Project scheduling under multiple resources constraints using a genetic algorithm

J. MAGALHÃES-MENDES
Department of Civil Engineering
School of Engineering – Polytechnic of Porto
Rua Dr. António Bernardino de Almeida, 431 – 4200-072 Porto
PORTUGAL
jjm@isep.ipp.pt

Abstract: - The resource constrained project scheduling problem (RCPSP) is a difficult problem in combinatorial optimization for which extensive investigation has been devoted to the development of efficient algorithms. During the last couple of years many heuristic procedures have been developed for this problem, but still these procedures often fail in finding near-optimal solutions. This paper proposes a genetic algorithm for the resource constrained project scheduling problem. The chromosome representation of the problem is based on random keys. The schedule is constructed using a heuristic priority rule in which the priorities and delay times of the activities are defined by the genetic algorithm. The approach was tested on a set of standard problems taken from the literature and compared with other approaches. The computational results validate the effectiveness of the proposed algorithm.

Key-Words: - Construction management, Project management, Scheduling, Genetic algorithm, RCPSP.

1 Introduction

As the complexity of projects increases, the requirement of an organized planning and scheduling process is enhanced.

The need for organized planning and scheduling of a construction project is influenced by a variety of factors (e.g., project size and number of project activities).

To plan and schedule a construction project, activities must be defined sufficiently so that adequate communication is provided to all those who will use the information. The level of detail determines the number of activities contained within the project plan and schedule. As the number of project activities increases and thus the complexity of their sequential ordering, the need for organized planning and scheduling increases. This need further increases when a large number of project activities are considered relative to the uniqueness of each construction project in terms of the dynamic plant and nonstandardized nature of the work [25].

The analysis of resources, particularly time, materials, labor and equipment is the key to good project management.

Project scheduling allows determine the project duration and involves the allocation of the limited resources to projects to determine the start and completion times of the detailed activities.

The use of microcomputers and project scheduling computer software is commonplace in the construction industry. This is principally true for scheduling project activities and managing resources.

During the last couple of years many heuristic procedures have been developed for this problem (called RCPSP), but still these procedures often fail in finding near-optimal solutions.

The RCPSP problem belongs to the class of NP-hard optimization problems, therefore justifying the indispensable use of heuristic solution procedures when solving large problem instances.

Recent classification and survey can be found in Brucker et al. [22] and Kolisch and Hartmann [16]. The survey provided by Kolisch and Hartmann [16] presents more than eighty models and algorithms for complex scheduling problems and discusses the RCPSP.

More recent work is due to Debels et al. [3], Debels and Vanhoucke [4], Mendes et al. [6], Fleszar and Hindi [8], Palpant et al. [9], Yeh and Pan [19], Kochetov and Stolyar [11], Valls et al. [12], Valls et al. [13], Ranjbar [14], Mendes and Gonçalves [15], Seda [17], Kljajc et al. [18] and Pan and Yeh [28].

2 Problem Definition

The resource constrained project scheduling problem (RCPSP) can be stated as follows. A project consists

of $n+2$ activities where each activity has to be processed in order to complete the project. Let $J = \{0, 1, \ldots, n, n+1\}$ denote the set of activities to be scheduled and $K = \{1, ..., k\}$ the set of resources. The activities 0 and $n+1$ are dummy, have no duration and represent the initial and final activities. The activities are interrelated by two kinds of constraints:

1. The *precedence constraints*, which force each activity j to be scheduled after all predecessor activities, P_j, are completed.
2. Performing the activities requires resources with *limited capacities*.

While being processed, activity j requires $r_{j,k}$ units of resource type $k \in K$ during every time instant of its non-preemptable duration d_j. Resource type k has a limited capacity of R_k at any point in time. The parameters d_j, $r_{j,k}$ and R_k are assumed to be non-negative and deterministic. For the project start and end activities we have $d_0 = d_{n+1} = 0$ and $r_{0,k} = r_{n+1,k} = 0$ for all $k \in K$.

The problem consists in finding a schedule of the activities, taking into account the resources and the precedence constraints, that minimizes the *makespan* (C_{max}).

Let F_j represent the finish time of activity j. A schedule can be represented by a vector of finish times $(F_1, \ldots, F_m, \ldots, F_{n+1})$. The *makespan* of the solution is given by the maximum of all predecessors activities of activity $n+1$, i.e. $F_{n+1} = Max_{l \in P_{n+1}} \{F_l\}$.

The conceptual model of the RCPSP was described by Christofides et al. [23] in the following way:

$$\textbf{Min } F_{n+1} \tag{1}$$

subject to:

$$F_l \le F_j - d_j \qquad j = 1, \ldots, N+1 \; ; \; l \in P_j \tag{2}$$

$$\sum_{j \in A(t)} r_{j,k} \le R_k \qquad k \in K \; ; \; t \ge 0 \tag{3}$$

$$F_j \ge 0 \qquad j = 1, \ldots, N+1 \tag{4}$$

The objective function (1) minimizes the finish time of activity $n+1$, and therefore minimizes the makespan. Constraints (2) impose the precedence relations between activities and constraints (3) limit the resource demand imposed by the activities being processed at time t to the capacity available. Finally (4) forces the finish times to be non-negative.

Fig. 1 shows an example (AON – activity-on-node) of a project with $n = 6$ activities, subject to two renewable resources types with a capacity of four and two units, respectively. A feasible schedule with a makespan of 14 time-periods is represented in Fig. 7.

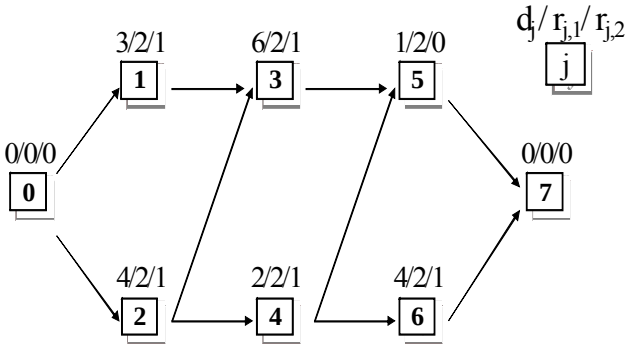

Fig.1 – AON diagram – example project.

3 Types of schedules

Classifying schedules is the basic work to be done before attacking scheduling problems [21].

Schedules can be classified into one of the following three types of schedules:

i) *Feasible* schedules. A schedule is said to be feasible if it is non-preemptive and if the precedence and resource constraints are satisfied.

ii) *Semi-active schedules*. These are feasible schedules obtained by sequencing activities as early as possible. In a semi-active schedule the start time of a particular activity is constrained by the processing of a different activity on the same resource or by the processing of the directly preceding activity on a different resource.

iii) *Active schedules*. These are feasible schedules in which no activity could be started earlier without delaying some other activity or breaking a precedence constraint. Active schedules are also semi-active schedules. An optimal schedule is always active.

iv) *Non-delay schedules*. These are feasible schedules in which no resource is kept idle at a time when it could begin processing some activity. Non-delay schedules are active and hence are also semi-active.

The set of active schedules is usually very large and contains many schedules with poor quality. To reduce the solution space we use the concept of parameterized active schedules.

The concept of parameterized active schedules is proposed in Gonçalves and Beirão [27], Mendes [20], Gonçalves et al. [5] and Mendes et al. [6]. This type of schedule consists of schedules in which no resource is kept idle for more than a predefined

period if it could start processing some activity. If the predefined period is set to zero, then we obtain a non-delay schedule.

4 Application of genetic algorithm

The approach presented in this paper is based on a genetic algorithm to perform its optimization process.

Genetic algorithms (GAs) are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search [1].

One fundamental advantaged of GAs from traditional methods is described by Goldberg [1]: in many optimization methods, we move gingerly from a single solution in the decision space to the next using some transition rule to determine the next solution. This solution-to-solution method is dangerous because it is a perfect prescription for locating false peaks in multimodal search spaces. By contrast, GAs work from a rich database of solutions simultaneously (a population of chromosomes), climbing many peaks in parallel; thus the probability of finding a false peak is reduced over methods that go solution to solution.

The general schema of GAs may be illustrated as follows (Fig. 2).

```
procedure GENETIC-ALGORITHM

Generate initial population P0;
 Evaluate population P0;
 Initialize generation counter g ←0;

 While stopping criteria not satisfied repeat
    Select some elements from Pg to copy into Pg+1;
    Crossover some elements of Pg and put into Pg+1;
    Mutate some elements of Pg and put into Pg+1;
    Evaluate some elements of Pg and put into Pg+1;
    Increment generation counter: g ← g+1;
  End while

End GENETIC-ALGORITHM;
```

Fig. 2 - Pseudo-code of a genetic algorithm.

First of all, an initial population of potential solutions (individuals) is generated randomly. A selection procedure based on a fitness function enables to choose the individuals candidate for reproduction. The reproduction consists in recombining two individuals by the crossover operator, possibly followed by a mutation of the offspring. Therefore, from the initial population a new generation is obtained. From this new generation, a second new generation is produced by the same process and so on. The stop criterion is normally based on the number of generations.

4.1 Decoding

The genetic algorithm uses a random key alphabet which is comprised of real random numbers between 0 and 1.

A chromosome represents a solution to the problem and is encoded as a vector of random keys (random numbers). Each solution chromosome is made of 2*n* genes where *n* is the number of activities:

$$Chromosome = (gene_1, .., gene_n, gene_{n+1}, ..., gene_{2n})$$

4.1.1 Decoding the priorities of activities

The priority decoding expression used the following expression

$$\boldsymbol{PRIORITY_j} = \frac{\boldsymbol{LLP_j}}{\boldsymbol{LCP}} \times \left[\frac{1+\boldsymbol{gene_j}}{2}\right] \qquad \boldsymbol{j} = 1,...,\boldsymbol{n}$$

where $\boldsymbol{LLP_j}$ is the longest length path from the beginning of the activity *j* to the end of the project and ***LCP*** is the length along the critical path of the project.

An example is presented in Tables 1, 2 and 3.

Table 1 shows how to obtain the ***LCP*** for the example project presented in Fig. 1 (***LCP*** = 11).

Path	*Time duration*
0-1-3-5-7	3+6+1 = 10
0-2-3-5-7	4+6+1 = 11
0-2-4-5-7	4+2+1 = 7
0-2-4-6-7	4+2+4 = 10

Table 1: Time duration of the critical path (***LCP***) without limited capacities.

Table 2 shows how to obtain the $\boldsymbol{LLP_j}$ values for each activity *j*.

$Activity_j$	LLP_j	$PRIORITY_j$
1	10	$\frac{10}{11}\times\left[\frac{1+gene_j}{2}\right]$
2	$Max\{11, 10\}=11$	$\frac{11}{11}\times\left[\frac{1+gene_j}{2}\right]$
3	7	$\frac{7}{11}\times\left[\frac{1+gene_j}{2}\right]$
4	$Max\{3, 6\}=6$	$\frac{6}{11}\times\left[\frac{1+gene_j}{2}\right]$
5	1	$\frac{1}{11}\times\left[\frac{1+gene_j}{2}\right]$
6	4	$\frac{4}{11}\times\left[\frac{1+gene_j}{2}\right]$

Table 2: Values of the LLP_j for each activity *j*.

For the following chromosome:

Chromosome = (0.12 ,0.54, 0.76, 0.23, 0.54, 0.44, 0.12, 0.87, 0.34, 0.27, 0.92, 0.55)

we obtain the $PRIORITY_j$ values presented in Table 3.

$Activity_j$	$gene_j$	$PRIORITY_j$
1	0.12	0.51
2	0.54	0.77
3	0.76	0.56
4	0.23	0.34
5	0.54	0.07
6	0.44	0.26

Table 3: Values of the $PRIORITY_j$ for each activity *j*.

4.1.2 Decoding the delay times

The genes between n+1 and 2n are used to determine the delay times used when scheduling an activity. The delay time used by each scheduling iteration g, $Delay_g$, is given by the following expression:

$$Delay_g = gene_{n+g} \times 1.5 \times MaxDur$$

where MaxDur is the maximum duration of all activities. The factor 1.5 was obtained after some experimental tuning.

4.2 Evolutionary strategy

To breed good solutions, the random key vector population is operated upon by a genetic algorithm.

There are many variations of genetic algorithms obtained by altering the reproduction, crossover, and mutation operators.

Reproduction is a process in which individual (chromosome) is copied according to their fitness values (*makespan*).

Reproduction is accomplished by first copying some of the best individuals from one generation to the next, in what is called an elitist strategy.

In this paper the fitness proportionate selection, also known as roulette-wheel selection, is the genetic operator for selecting potentially useful solutions for reproduction. The characteristic of the roulette wheel selection is stochastic sampling.

The fitness value is used to associate a probability of selection with each individual chromosome. If f_i is the fitness of individual i in the population, its probability of being selected is,

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i} \quad , \qquad i = 1,\ldots,n \qquad (5)$$

An example is presented in Table 4.

A roulette wheel model is established to represent the survival probabilities for all the individuals in the population. Then the roulette wheel is rotated for several times [1], see Fig. 3.

After selection the mating population consists of the chromosomes (individuals): 1, 2, 3, 4, 5 and 6.

Number of chromosome	Fitness value	Selection probability
1	14	0.20
2	12	0.17
3	10	0.14
4	9	0.13
5	8	0.11
6	7	0.10
7	4	0.06
8	3	0.04
9	2	0.03
10	1	0.01

Table 4: Selection probability and fitness value.

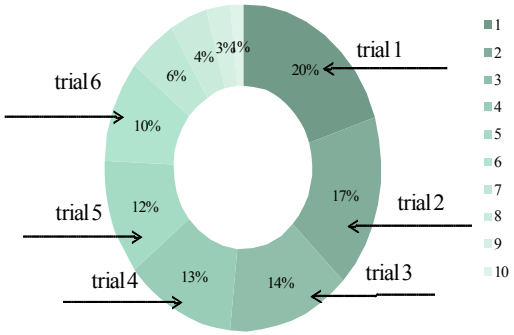

Fig. 3 - Roulette-wheel selection.

After selecting, crossover may proceed in two steps. First, members of the newly selected (reproduced) chromosomes in the mating pool are mated at random. Second, each pair of chromosomes undergoes crossover as follows: an integer position *k* along the chromosome is selected uniformly at random between 1 and the chromosome length *l*. Two new chromosomes are created swapping all the genes between *k*+1 and *l* [1], see Fig. 4.

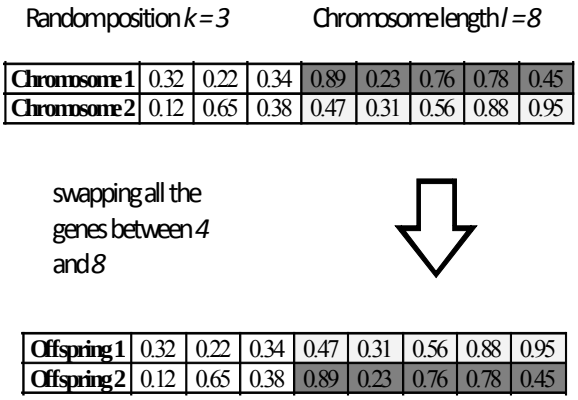

Fig.4 – Crossover operator example.

The mutation operator preserves diversification in the search. This operator is applied to each offspring in the population with a predetermined probability. We assume that the probability of the mutation in this paper is 0.001. With 60 genes positions we should expect 60 x 0.001 = 0.06 genes to undergo mutation for this probability value.

5 Schedule Generation Schemes

Schedule generation schemes (SGS) are the core of most heuristic solution procedures for the RCPSP. SGS start from scratch and build a feasible schedule by stepwise extension of a partial schedule. A partial schedule is a schedule where only a subset of the *n*+2 activities have been scheduled. There are two different classics methods SGS available. They can be distinguished into activity and time incrementation. The so called serial SGS performs activity-incrementation and the so called parallel SGS performs time-incrementation [26].

The constructive heuristic used to construct active schedules is based on a scheduling generation scheme that does time incrementing, called *parallel modified*.

This heuristic makes use of the priorities and the delay times defined by the genetic algorithm and constructs parameterized active schedules. The concept of parameterized active schedules is proposed in Gonçalves and Beirão [27], Mendes [20], Gonçalves et al. [5] and Mendes et al. [6].

Figure 5 illustrates where the set of parameterized active schedules is located relative to the class of semi-active, active, and non-delay schedules.

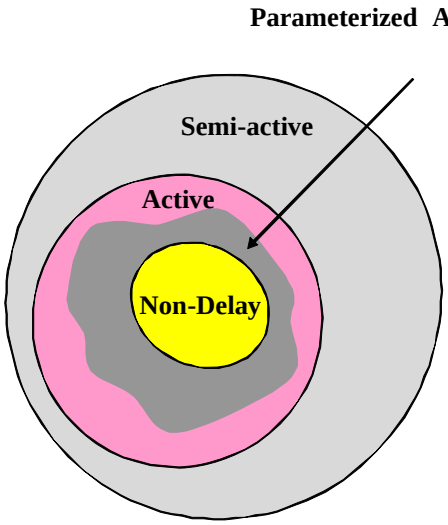

Fig. 5 – Parameterized active schedules.

The heuristic used to construct parameterized active schedules is based on a scheduling generation scheme that does time incrementing. For each iteration $\boldsymbol{g}$, there is a scheduling time $\boldsymbol{t_g}$. The active set comprises all activities which are active at $\boldsymbol{t_g}$, i.e.

$$\boldsymbol{A_g} = \left\{ \boldsymbol{j} \in \boldsymbol{J} \,|\, \boldsymbol{F_j} - \boldsymbol{d_j} \leq \boldsymbol{t_g} < \boldsymbol{F_j} \right\}.$$

The remaining resource capacity of resource *k* at instant time $\boldsymbol{t_g}$ is given by

$$RD_k(t_g) = R_k(t_g) - \sum_{j \in A_g} r_{j,k} \quad .$$

Sg comprises all activities which have been scheduled up to iteration g, and F_g comprises the finish times of the activities in S_g. Let $Delay_g$ be the delay time associated with iteration g, and let E_g comprise all activities which are precedence feasible in the interval $[t_g, t_g + Delay_g]$, i.e.

$$E_g = \left\{ j \in J \setminus S_{g-1} \mid F_i \le t_g + Delay_g \ \ (i \in P_j) \right\} .$$

The algorithmic description of the scheduling generation scheme used to create parameterized active schedules is given by the pseudo-code shown in Figure 6.

Initialization: $g=1, t_1=0, A_0=\{0\}, \Gamma_o=\{0\}, S_0=\{0\}, RD_k(0)=R_k \ (k \in K)$

while $|S_g| < n+2$ **repeat**
{
  Update E_g

  while $E_g \neq \{\}$ **repeat**
  {
    Select activity with highest priority

    $j^* = \underset{j \in E_g}{\operatorname{argmax}} \left\{ PRIORITY_j \right\}$

    Calculate earliest finish time (in terms of precedence only)
    $EF_{j^*} = \max_{i \in P_j} \{F_i\} + d_{j^*}$

    Calculate the earliest finish time (in terms of precedence and capacity)
    $F_{j^*} = \min \ \{ \ t \in \left[FMC_{j^*} - d_{j^*}, \infty \right] \cap \Gamma_g \ \mid \ r_{j^*,k} \le RD_k(\tau), \ k \in K \mid r_{j^*,k} > 0, \ \tau \in \left[t, \ t + d_{j^*} \right] \ \} + d_{j^*}$

    Update $S_g = S_{g-1} \cup \{ j^* \}$, $\Gamma_g = \Gamma_{g-1} \cup \{ F_{j^*} \}$

    Iteration increment: $g = g+1$

    Update $A_g, E_g, RD_k(t) \mid t \in \left[F_{j^*} - d_{j^*}, F_{j^*} \right], k \in K \mid r_{j^*,k} > 0$
  }
  Determine the time associated with activity g

  $t_g = \min \left\{ t \in \Gamma_{g-1} \mid t > t_{g-1} \right\}$
}

Fig. 6 - Pseudo-code to construct parameterized active schedules, Mendes et al. [6]

The basic idea of parameterized active schedules is incorporated in the selection step of the procedure,

$$j^* = \underset{j \in E_g}{\operatorname{argmax}} \left\{ PRIORITY_j \right\} .$$

The set E_g is responsible for forcing the selection to be made only amongst activities which will have a delay smaller or equal to the maximum allowed delay.

The parameters $PRIORITY_j$ and $Delay_g$ (priority of activity j and delays used at each g) are supplied by the genetic algorithm.

6 Local Search

Local search algorithms move from solution to solution in the space of candidate solutions (the search space) until a solution optimal or a stopping criterion is found. In this paper was applying backward and forward improvement based on Klein [10].

Initially is constructed a schedule by planning in a forward direction starting from the project's beginning, see Fig. 7. After is applying backward and forward improvement based on Klein [10].

The backward planning consists in reversing the project network and applying the scheduling generator scheme.

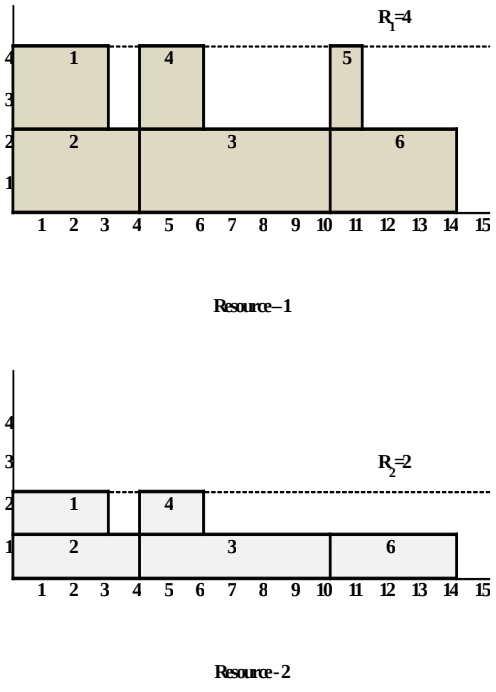

Fig.7 – Feasible schedule with a makespan of 14.

Fig. 8 presents the solution obtained by backward planning. Having scheduled the dummy end activity 7, activities 5 and 6 which are backward eligible at t = 14 can be executed in parallel. Due to the precedence constraint, activities 3 and 4 are scheduled that they finishes at t = 13 and t = 10, respectively. Finally activities 1 and 2 are scheduled. By reducing all the scheduled starting and finisinh times by 3, a schedule with a makespan of 11 is obtained, i.e., the initial schedule (Fig. 7) is improved.

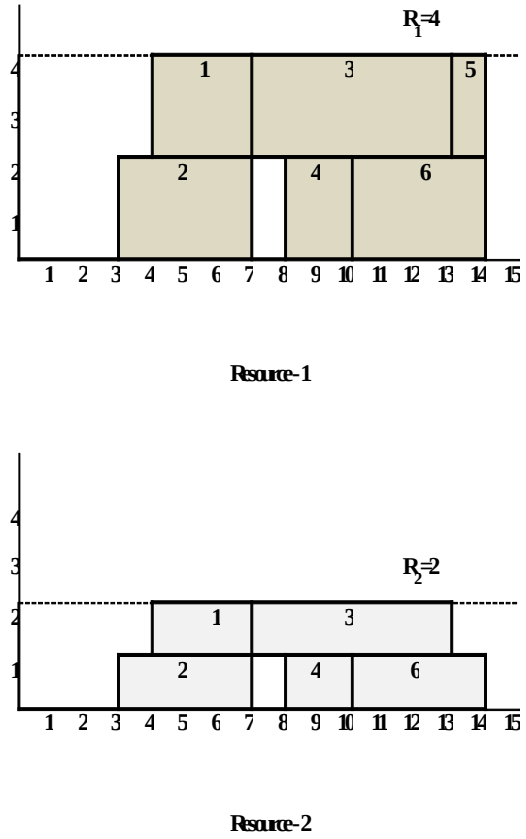

Fig.8 – Improved schedule with a makespan of 11.

Fig. 9 presents the solution obtained by forward planning. Having scheduled the dummy initial activity 0, activities 1 and 2 which are eligible at t = 0 can be executed in parallel. Due to the precedence constraint, activities 3 and 4 are scheduled that they finishes at t = 10 and t = 6, respectively. Finally activities 5 and 6 are scheduled. A schedule with a makespan of 11 is obtained without improving the schedule in Fig. 7.

7 Computational results

This section presents results of the computational experiments done with the algorithm proposed in this paper. The experiments were performed on an Intel Core 2 Duo CPU T7250 @2.00 GHz. The algorithm was coded in Visual Basic 6.0. The GA-RKV (Genetic Algorithm - Random Key Variant) was tested on the instance sets:

- J30 (480 instances each with 30 activities)
- J60 (480 instances each with 60 activities)
- J120 (600 instances each with 120 activities)

available in PSPLIB. All problem instances require four resource types.

Instances details are described in Kolisch et al. [24].

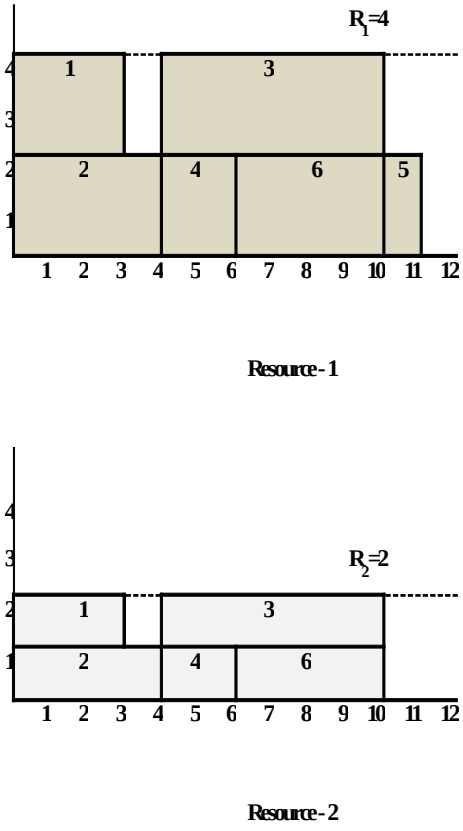

Fig.9 – Final schedule with a makespan of 11.

7.1 Genetic algorithm configuration

Though there is no straightforward way to configure the parameters of a genetic algorithm, we obtained good results with values: **population size** of 5 × number of activities in the problem; **mutation probability** of 0.001; **top** (best) 1% from the previous population chromosomes are copied to the next generation; **stopping criterion** of 250 generations.

7.2 Experimental results

The results obtained are given in Tables 5, 6 and 7.

Table 5, column 3, summarizes the average deviation percentage from the optimal *makespan* ($\boldsymbol{D_{OPT}}$), for the instance set J30. The GA-RKV obtained $\boldsymbol{D_{OPT}}$ = 0.01. The number of instances for which the algorithm obtains the optimal solution is 476. For the set J30, GA-RKV ranks five.

Algorithm	***Reference***	***J30*** $\boldsymbol{D_{OPT}}$
MAPS	Mendes and Gonçalves [15]	0.00
GA-TS path relinking	Kochetov and Stolyar [11]	0.00
Decomp. & local opt.	Palpant et al. [9]	0.00
F&F(5)	Ranjbar [14]	0.00
GA - RKV	***This paper***	**0.01**
GAPS	Mendes et al. [6]	0.01
Scatter Search - FBI	Debels et al. [3]	0.01
VNS-activity list	Fleszar and Hindi [8]	0.01
GA - DBH	Debels and Vanhoucke [4]	0.02
GA – hybrid, FBI	Valls et al. [12]	0.02
GA - FBI	Valls et al. [13]	0.02

Table 5: Top-ten computational results for J30 instances.

Algorithm	***Reference***	***J60*** $\boldsymbol{D_{LB}}$
F&F(5)	Ranjbar [14]	10.56
MAPS	Mendes and Gonçalves [15]	10.64
GAPS	Mendes et al. [6]	10.67
GA - DBH	Debels and Vanhoucke [4]	10.68
Scatter Search - FBI	Debels et al. [3]	10.71
GA – hybrid, FBI	Valls et al. [12]	10.73
GA, TS – path relinking	Kochetov and Stolyar [11]	10.74
GA - FBI	Valls et al. [13]	10.74
Decomp. & local opt.	Palpant et al. [9]	10.81
GA - RKV	***This paper***	**10.88**
VNS-activity list	Fleszar and Hindi [8]	10.94

Table 6: Top-ten computational results for J60 instances.

Tables 6 and 7, columns 3, summarize the average deviation percentage from the well-known critical path-based lower bound ($\boldsymbol{D_{LB}}$) for the instance set J60 and J120, respectively. For the instances set J60 and J120, GA-RKV ranks ten. The lower bound values ($\boldsymbol{D_{LB}}$) are reported by Stinson et al. [7].

Algorithm	***Reference***	***J120*** $\boldsymbol{D_{LB}}$
GA-DBH	Debels and Vanhoucke [4]	30.82
MAPS	Mendes and Gonçalves [15]	31.19
GAPS	Mendes et al.[6]	31.20
GA – hybrid, FBI	Valls et al. [12]	31.24
F&F(5)	Ranjbar [14]	31.42
Scatter Search - FBI	Debels et al. [3]	31.57
GA - FBI	Valls et al. [13]	31.58
GA, TS – path relinking	Kochetov and Stolyar [11]	32.06
Decomp. & local opt.	Palpant et al. [9]	32.41
GA - RKV	***This paper***	**32.50**
GA - Self adapting	Hartmann [16]	33.21

Table 6: Top-ten computational results for J120 instances.

The maximum computational time dispended is 120 seconds for each instance of J60 and 300 seconds for each instance of J120.

8 Conclusions and further research

This paper presents a new genetic algorithm (a variant of the genetic algorithm proposed by Goldberg [1] with binary code) for the resource constrained project scheduling problem. The chromosome representation of the problem is based on random keys. Reproduction, crossover and mutation are applied to successive chromosome populations to create new chromosome populations. These operators are simplicity itself, involving random number generation, chromosome copying and partial chromosome exchanging.

The schedules are constructed using a priority rule in which the priorities are defined by

the genetic algorithm with a constructive heuristic.

The constructive heuristic for constructing feasible schedules is extended by the flexible use of different planning directions including the backward and forward planning (LS). For some instance, a combination of a heuristic constructive and the genetic algorithm may yield a good result, but in another instance the LS can improve the initial schedule.

The approach was tested on a set of 1560 standard instances taken from the literature and compared with the best state-of-the-art approaches. The algorithm produced good results when compared with other approaches therefore validating the effectiveness of the proposed algorithm.

Further work could be conducted to explore the possibility of using activities with multi-mode usage of limited resources.

Acknowledgements

This work has been partially supported by the Polytechnic of Porto, IPP/PADInv2007.

References:

[1] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, 1989.

[2] D. Beasley, D.R. Bull and R.R. Martin, *An Overview of Genetic Algorithms: Part 1, Fundamentals*, University Computing, Department of Computing Mathematics, University of Cardiff, UK, Vol. 15(2), 1993, pp. 58-69.

[3] D. Debels, B. De Reyck, R.Leus and M.Vanhoucke, A Hybrid Scatter Search/Electromagnetism Meta-Heuristic for Project Sheduling, *European Journal of Operational Research,* Vol. 169, 2006, pp. 638-653.

[4] D. Debels and M. Vanhoucke. *A Decomposition-Based Heuristic for the Resource-Constrained Project Scheduling Problem*. Working Paper 2005/293, Faculty of Economics and Business Administration, University of Ghent, Ghent, Belgium, 2005.

[5] J.F. Gonçalves, J.M. Mendes, and M.C.G. Resende. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, Vol. 167, 2005, pp. 77-95.

[6] J.J.M. Mendes, J.F. Gonçalves and M.G.C. Resende, A random key based genetic algorithm for the resource constrained project scheduling problem, *Computers & Operations Research*, Vol. 36, 2009, pp. 92-109.

[7] J.P. Stinson, E.W. Davis and B.M. Khumawala, Multiple Resource-Constrained Scheduling Using Branch and Bound, *AIIE Transactions*, Vol. 10, 1978, pp. 252-259.

[8] K. Fleszar and K.S. Hindi, Solving the resource-constrained project scheduling problem by a variable neighbourhood search, *European Journal of Operational Research*, Vol. 155, 2004, pp. 402-413.

[9] M. Palpant, C. Artigues and P. Michelon, LSSPER: Solving the resource–constrained project scheduling problem with large neighbourhood search, *Annals of Operations Research*, Vol.131, 2004, pp. 237-257.

[10] R. Klein, Bidirectional planning : improving priority rule-based heuristics for scheduling resource-constrained projects, *European Journal of Operational Research,* Vol. 127, 2000, pp. 619-638.

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

[12] V. Valls, F. Ballestin and M.S. Quintanilla. *A hybrid genetic algorithm for the RCPSP*. Technical report, Department of Statistics and Operations Research, University of Valencia, 2003.

[13] V. Valls, F. Ballestin and M.S. Quintanilla, Justification and RCPSP: A technique that pays, *European Journal of Operational Research*, Vol.165, 2005, pp. 375-386.

[14] M. Ranjbar, Solving the resource-constrained project scheduling problem using filter-and-fan approach, *Applied Mathematics and Computation*, Vol. 201, 2008, pp. 313–318.

[15] J.J.M. Mendes and J.F. Gonçalves, A Memetic Algorithm-Based Heuristics for the Resource Constrained Project Scheduling Problem, Proceedings of *II International Conference on Computational Methods for Coupled Problems in Science and Engineering*, Spain, 2007, pp. 644-648.

[16] R. Kolisch and S. Hartmann, Experimental investigation of heuristics for resource-constrained project scheduling: an update, *European Journal of Operational Research*, Vol.174 (1), 2006, pp. 23-37.

[17] M. Seda, Solving Resource-Constrained Project Scheduling Problem as a Sequence of Multi-Knapsack Problems, *WSEAS Transactions on Information Science & Applications*, Issue 10, Vol. 3, 2006, pp.1785-1791.

[18] M. Kljajc, U. Breskvar and B. Rodic, Computer aided scheduling with use of genetic algorithms and a visual discrete event simulation model, *WSEAS Transactions on Systems*, Issue 3, Vol. 3, 2004, pp. 1021-1026.

[19] C.H. Yeh and H. Pan, System Development for Fuzzy Project Scheduling, *WSEAS Transactions on Business and Economics*, World Scientific and Engineering Academy and Society, USA, Vol. 1(4), 2005, pp. 311-317.

[20] J.J.M. Mendes, "Sistema de Apoio à Decisão para Planeamento de Sistemas de Produção do Tipo Projecto", Ph.D. Thesis, Departamento de Engenharia Mecânica e Gestão Industrial, Faculdade de

Engenharia da Universidade do Porto, Portugal, 2003. (*In portuguese*)

[21] R. Kolisch, *Project Scheduling under Resource Constraints*, Physica-Verlag, Germany, 1995.

[22] P. Brucker, A Drexl, R. Mohring, K. Neumann, E. Pesch, Resource-constrained project scheduling: Notation, classification, models and methods, *European Journal of Operational Research*, Vol.112 (1), 1999, pp. 3-41.

[23] N. Christofides, R.Alvarez-Valdés and J. Tamarit, Problem scheduling with resource constraints: A branch and bound approach, *European Journal of Operational Research*, Vol. 29, 1987, pp. 262-273.

[24] R. Kolisch, Schwindt, A.Sprecher, *Benchmark instances for scheduling problems*. In J.Weglarz, (ed.) Handbook on recent advances in project scheduling, Kluwer, Amsterdam, 1998, pp. 197-212.

[25] C. Patrick, *Construction Project Planning and Scheduling*, PEARSON Prentice Hall, Columbus, Ohio, 2004.

[26] R. Kolisch and S. Hartmann, *Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis*, J. Weglarz (editor), Kluwer, Amsterdam, the Netherlands, 1999, pp. 147–178.

[27] J.F. Gonçalves and N.C. Beirão, Um algoritmo genético baseado em chaves aleatórias para sequenciamento de operações, *Revista Associação Portuguesa Investigação Operacional*, Vol. 19, 1999, pp.123–37, (in Portuguese).

[28] H. Pan and C.H. Yeh, GA for Fuzzy Multi-Mode Resource-Constrained Project Scheduling, *WSEAS Transactions on Systems*, Issue 4, Volume 2, October 2003, pp.893-990.