Annales UMCS Informatica AI X, 1 (2010) 117-130
DOI: 10.2478/v10065-010-0042-8

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

A genetic algorithm for the project scheduling with the resource constraints

Marcin Klimek*

The Institute of Computer Science, State School of Higher Vocational Education, Sidorska 102, 21-500 Biala Podlaska, Poland.

Abstract – The resource-constrained project scheduling problem (RCPSP) has received the attention of many researchers because it can be applied in a wide variety of real production and construction projects. This paper presents a genetic algorithm (GA) solving the RCPSP with the objective function of minimizing makespan. Standard genetic algorithm has to be adapted for project scheduling with precedence constraints. Therefore, an initial population was generated by a random procedure which produces feasible solutions (permutation of jobs fulfilling precedence constraints). Besides, all implemented genetic operators have taken sequential relationships in a project into consideration. Finally, we have demonstrated the performance and accuracy of the proposed algorithm. Computational experiments were performed using a set of 960 standard problem instances from Project Scheduling Problem LIBrary (PSPLIB) presented by Kolisch and Sprecher [1]. We used 480 problems consisting of 30 jobs and 480 90-activity instances. We have tested effectiveness of various combinations of parameters, genetic operators to find the best configuration of GA. The computational results validate the good effectiveness of our genetic algorithm.

1 Introduction

Project scheduling is more and more often applied in production planning. Applications can be found in diverse industries such as: research-and-development (R&D), public infrastructure, construction engineering, software development etc.

*marcin_kli@interia.pl

Also, project scheduling is very important for the companies which steer production on client request so-called Make-To-Order (MTO) production. MTO is used for inventory products for the individual recipient. Every such a production order should be treated as a project developing in consultation with the customer.

Project scheduling is one of the most intractable domains for researchers as the theoretical models in this area are useful in practice and are not easy to solve. It has been shown by Błażewicz et al. [2] that the considered resource-constrained project scheduling problem, as the generalization of the job shop problem, is strongly NP-hard. Therefore, exact solution procedures to solve RCPSP can be used only for small problem instances. For large projects it is justified to use heuristic algorithms, in particular metaheuristics e.g. genetic algorithm, simulated annealing (SA), tabu search (TS) etc. Metaheuristics are effective for many optimization problems, because of sampling promising areas from the space of possible solutions.

In this work, effectiveness of applying one of metaheuristic genetic algorithms for the RCPSP will be tested.

2 Problem description

Project scheduling, as a part of project management, is aimed at deciding the time to start and/or finish jobs (activities, tasks). All activities in a project have to be performed in accordance with a set of precedence and resource constraints, with the fulfilment of properly defined optimisation criteria. The considered resource-constrained project scheduling problem with the objective function of minimizing time completion of all jobs in project (makespan) can be formulated as follows [3]:

$$minimize(s_{n+1}) \tag{1}$$

subject to:

$$s_i + d_i \leq s_j, \qquad \forall (i,j) \in E, \tag{2}$$

$$\sum_{\forall i \in A(t))} r_{ik} \leq a_k, \qquad \forall t, \forall k, \tag{3}$$

where s_i – the planned starting time of the activity i (decision variable), d_i – the non-preemptable duration of the activity i, a_k – quantity of available renewable resources of the type k ($k = 1, 2, \ldots K$, where K is the number of resource types) at any point in time, r_{ik} – the activity i requirement for the type k resource, $A(t)$ – the set of activities being processed (in progress) at time period t.

The problem deals with finding a schedule, taking into account the precedence and resource costraints to minimize makespan. A schedule is represented by the vector $S = (s_0, s_1, \ldots, s_{n+1})$ of the starting times of each activity s_i (decision variables). The

objective function (1) minimizes the start time of dummy project end activity n+1, which is equivalent to the considered objective of minimizing makespan of the project.

Constraints (2) enforce the precedence relations between jobs. The finish-start and zero-lag precedence relationships occur between the activities: the subsequent operation may start immediately after the completion of the previous one (sequential constraints).

The renewable resource of type k has a constant availability a_k. Resource constraints (3) are described as follows: at each moment of time t, the resource consumption does not exceed the available quantities a_k for every type of renewable resources $k = 1, 2, \ldots, K$.

The project schedules in the activity networks will be represented by a non-cyclical, coherent and simple directed graph $G(V, E)$, in which V means a set of nodes corresponding to the activities and E is a set of arcs which describe the sequential dependences between the activities. The set V is composed of $n + 2$ activities, numbered from 0 to $n + 1$, in a topological order, i.e. the predecessor has always a lower number than the successor. Two activities 0 and $n + 1$ are dummy: have no duration ($d_0 = d_{n+1} = 0$) and require 0 unit of resources ($r_{0,k} = r_{n+1,k} = 0$ for all $k = 1, 2, \ldots, K$). Activity 0 and $n+1$ represent the "project start" and "project end", respectively.

The project network is a graphical representation of the precedence relationships between activities. We use Activity-on-Node (AoN) representation, which is more often used than Activity-on-Arc (AoA) notation scheme for time optimisation problems. In AoN the set V represents activities and the set E denotes relationships between jobs.

The examplary project network with one renewable resource (K = 1) type is presented in Fig. 1.

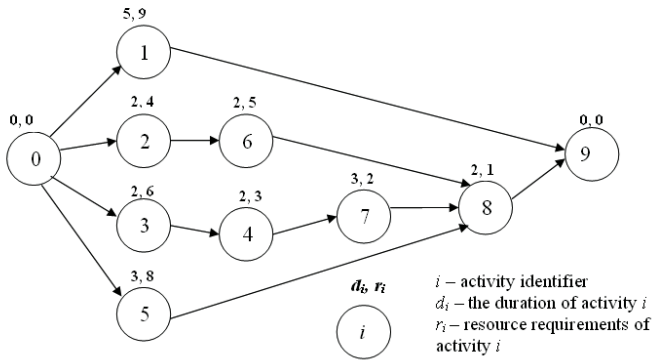

Fig. 1. Examplary project network in the activity-on-node representation.

The examplary project consists of 10 jobs (0 and 9 are dummy) which have to be a scheduled subject to one renewable resource type with the capacity a of 10 units. The set E consists of the following arcs (0, 1), (0, 2), (0, 3), (0, 5), (1, 9), (2, 6), (3, 4), (4,

7), (5, 8), (6, 8), (7, 8), (8, 9) which illustrate all immediate precedence relationships between activities.

The analysis of all jobs durations and their demand for resources points that a minimum makespan of the project is 12 time-periods:

$$\left\lceil \frac{\sum_{i=1}^{n} d_i r_i}{a} \right\rceil = \left\lceil \frac{45+8+12+6+24+10+6+2}{10} \right\rceil = \left\lceil \frac{112}{10} \right\rceil = 12 \qquad (4)$$

A feasible (satisfying all the resources and precedence constraints) schedule with an optimal makespan of 12 is presented in Fig. 2.

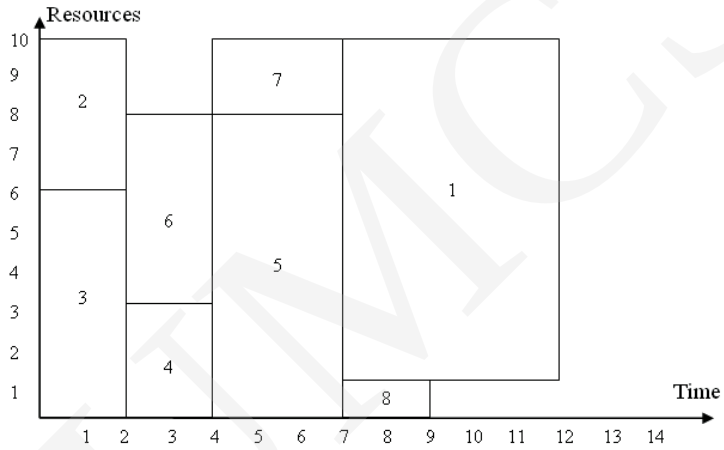

Fig. 2. A minimum duration schedule (Gantt chart) for a sample project presented in Fig. 1.

In the case of the RCPSP in order to find schedule (starting times of all activities), the decoding procedures, so-called Schedule Generation Schemes (SGSs), are used. SGS generates the sequence, based on the activity list or the priority list, taking into account the availability of the resources and the precedence relationships. SGS starts from an empty set of sequenced jobs and constructs a schedule by stepwise extension of a partial schedule. For the RCPSP two decoding procedures are used (introduced by Kelley [4]): serial SGS and parallel SGS.

Serial SGS performs activity-incrementation [5]. It consists of n steps. For one step, one job (the first non-sequenced and eligible job from the activity list or the priority list) is selected and scheduled at the earliest possible commencement time upon fulfilment of the sequential and resource constraints. Activities are so-called eligible if they can be started in actual step because all of their predecessors have been scheduled.

Parallel SGS performs time-incrementation [5]. Iteratively at subsequent moments of the time t (at the decision-making points), all the unscheduled and eligible activities (considered in the sequence arranged on the activity list or the priority list) are

started, the ones which may be started upon fulfilment of the precedence and resource constraints.

Decoding procedures are the core of heuristics for the RCPSP. When we use SGS, we can apply algorithms of searching solutions using the permutation coding (e.g. metaheuristics GA, TS, SA), popular representation of many optimization problems.

3 Genetic algorithm

Genetic algorithm was first specified in 1975 (by Holland [6]). GAs are a kind of stochastic, multi-point, parallel search algorithms applied to a lot of optimization problems. Genetic algorithm is population based technique which is inspired by the biological evolution. The techniques of searching potential solutions mimic natural genetic inheriting and phenomenon of natural selection. Mechanisms of natural selection are applied as follows: only the strongest individuals survive and take part in a reproduction (crossover, mutation). Solutions are represented by chromosomes which are evaluated with the fitness function – the degree of adaptation of individuals in the environment.

In GA the strongest individuals pass the genetic information over to their descendants (in crossover and mutation operations). Next generations are better and better adapted for conditions of the environment. GA simultaneously considers a population of solutions instead of only one in contrast to the other popular metaheuristics like simulated annealing or tabu search. Genetic algorithm is easy to implement metaheuristics which produces the results whose quality is uncertain, but GA can be designed to execute in a given amount of time.

Many variants of basic idea GA in research works and applications may be largely different. A pseudo-code for our genetic algorithm is given below in Listing. 3.

Listing 1. Pseudo-code of genetic algorithm [7].

```
procedure GA;
begin
t := 0;
initialize;
generate P(0); {generating initial population}
evaluate P(0); {fitness function evaluation}
        while (not stop_condition) do
        begin
        t := t + 1;
     select P(t) of P(t-1); {selection of individuals to the next
         generation}
        change P(t); {crossover, mutation}
        evaluate P(t); {fitness function evaluation}
        end
end;
```

Every application of genetic algorithm should include the following elements which can distinguish variants of GA [7]:

- coding and a method for decoding solutions to the problem,
- way of generating of initial population,
- evaluation (fitness) function which measures the quality of the solution,
- methods of selecting individuals for the next generation,
- crossover operators,
- mutation operators.

Below we describe the key aspects of our genetic algorithm.

3.1 Coding

The chromosome representation of the RCPSP is an activity list $< j_1, j_2, \ldots, j_n >$ – permutation of non dummy jobs [8]. The activity list consists of numerals "1" to "n" where each numeral corresponds to a job in a project. This list is precedence feasible – it means that each activity must have a higher index in the activity list than each of its predecessors in the project network.

Decoding the activity list (solution of GA) to a schedule (solution of the RCPSP) is realized by parallel or serial SGS.

3.2 Initial population

The initial chromosomes will be generated by a procedure which creates precedence feasible solutions. It will be reached at random using the serial SGS decoding procedure for the random permutation list of jobs.

3.3 Evaluation – fitness function

Fitness function is used for evaluation of chromosomes. These evaluations are important for selection methods.

Fitness function is a particular type of objective function that measures the quality of an individual (chromosome) in a genetic algorithm. For the RCPSP the fitness function before scaling is equal to makespan of a project.

If we use the tournament selection operator, the only important thing is which one of the individuals is better or worse. The level of difference between chromosomes is important in the roulette wheel selection. Our conception of fitness scaling for the minimization criterion make use of a slight difference from the worst individual:

$$fitness_i = max_{j=1\ldots N}\{f_j\} - f_i + \gamma, \tag{5}$$

where f_i – fitness of individual i before scaling (equal to makespan of a project), $fitness_i$ – fitness of individual i after scaling, γ – constant parameter for all individuals,

enables "surviving" of the worst individual, in experiments we set $\gamma = 0.1$, N – number of individuals in population.

3.4 Methods of selection

Selection is a genetic operator that chooses individuals from the current population for inclusion in the next generation. It is an element of adaptive plan, purpose of which is to produce improved population of solutions from the current one. In this work, we implement the following selection operators:

- roulette wheel selection (fitness proportionate selection) – developed by Holland (1975), the method is constructed in such a way that selection probability for each individual is proportional to the fitness value,
- tournament selection – a few chromosomes, chosen at random from the actual population, take part in "tournament", one of them, with the best fitness, wins and goes to the next generation; the number of tournaments should be equal to the size of population (adding elite solutions).

In selection we use elitism, which ensures that the best found solutions are preserved and passed on to the next population.

3.5 Crossover operators

Crossover is a genetic operator that combines two parent individuals to produce a new child individual (offspring). In our work two parent chromosomes are replaced by children chromosomes The idea behind crossover is that new individuals may be better than both of the parents if they take the best characteristics from each of the parents. The parents taking part in crossover are established at random according to a user-definable crossover probability.

We consider various genetic operators fulfilling the requirement of permutation problems that each job (1 to n) should appear once in the generated children chromosomes. The following crossover operators will be used [9]:

- 1PX (One-Point order Crossover),
- 2PX (Two-Point order Crossover),
- PPX (Precedence Preservance Crossover).

1PX, 2PX, PPX are used in GA for many optimization problems with the permutation coding. These crossover operators are used for the RCPSP because they generate children chromosomes which satisfy all precedence constraints (of course if parents respect the precedence relationships).

In 1PX, one crossover point (cut-point) is randomly selected for dividing parents. The set of genes, on the left side of this crossover point, is copied from the parent to the offspring (from parent 1 to child 1 and from parent 2 to child 2), and all the remaining jobs, on right the side of the cut-point, are placed in the order of their appearance in the other parent.

In 2PX, two cut-points are randomly selected for dividing parents. The genes outside the selected two cut-points are inherited (from parent 1 to child 1 and from parent 2 to child 2), and the other genes (the mid part of the chromosome) are placed in the order of their appearance in the other parent.

PPX was developed by Bierwirth et al. [10], specially for scheduling problems. In PPX at the beginning there is created a mask which consists of n-elements random 0 or 1, indicating which parent genes should be taken from. Child 1 is created by copying the available gene based on the next mask values: if the actual mask value equals 1, gene is copied from parent 1, otherwise from parent 2. Gene copied to the child is removed from both parents. This procedure is repeated using a new mask for child 2.

3.6 Mutation operators

Mutation is a genetic operator that changes one or more gene values in an individual. Using this operator prevents the population from stagnating at any local optima. Chromosomes to mutation are established at random according to a user-definable mutation probability.

The traditional mutation operators are not well suited for the RCPSP and will be modified because all precedence relationships in the project must be satisfied [9]. Modifications of mutations for the resource-constrained project scheduling problem are presented in Table 1.

Table 1. Mutation operators for the RCPSP.

Mutation	Standard procedure	Modified procedure for the RCPSP
Invert	Reverse the order of the elements between randomly selected two positions.	**Step 1**: Select a gene g at random. **Step 2**: Find a set of genes in chromosome from the left side of the gene g which can be exchanged with the gene g with satisfying precedence relationships. **Step 3**: Exchange the gene g with randomly selected gene from the set of genes determined in Step 2.
Swap	Randomly selected two jobs are exchanged.	**Step 1**: Select a gene g at random. **Step 2**: Find a set of genes in chromosome from the left and right sides of the gene g which can be exchanged with the gene g respecting the precedence relationships. **Step 3**: Exchange the gene g with a randomly selected gene from the set of genes determined in Step 2.

Table 1. Continued.

Swap adjacent	Two adjacent randomly selected genes are exchanged.	**Step 1**: Select a gene g at random. **Step 2**: Check if the swap mutation of adjacent gene from the left side of the gene g with the gene g can be realized with respect to sequential constraints, exchange these genes, otherwise go to Step 2. **Step 3**: Check if the swap mutation of adjacent gene from the right side of the gene g with the gene g can be realized with satisfying precedence constraints, exchange these genes, otherwise end procedure.
Insert	A gene at one random position is removed and put at another random position with maintaining the relative order of all other genes.	**Step 1**: Select a gene g at random. **Step 2**: Find a list of all positions in chromosome where the gene g can be inserted with satisfying precedence relationships. **Step 3**: Insert the gene g at randomly selected position from the list found in Step 2 with maintaining the relative order of all other genes.

In this work, the author proposes hybridization GA with local search (LS). It will be realized by special construction of moves Insert all, Swap all and Swap adjacent all. Insert all. Swap all, Swap adjacent all are implemented similarly. At the beginning all possible moves, respecting the precedence constraints, of the given type (Insert all – all possible mutations Insert, Swap all – all possible mutations Swap, Swap adjacent all – all possible mutations Swap adjacent) are performed and evaluated. Next the best found solution replaces elite chromosome (in our approach LS is used only for actual best chromosome) if only its fitness is better than the actual best fitness.

For an example, in move Insert all a random gene g in the actual best chromosome (parent) is inserted in all possible positions with respect to the precedence constraints. After insertion each chromosome is evaluated and finally the best position for insertion the gene g is chosen. If fitness of the generated offspring is better than that of the parent, the offspring is included into the population.

4 Computational results

All algorithms were implemented in C# in Microsoft Visual Studio 2005. The tests were performed using a computer with a 1,7 GHz Intel Pentium CPU. To present the effectiveness of GA we considered 1080 instances of test problems from the two classes PSPLIB [1]: J30 (480 instances with 30 jobs) and J120 (600 instances with 120 jobs).

The experiments were performed using the following configuration of genetic algorithm for all size problems:

- population size – 50, mating pool is also 50,
- mutation rate – 0.2,
- crossover rate – 0.7,
- elite size is equal to 0 (no elitist) or 2 (two elite chromosomes),
- maximal number of generations – 100.

Stopping criterion of GA was a maximal number of all generated and evaluated schedules which equals 5000 schedules (counting schedules includes random solutions in the first generation), so hybrid algorithm with LS can stop before the 100th generation. This stopping criterion was chosen in order to enable us to compare our algorithm with other heuristics from the literature.

We decoded the activity-list (chromosome representation) into a feasible schedule with serial or parallel SGS. Better schedules were created using serial SGS (by 0.2% on the average greater values of fitness). Besides, parallel SGS took more computational time by 3.2 times on the average than serial SGS.

Each problem was solved employing two selection operators (roulette wheel, tournament), three crossover operators (1PX, 2PX, PPX) and four mutation operators (Invert, Swap adjacent, Swap, Insert). Local search in the elitist GA was performed with the three moves Swap adjacent all, Swap all, Insert all. The results of effectiveness of GA using serial SGS with different selection, crossover, mutation operators, are presented in Table 2.

Table 2. Results of experiments (serial SGS).

GA settings*	**30 jobs**				**120 jobs**			
	roulette wheel		**tournament**		**roulette wheel**		**tournament**	
	a	**b**	**a**	**b**	**c**	**d**	**c**	**d**
PPX,SW,0,NO	2.9%	275	1.9%	313	21.4%	32	19.6%	46
PPX,SW,2,NO	1.4%	319	1.3%	326	17.8%	55	17.7%	51
PPX,SW,2,ADA	2.4%	288	2.2%	290	24.5%	25	23.3%	31
PPX,SW,2,ISA	1.8%	307	1.6%	314	22.0%	33	21.5%	40
PPX,SW,2,SWA	1.7%	312	1.5%	313	20.5%	46	19.6%	43
PPX,AD,0,NO	2.4%	289	2.0%	304	20.5%	31	19.2%	35
PPX,AD,2,NO	1.6%	312	1.8%	299	19.0%	47	18.8%	51
PPX,AD,2,ADA	2.7%	278	2.2%	301	24.6%	26	23.5%	24
PPX,AD,2,SWA	1.9%	308	1.7%	309	20.6%	39	19.8%	45
PPX,IS,0,NO	4.2%	244	2.6%	306	22.6%	28	20.2%	62
PPX,IS,2,NO	1.3%	335	1.2%	335	17.3%	50	17.2%	61
PPX,IS,2,ADA	2.4%	286	2.1%	292	24.6%	24	23.3%	28
PPX,IS,2,ISA	1.9%	302	1.5%	323	22.0%	30	21.3%	35

Table 2. Continued.

PPX,IS,2,SWA	1.7%	316	1.5%	323	20.4%	43	19.7%	46
PPX,IN,0,NO	3.0%	291	2.0%	307	21.7%	38	19.2%	41
PPX,IN,2,NO	1.3%	332	1.3%	329	17.8%	52	17.7%	57
PPX,IN,2,ADA	2.4%	284	2.2%	296	24.6%	26	23.4%	27
PPX,IN,2,ISA	1.8%	312	1.6%	322	22.0%	35	21.2%	36
PPX,IN,2,SWA	1.8%	309	1.6%	316	20.5%	44	19.7%	45
PPX,IN,2,SWA	2.1%	303	1.7%	339	18.5%	59	17.4%	62
1PX,SW,2,NO	1.3%	328	1.2%	330	17.6%	59	17.8%	55
1PX,SW,2,ADA	2.5%	298	2.1%	298	24.3%	25	23.5%	32
1PX,SW,2,ISA	1.6%	314	1.4%	326	22.1%	39	21.3%	38
1PX,SW,2,SWA	1.7%	309	1.5%	321	20.0%	49	19.4%	48
1PX,AD,0,NO	1.9%	302	1.6%	324	18.6%	45	18.2%	56
1PX,AD,2,NO	1.6%	319	1.7%	303	18.6%	47	18.9%	52
1PX,AD,2,ADA	2.6%	285	2.1%	298	24.6%	26	23.6%	27
1PX,AD,2,ISA	1.7%	313	1.6%	305	21.9%	34	21.4%	37
1PX,AD,2,SWA	1.6%	315	1.6%	317	20.1%	48	19.5%	52
1PX,IS,0,NO	2.7%	300	1.7%	336	17.8%	63	16.5%	72
1PX,IS,2,NO	1.2%	331	**1.1%**	340	17.1%	66	17.0%	63
1PX,IS,2,ADA	2.5%	291	2.0%	301	24.6%	28	23.4%	22
1PX,IS,2,ISA	1.5%	317	1.4%	324	22.0%	32	21.3%	35
1PX,IS,2,SWA	1.5%	317	1.5%	323	19.9%	48	19.2%	49
1PX,IN,0,NO	1.8%	314	1.6%	334	18.0%	56	16.8%	68
1PX,IN,2,NO	1.2%	339	1.2%	342	17.5%	57	17.5%	56
1PX,IN,2,ADA	2.5%	290	2.1%	305	24.5%	27	23.5%	28
1PX,IN,2,ISA	1.7%	313	1.5%	315	22.0%	32	21.3%	38
1PX,IN,2,ISA	1.6%	309	1.5%	324	20.0%	48	19.4%	46
1PX,IN,2,ISA	2.2%	307	1.4%	347	17.9%	62	17.1%	65
2PX,SW,2,NO	1.3%	340	1.3%	325	17.2%	63	17.5%	54
2PX,SW,2,ADA	2.4%	295	2.2%	291	24.5%	21	23.5%	31
2PX,SW,2,ISA	1.7%	310	1.6%	320	22.0%	34	21.4%	33
2PX,SW,2,SWA	1.6%	314	1.4%	330	19.9%	53	19.3%	54
2PX,AD,0,NO	2.0%	305	1.5%	330	18.0%	58	17.8%	51
2PX,AD,2,NO	1.5%	323	1.6%	318	18.0%	57	18.5%	56
2PX,AD,2,ADA	2.7%	294	2.1%	297	24.5%	25	23.3%	24
2PX,AD,2,ISA	1.7%	312	1.6%	324	22.0%	35	21.4%	40
2PX,AD,2,SWA	1.7%	309	1.6%	314	19.9%	45	19.3%	52
2PX,IS,0,NO	2.5%	299	1.6%	**349**	17.0%	79	**16.0%**	**82**

Table 2. Continued.

2PX,IS,2,NO	1.1%	347	**1.1%**	**346**	16.6%	70	16.8%	70
2PX,IS,2,ADA	2.5%	293	2.2%	288	24.6%	21	23.5%	27
2PX,IS,2,ISA	1.5%	318	1.6%	320	21.8%	39	21.2%	40
2PX,IS,2,SWA	1.4%	320	1.4%	319	19.7%	54	19.1%	55
2PX,IN,0,NO	2.1%	308	1.7%	338	17.5%	64	16.3%	68
2PX,IN,2,NO	1.2%	337	**1.1%**	1.1%	17.1%	60	17.2%	62
2PX,IN,2,NO	2.6%	283	2.1%	296	24.6%	28	23.5%	23
2PX,IN,2,ISA	1.7%	321	1.6%	324	21.9%	35	21.4%	39
2PX,IN,2,SWA	1.5%	322	1.4%	334	20.0%	51	19.3%	55

where a – number of solutions (among 480 possible) whose makespan is equal to the optimal makespan of project, b – average deviations (%) from the optimal makespan, c – number of solutions (among 480 possible) whose makespan is equal to the best known makespan, d – average deviations (%) from the critical path lower bound [8], * – the description of algorithm settings has the following construction: crossover type, mutation type, elite size, local search move type.
The abbreviations used in the operator descriptions: SW – mutation Swap, AD – mutation Swap adjacent, IS – mutation Insert, IN – mutation Invert, SWA – mutation Swap all, ADA – mutation Swap Adjacent all, ISA – mutation Insert all.

The analysis of experimental results indicates that:

- tournament selection is more effective than roulette wheel selection (other fitness scaling can improve results) in the majority of cases,
- Swap adjacent is the worst mutation operator, Insert is the best mutation operator,
- local search in elitist GA is not effective, better results are obtained without LS,
- for 30-job problems the elitist strategy is better than selection without elite chromosomes,
- for 120-job instances the elitist strategy is worse than selection without elite chromosomes,
- 2PX is on the average the best crossover operator, 1PX is more effective than PPX,
- the best settings of GA for J30: tournament selection, crossover 2PX, mutation Insert, 2 elite chromosomes,
- the best settings of GA for J120: tournament selection, crossover 2PX, mutation Insert, without elite chromosomes.

The results obtained by our genetic algorithm are good compared to other algorithms from the literature [8, 11].

5 Conclusions

This paper presents a genetic algorithm for the Resource-Constrained Project Scheduling Problem with the objective of makespan minimization. The schedules were created using parallel or serial generation schemes with the activity list representation of chromosomes. Genetic algorithm was tested with various settings of parameters and operators on a set of problem instances from the PSPLIB.

The proposed GA with the best configuration of parameters experimentally found generates good solutions compared to other approaches. The key to success is appropriate search for solution space with adapting known genetic operators to the RCPSP. The simulation results show that the adapted operators are good for all size problems.

References

[1] Kolisch R., Sprecher A., PSPLIB – a project scheduling library, European Journal of Operational Research 96 (1997): 205.

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

[3] Herroelen W., De Reyck B., Demeulemeester E., Resource constrained scheduling: a survey of recent developments, Computers and Operations Research 25 (1998).

[4] Kelley J. E. Jr., The critical-path method: resources planning and scheduling, Muth J. F., Thompson G. L. (Industrial Scheduling, Prentice-Hall, New Jersey, 1963): 347.

[5] Kolisch R., Serial and parallel resource-constrained project scheduling methods revisited: theory and computation, European Journal of Operational Research 90 (1996): 320.

[6] Holland J.H., Adaptation in natural and artificial systems (University of Michigan Press, Ann Arbor, 1975).

[7] Michalewicz Z., Genetic algorithms + data structures = evolution programs (Springer-Verlag, 1992).

[8] Kolisch R., Hartmann S., Heuristic algorithms for solving the resource-constrained project scheduling problem: classification and computational analysis, Handbook on Recent Advances in Project Scheduling: Recent Models, Algorithms and Applications, J. Weglarz (Kluwer Academic Publishers, 1999): 147.

[9] Kostrubiec A., Metody generowania sasiedztwa w metaheurystycznych metodach harmonogramowania projektów. Inżynieria systemów zarzadzania. Ilościowe metody wspomagania decyzji w systemach produkcji (Wydawnictwo Politechniki Gdańskiej, Gdańsk, 2005): 45.

[10] Bierwirth C., Mattfeld D.C., Production scheduling and rescheduling with genetic algorithms, Evolutionary Computation 7 (1999): 1.

[11] Kolisch R., Hartmann S., Experimental investigation of heuristics for resource-constrained project scheduling: an update, European Journal of Operational Research 174 (2006): 23.