of 10^6 evaluations.

TABLE II
PROBLEMS WITH 90 ACTIVITIES

Instance#	Opt. solution	Formulation 1		Formulation 2		GA - permutation		GA - floating point	
		Presolve phase (in s)	Solve phase (in s)	Presolve phase (in s)	Solve phase (in s)	Solution	Time (in s)	Solution	Time (in s)
1	92	0.36	5.84	1.05	19.38	92	16	92	12
2	70	0.35	1.01	0.64	1.85	70	5	70	13
3	87	0.45	2.36	0.85	1.85	87	6	87	13
4	78	0.35	0.39	0.84	1.09	78	5	78	13
5	105	x		x		**110**	**382**	**110**	**2008**
6	71	0.34	2.58	1.22	6.69	71	5	71	12
7	90	0.64	1.14	1.56	2.83	90	5	90	13
8	70	0.95	0.99	1.24	1.90	70	5	70	13
9	110	x		x		**123**	**10140**	**124**	**1812**
10	95	1.41	8.27	1.95	54.73	95	22s	95	81
11	99	1.36	2.43	2.12	12.00	99	5	99	13
12	83	1.47	1.65	1.66	2.54	83	5	83	14
13	112	x		x		**121**	**10710**	**124**	**1924**
14	94	1.61	2.32	2.24	10.28	94	5	94	13
15	92	1.73	2.67	2.06	4.45	92	6	92	14
16	71	1.66	1.97	1.55	3.69	71	5	71	14
17	94	0.43	45.71	1.20	21.73	94	5	94	150
18	90	0.41	0.99	1.08	13.91	90	5	90	13
19	66	0.18	0.51	0.78	1.76	66	5	66	11
20	83	0.36	0.41	1.03	1.78	83	6	83	14
21	124	x		x		**127**	**10560**	**131**	**2085**
22	83	0.57	6.72	1.56	51.72	83	121	**84**	**2019**
23	116	0.85	1.75	2.33	4.43	116	6	116	14
24	92	1.25	1.35	1.68	3.13	92	6	92	14
25	114	x		x		**129**	**8410**	**130**	**1878**
26	108	1.54	2.93	2.51	4.33	108	17	108	16
27	81	0.80	2.96	1.75	4.25	81	12	81	14
28	80	1.29	1.44	1.74	2.59	80	11	80	13
29	141	x		x		**152**	**19570s**	**157**	**1844**
30	102	2.16	15.2	2.57	13.72	102	68	102	218
31	106	2.16	2.84	2.65	4.49	106	11	106	13
32	104	2.37	2.87	2.55	4.89	104	12	104	14
33	112	0.56	19.89	1.61	87.08	112	6	112	14
34	83	0.37	0.86	1.17	3.36	83	6	83	14
35	98	0.44	2.15	1.26	2.96	98	6	98	15
36	79	0.69	0.74	1.02	1.83	79	5	79	13
37	123	x		x		**124**	**21770**	**128**	**1971**
38	78	0.57	7.94	1.54	6.10	78	52	78	179
39	102	0.91	4.21	2.13	4.40	102	6	102	14
40	91	1.18	1.38	1.91	2.27	91	6	91	14
41	158	x		x		**172**	**21130**	**179**	**1893**
42	102	1.78	8.51	2.63	41.06	102	6	102	39
43	92	0.96	3.62	2.16	4.70	92	11	92	14
44	110	1.85	2.11	2.71	4.21	110	5	110	14
45	136	x		x		**152**	**20540**	**156**	**1866**
46	93*	x		x		**95**	**384**	**97**	**2248**
47	90*	1.66	2.93	2.32	5.31	90	5	90	2002
48	114*	2.58	2.95	3.08	4.71	114	6	114	2331

If we compare IP and GA we can see that for instances in which IP found a solution, GA also found the optimal solution with execution time that is approximately the same as the execution time of IP. For some instances, IP (especially for instances with 90 activities) found solution in less time, but for problems where IP did not found a solution, GA always managed to obtain a solution of an acceptable quality. One can conclude that when dealing with problems with less activities, it would be better to use an IP solver because then one can be sure the that optimal solution is achieved. But when dealing with problems with more activities, GA shows its strength because in a reasonable amount of time one can get a good enough solution for most applications.

VII. CONCLUSION

In this paper two different formulations of IP and GA using two different representations for RCPSP were compared. The results indicate that RCPSP belongs to NP hard problems for which the exact methods are applicable only on smaller size problems and even for that problems there is no guarantee that solution will be found in reasonable time. On the other hand GA shows better results than IP, but for some instances the solving process lasts long considering the fact that there was only 30 or 90 activities and real problems are usual larger than that. Also the downside of both approaches is the fact that these methods are generally not applicable in dynamic environments.

Due to these facts, as future research it is planned to try out other heuristic methods that could give better results,

TABLE III
GENETIC ALGORITHM PERFORMANCE COMPARISON

	Permutation GA			Floating point GA		
Instance#	Best (min)	Median	Worst (max)	Best (min)	Median	Worst (max)
5	110	111.5	113	112	115	118
9	123	124	126	127	130.5	135
13	121	123	125	125	128	131
21	127	130	131	132	135	138
22	83	83.5	85	84	85	86
25	129	131	133	132	136	140
29	152	156	158	159	161	164
37	124	127	129	128	133	137
41	172	175	178	179	183.5	189
45	152	156	158	156	160	164
46	95	96	97	97	100.5	103

especially hybrid optimization algorithms in continuous domain, which could be applied to the floating point encoding. Also it is planned to find some methods that will be applicable to scheduling in dynamic conditions.

REFERENCES

[1] J. Blazewicz, J. K. Lenstra, A. R. Kan, Scheduling subject to resource constraints: classification and complexity, Discrete Applied Mathematics 5 (1) (1983) 11–24.

[2] K. S. Hindi, H. Yang, K. Fleszar, An evolutionary algorithm for resource-constrained project scheduling, Evolutionary Computation, IEEE Transactions on 6 (5) (2002) 512–518.

[3] M. Abdolshah, A review of resource-constrained project scheduling problems (rcpsp) approaches and solutions, International Transaction Journal of Engineering, Management, Applied Sciences and Technologies.

[4] P. Brucker, A. Schoo, O. Thiele, A branch and bound algorithm for the resource constrained project scheduling problem, European Journal of Operation Research 17 (1998) 143–158.

[5] O. Icmeli, W. O. Rom, Solving the resource-constrained project scheduling problem with optimization subroutine library, Computers and Operation Research 23 (1996) 801–817.

[6] A. A. B. Pritsker, L. Watters, P. Wolfe, Multiproject scheduling with limited resources: A zero-one programming approach, Management Science 16 (1969) 93–108.

[7] J. Patterson, F. Talbot, R. Slowinski, J. Weglarz, Computational experience with a backtracking alogrithm for solving a general class of precedence and resource-constrained project scheduling problems, European Journal of Operational Research 49 (1990) 68–79.

[8] L. Schrage, Solving resource-constrained network problems by implicit enumeration - nonpreemptive case, Operations Research 18 (1970) 263–278.

[9] E. Demeulemeester, W. Herroelen, A branch-and-bound procedure for the multiple resource-constrained project scheduling problem, Management Science 38 (1992) 1083–1818.

[10] J. P. Stinson, E. W. Davis, B. M. Khumawala, Multiple resource-constrained scheduling using branch and bound, AIIE Transactions 10 (1978) 252–259.

[11] R. Klein, Bidirectional planning: improving priority rule-based heuristics for scheduling resource-constrained projects, European Journal of Operational Research 127 (3) (2000) 619–638.

[12] S. Kadam, N. Kadam, Solving resource-constrained project scheduling problem by genetic algorithms, Business and Information Management (ICBIM) (2014) 159–164.

[13] H. Ouerfelli, A. Dammak, The genetic algorithm with two point crossover to solve the resource-constrained project scheduling problems, Modeling, Simulation and Applied Optimization (ICMSAO) (2013) 1–4.

[14] V. Valls, F. Ballestn, Population-based approach to the resource-constrained project scheduling problem, Annals of Operations Research 131 (2004) 305–324.

[15] D. Merkle, M. Middendorf, H. Schmeck., Ant colony optimization for resource-constrained project scheduling, IEEE Transactions on Evolutionary Computation 6 (2002) 333346.

[16] S. Kadam, S. Mane, Genetic-local search algorithm approach for resource constrained project scheduling problem, Computing Communication Control and Automation (ICCUBEA) (2015) 841–846.

[17] C. Artigues, S. Demassey, E. Neron, Resource-constrained project scheduling: models, algorithms, extensions and applications, John Wiley & Sons, 2013.

[18] K. Sastry, D. Goldberg, G. Kendall, Search methodologies, Springer, 2014, Ch. Genetic algorithms.

[19] M. Čupić, Prirodom inspirirani optimizacijski algoritmi. Meta-heuristike., Fakultet elektrotehnike i računarstva, Zagreb, 2013.

[20] I. Gurobi Optimization, Gurobi Optimizer Reference Manual., `http://www.gurobi.com/documentation/`.

[21] S. Hartmann, A competitive genetic algorithm for resource-constrained project scheduling, Manuskripte aus den Instituten fr Betriebswirtschaftslehre der Universitt Kiel 451.

[22] R. Kolisch, S. Hartmann, Handbook on recent advances in project scheduling, Christian-Albrechts-Universitt, Kiel, 1999.

[23] R. Kolisch, C. Schwindt, A. Sprecher, Benchmark instances for project scheduling problems, in: Project Scheduling, Springer, 1999, pp. 197–212.