

A Dynamic Population Steady-State Genetic Algorithm for the Resource-Constrained Project Scheduling Problem

Mariammar Cervantes^{1,2}, Antonio Lova³, Pilar Tormos³, and Federico Barber¹

¹ DSIC – Universidad Politécnica de Valencia, Spain
{mcervantes, fbarber}@dsic.upv.es

² Universidad de La Sabana, Bogotá – Colombia

{maria.cervantes}@unisabana.edu.co

³ DEIOAC – Universidad Politécnica de Valencia, Spain
{alova, ptormos}@eio.upv.es

Abstract. Resource Constrained Project Scheduling Problem (RCPSp) is a well known problem that is easy to describe but very difficult to solve, and therefore, it has attracted the attention of many researchers over the last few decades. In this context, heuristics are the only option when solving realistically-sized projects. In this paper we develop a steady-state genetic algorithm that uses a dynamic population and four decoding methods. These features allow the algorithm to adapt itself to the characteristics of the problem. Finally, its performance is compared against the best project scheduling methods published so far. The results show that the proposed scheduling method is one of the best scheduling techniques when compared with results reported in the literature.

Keywords: project scheduling, resource constraints, genetic algorithm.

1 Introduction

The Resource Constrained Project Scheduling Problem (RCPSp) is a well-known problem which is widely studied in the literature. In recent years, many works have been published, including [6], which summarizes the research for this problem.

Assume a project is represented in activity-on-the-node format by a directed graph $G=(N,A)$ in which N is the set of activities that have to be processed and A is the set of precedence relationships. The non-preemptable activities are numbered from 0 to $n+1$, where the dummy activities 0 and $n+1$ mark the beginning and the end of the project. The duration of an activity is denoted by d_i where the duration is zero for dummy activities and non-zero for others. The start time of each activity is denoted by S_i ($0 \leq i \leq n+1$) and its finish time by F_i ($0 \leq i \leq n+1$). There are K renewable resource types, r_{ik} ($0 \leq i \leq n+1, 1 \leq k \leq K$) being the resource requirements of activity i with respect to resource type k . The resource requirement of dummy activities is 0. R_k ($1 \leq k \leq K$) is the constant availability of resource type k throughout the duration of the project.

Based on the definitions of the variables and considering that $P(t)$ denotes the set of activities in process at time t , T being an upper bound on the feasible project duration, the RCPSP can be formulated as follows [3]:

$$\text{Minimize } S_{n+1} . \quad (1)$$

Subject to:

$$S_j - S_i \geq d_i \quad (i, j) \in A . \quad (2)$$

$$\sum_{i \in P(t)} r_{ik} \leq R_k \quad t = 1, \dots, T ; \quad k = 1, \dots, K . \quad (3)$$

$$S_i \geq 0, \quad i \in N . \quad (4)$$

The objective function (1) minimizes the completion time of the dummy activity $n+1$ and thus the makespan of the project. Constraints (2) enforce the precedence constraints between activities. Constraints (3) ensure that for each resource k and each time period t , the resource demand of the set of activities that are currently being processed does not exceed the available capacity.

As the RCPSP is a generalization of the job shop problem, it belongs to the class of NP-hard problems [2]. Taking the combinatorial nature of the problem, the majority of scheduling algorithms are heuristics. With this background in mind, the main challenge is to obtain better solutions (e.g. reduce the project makespan) in less computational time. Several heuristic procedures can be applied and one of the most widely used is Genetic Algorithms because of its good performance.

In this paper we develop a steady-state genetic algorithm which builds individuals (e.g. project schedules) using four codification methods based on priority rules, and an improvement method. The codification methods are also used to evaluate the individuals that are generated via the application of the genetic operators.

The algorithm has two major innovative features which make it stand out when compared with previous works in the RCPSP [7]. First, it has a dynamic population which means that the number of individual increases as the search deepened. Secondly it uses a steady-state replacement strategy.

The computational experiments show that the performance of our algorithm can be consider one of the best, and in certain instances it obtains the optimal solution in the first iteration.

The remainder of this paper is organized as follows; firstly we review the methods that will be used in the generation and evaluation of individuals. In section 3 we describe in detail the genetic algorithm, including the new operators developed and their settings. Section 4 is devoted to describe the results obtained for the standard set of project instances with 30, 60 and 120 activities of the well known PSPLIB. In Section 5 we compare the effectiveness of the algorithm against the best project scheduling methods currently available. Finally, we highlight the main conclusions and directions of further research.

2 Representation and Decoding of Schedules

One of the key factors of genetic algorithms is the representation of solutions. When representations are easy to decode the computational time needed to obtain the schedule is low. Furthermore, it is possible to exploit the properties of the representation in order to design genetic operators that improve the performance of the algorithm.

In this work we use the *activity list* (AL) representation because it was determined that it performs the best to solve the RCPSP [1]. An AL can be seen as an array with one activity in each position. An AL is *precedence-feasible* when an activity never comes after the position of one of its successors.

In order to generate the activity list for the individuals of the initial population and the individuals that are introduced when the population grows, we use a priority rule based heuristic. A priority rule is a criterion to decide the order in which the activities are going to be scheduled and is based on activity's characteristics.

In addition we use the *Parameterised Regret-Biased Random Sampling* (RBRS [7]) in which a probability of being selected is assigned to each activity from the eligible set, depending on its priority value. An eligible activity is one whose predecessors are all in the partial schedule. Each pass of the method obtains a different schedule and the best one will be the final schedule.

The RBRS uses α as a parameter to control the amount of bias. When $\alpha=0$, there is a pure random activity selection. On the other hand, if α is large the activity selection is deterministic.

2.1 Schedule Generation Scheme

To decode the activity list generated by means of the priority rule and the RBRS we need to use a schedule generation scheme.

There are two different schedule generation schemes (SGS), *serial* and *parallel*, that can be applied in two scheduling directions (SD) *forward* and *backward*. Both schemes build feasible project schedules by stepwise extension of a partial schedule (a schedule where only a subset of activities has been scheduled).

In the Serial SGS we schedule the activities one by one, in the order given by the list. In the Parallel SGS the position number of the activity will indicate its priority to be scheduled, the activity with the lowest position being the activity with the highest priority [4]. Both scheduling schemes are presented in detail in [5].

Each SGS can be applied in two schedule directions (SD): Forward and Backward. A forward schedule is built from the first activity to the last one, and each activity is scheduled in the earliest time within the precedence and resource constraints. On the contrary, a backward schedule starts from the last activity and finishes in the first one, scheduling the activities as late as possible.

2.2 Improving Method: Backward-Forward Method

This procedure was proposed by Tormos and Lova in [11], and more recently used in [12] and [14] among many others. Once a feasible schedule has been obtained, the Forward-Backward Method employs the serial SGS in order to iteratively schedule the project by alternating between backward and forward scheduling (BF pass).

Starting from a Forward Schedule, the procedure creates an AL by sorting the activities in decreasing order of their finish times. Then the S-SGS is used to build a Backward Schedule. (Backward Pass). The b/f gene of the individual changes from *f* to *b* after the Backward Pass. Then, one BF pass implies the generation of 2 solutions.

The Forward Pass starts from a Backward Schedule, sorting the activities in increasing order of their start times. The S-SGS is used to build a Forward Schedule. In this case the b/f gene changes from B to F. Because the use of this procedure can not increase the schedule makespan, the new sequence replaces the original in both cases.

3 Genetic Algorithm Proposed

Genetic Algorithms (GA) is the metaheuristic procedure which achieves the best results when solving the RCPSP (see Tables 3, 4 and 5). However, in the most recent works like [1], [4], [7] or [14] among others, authors use different parameters' values for each set of problems and each number of maximum generated schedules, making the solution very dependent on the problem settings. Moreover, in real problems the maximum number of schedules generated is not known in advance, therefore, these assumptions are not practical.

This situation motivated us to build an adaptive algorithm to reduce the external settings by the user. In order to obtain this independence, we developed an original approach for the RCPSP by means of three factors: we used a dynamic population; the local search varies according to the quality of the solution and the degree of depth of the search; and the individuals generated by the genetic operators enter the population immediately. Fig 1 shows the structure of the new algorithm. In the following sections we explain the operators used.

```

GeneticAlgorithm
  CreatePopulation(Population, Pop_Size);
  Evaluation(Population, Local Search)
  While (NSched < MaxSched);
    Selection(Population, F, M);
    Crossover(F, M);
    Mutation(S, D);
    Evaluation(S, D, SelectiveLocalSearch)
    PopulationUpdate(Population, S, D);
    DinamicPopulation(Population, NSched, Pop_Size);
  end While
  Report BestSchedule
End

```

Fig. 1. Pseudo code for the Dynamic Population Steady State Algorithm

3.1 Encoding and Evaluation

We use the AL representation explained above. Each individual in the population is represented by an array with as many positions as activities in the project.

As in [2] each individual has a position (S/P): to code the SGS serial or parallel; and another (B/F) for the direction of the schedule: forward or backward. To compute a schedule we use the SGS and the scheduling direction specified in the genotype. The fitness of the individual is defined as the makespan of the schedule. That is, a lower fitness implies a better individual (and thus a higher chance to survival).

3.2 Initial Population

We start the genetic algorithm by building an initial population using priority rules to determine the order in which the activities will be selected for scheduling.

The generation of the initial population is a two stage procedure: First we perform a deterministic multi-pass priority rule-based method using S-SGS and P-SGS both backward and forward. For each feasible schedule the improvement method iteratively schedules the project by alternating between backward and forward scheduling (BF pass) until no improvement is achieved in two consecutive BF iterations.

The second stage randomly selects the SGS, the SD and the priority rule, and builds an individual by means of the RBRS, using $\alpha=2, 3.5$ and 10 for the sets J30, J60 and J120 respectively as in [8]. Based on the population sizes in [4], the initial population size is fixed at $2000/\text{number_of_activities}$.

3.3 Selection and Crossover Operators

We use the standard 2-tournament selection operator. In this selection procedure, two individuals are chosen randomly, and the element with the lowest makespan is selected. Afterwards, we determine randomly whether a or b represents the father F. The other parent represents the mother M.

The crossover used in this work is the standard 2-point crossover operator which exploits the advantages of the codification chosen, randomly mating pairs of individuals of the population. The crossover operation is performed with a probability of P_{cross} . If the operator is not performed they remain unaltered for the next stage. P_{cross} is set to 0.8 .

3.4 Mutation Operator

We use the insertion operator for the activity list and the standard flip bit for the S/P and B/F genes.

The insertion operator works as follows: for each activity in the AL, a new position is randomly chosen between the highest position of its predecessors and the lowest position of its successors. The activity is inserted into the new position with a probability of $P_{\text{muta}}=0.05$.

The s/p and b/f genes could be altered with the same probability. That is, the p/f could change from p to s or from s to p , while the b/f gene could change from b to f or from f to b .

3.5 Selective Local Search

We use the improvement method described in section 2.2 to improve the quality of the individual. As in [12] the improving method is selectively applied to two types of

schedules: We consider a schedule as a **Good Schedule** if its makespan is less than the mean of the solutions generated so far, and a **Very Good Schedule** if its makespan is less than $(\text{mean} - \text{best}/2)$.

In this respect, the novelty of our approach is that the intensity of the local search is related to the number of schedules (NSched) generated so far. If the number of schedules generated so far is less than 5000 we use a simple B or F pass for good schedules and a BF pass for very good schedules. If the total solutions are greater than 5000 we use a maximum of two BF passes for good schedules and a maximum of 3BF passes for very good schedules.

When we say a maximum of 2 or 3 BF passes we mean that the process ends when no improvement is achieved in two consecutive BF iterations.

3.6 Replacement of the Individuals

Another innovative feature of this algorithm is that, instead of using generational replacement we develop a steady state strategy. In RCPSP literature all GAs use generational replacement [7], which means that the complete population is replaced by the offspring population. Alternatively, in the steady state strategy the new individuals are available to be selected in the next iteration.

The new individuals enter the population and replace the worst if (i) they are better than the worst and (ii) the AL of the new individual is different to the AL of all of the individuals in the population with the same fitness as the new individual. The second condition was developed to diminish the high selective pressure over the population that is generated by replacing the worst elements of the population [10].

3.7 Population Size

In previous works, authors use different population sizes for each set of problems and for each number of maximum schedules (1.000, 5.000 and 50.000). Therefore, the algorithm becomes dependent on the parameters of the problem that must be known in advance.

To overcome this problem we have designed an algorithm with a dynamic population size, meaning that the algorithm starts with a determined number of individuals and as the search is deepened, the size of the population grows.

In order to implement this feature we increase the population size by 25% of the previous population size each time that 1000 new schedules have been evaluated.

The new individuals are generated using the priority rule and RBRS described in section 2.2 and the selective local search described in section 3.6. These individuals are added to the population as soon as they are evaluated.

4 Computational Experiments and Results

In the experiments carried out in this study we use the standard set of project instances with 30, 60 and 120 activities (small-sized, medium-sized and large-sized projects respectively) from the well-known PSPLIB library [9] using an implementation in C compiled with Microsoft Visual C++ v.6.0, running on a 1 GB RAM, 3 GHz Pentium under Windows XP.

To characterize the algorithm, we calculate average deviation over the critical-path based lower bound (%ADLB). Since the optimal solutions for the small-sized projects are known, the lower bound is the optimal solution. As for the other two instance sets, some of the optimal solutions are not known the critical-path based lower bound has been used. In table 1, we present the main results of the steady-state and dynamic population genetic algorithm (SS-DP) for 1000, 5000 and 50000 schedules respectively.

Table 1. Results of the SS-DP Genetic Algorithm

	1,000		5,000		50,000	
	%ADBL	Std Dev	%ADBL	Std Dev	%ADBL	Std Dev
Small sized	0.16	0.58	0.04	0.26	0.01	0.11
Medium-sized	11.43	22.76	10.96	21.85	10.81	21.53
Large-sized	33.94	45.34	32.57	44.24	31.65	43.24

We see that the algorithm evolves successfully, finding better solutions as the number of schedules increases. In the three sets of problems, there are cases where the best solution found is the first one, all of them being the optimal. This is because of the initial population generating scheme, which enables the algorithm to find better solutions in early stages.

In order to evaluate the impact of the steady state replacement and dynamic population strategies we have designed three more algorithms (i) Generational Replacement and Static Population, (ii)Generational Replacement and Dynamic Population and (iii) Steady State Replacement and Static Population. The settings of the genetic operators are the same as the new algorithm proposed. The results can be seen on Table 2.

Table 2. %ADBL Algorithms for comparison

	(i)			(ii)			(iii)		
	1,000	5,000	50,000	1,000	5,000	50,000	1,000	5,000	50,000
Small sized	0.15	0.05	0.04	0.15	0.06	0.02	0.16	0.05	0.04
Medium-sized	11.42	11.03	10.84	11.42	11.01	10.87	11.43	10.98	10.85
Large-sized	34.29	33.31	32.54	34.00	33.16	32.54	33.94	32.75	31.98

As can be seen, the SS-DP Genetic Algorithm performs better when compared with the previous three algorithms, confirming that the new strategies used work through the entire search and the impact is bigger on the large-sized set, which is the most difficult to solve. In particular, the steady state (SS-GA) replacement seems to have more impact than the dynamic population. Both SS-GA outperform the generational ones.

5 Comparison with the Best Heuristics for the RCPSP

The results of the genetic algorithm show that it obtains an excellent performance when compared against the best five heuristics published and reported in [6] and they are ordered by the results obtained for 1,000 schedules.

Small-Sized Projects

The increase, with respect to the optimal solution when 1,000 schedules are generated, is 0.16%, optimally solving 92.29% of the instances. According to the results, the proposed algorithm ranks second. When 5,000 solutions are generated, the increase is 0.04%, coming in second in the ranking, optimally solving 97.29% of the instances. For 50,000 solutions we obtained 0.01% of deviation over the optimal solutions, optimally solving 99.17% of the instances. For the complete results see Table 3.

Table 3. Small-sized set

Heuristic	SGS	Reference	% ADBL		
			1,000	5,000	50,000
GA- TS - path relinking	P/S	Kochetov and Stolyar (2003)	0.10	0.04	0.00
GA- SS - DP	P/S	This paper	0.16	0.04	0.01
Comb-RBRS BF/FB	P/S	Tormos and Lova (2003b)	0.25	0.13	0.05
GA- fwd-backwd	P/S	Alcaraz et al. (2004)	0.25	0.06	0.03
GA - hybrid	S	Valls et al. (2003)	0.27	0.06	0.02
Scatter Search	S	Debels et al. (2007)	0.27	0.11	0.01

Medium-Sized Projects

With this set of project instances the SS-DP Genetic Algorithm improves its performance, ranking first on the list with 1,000 and 5,000 schedules obtaining 11.43% and 10.96 %ADBL respectively. For 50,000 schedules it obtains 10.8 1%ADBL as shown in Table 4.

Table 4. Medium-sized projects

Heuristic	SGS	Reference	%ADBL		
			1,000	5,000	50,000
GA- SS - DP	P/S	This paper	11.43	10.96	10.81
GA-hybrid	S	Valls et al. (2003)	11.56	11.10	10.73
GA, TS – path relinking	P/S	Kochetov and Stolyar (2003)	11.71	11.17	10.74
Scatter Search	S	Debels et al. (2007)	11.73	11.10	10.71
Comb-RBRS BF/FB	P/S	Tormos and Lova (2003b)	11.88	11.62	11.36
GA – fwd-backwd	P/S	Alcaraz et al. (2004)	11.89	11.19	10.84

Table 5. Large-sized projects

Heuristic	SGS	Reference	%ADLB		
			1,000	5,000	50,000
GA- SS - DP	P/S	This paper	33.71	32.57	31.65
GA - hybrid	S	Valls et al. (2003)	34.07	32.54	31.24
GA, TS - path relinking	P/S	Kochetov and Stolyar (2003)	34.74	33.36	32.06
RBRS BF/FB	P/S	Tormos and Lova (2003 b)	35.01	34.41	33.71
Population-based	S	Valls (2005)	35.18	34.02	32.81
Scatter Search	S	Debels et al. (2007)	35.22	33.10	31.57

Large-Sized Projects

The percentage of increase over the lower bound of the algorithms analyzed is reported in Table 5. The proposed algorithm obtains 33.71 %ADBL with 1.000 schedules, taking first place in the ranking. With 5.000 schedules the algorithm ranks second with 32.57 %ADBL and for 50.000 schedules it obtains 31.65 %ADBL.

6 Conclusions

In this paper a genetic algorithm has been developed with two innovative features that make it different when compared with previous works. Firstly, it has a dynamic population meaning that the number of individuals increases as the search deepens. Secondly it uses a steady-state replacement strategy.

The two features are complementary. The steady-state replacement generates high selective pressure. The dynamic population introduces variety to the population on a periodic basis.

We use the problem's characteristics and the degree of depth of the search to set the algorithm parameters, in particular the intensity of the local search.

We found that using good priority rules and the improving method when generating the initial population, the algorithm can find optimal solutions in the first schedule.

Finally, once the dynamic population steady-state algorithm heuristic has been set, we have compared the performance of our approach with the most recent heuristics and metaheuristics developed for RCPSP using the well known standard sets of instances PSPLIB. The results indicate that the proposed heuristic is the best for the medium-sized and large-sized set of problems, making our genetic algorithm competitive with the best scheduling methods published so far in the literature.

Acknowledgment. This work has been partially supported by the research projects TIN2004-06354-C02- 01 (Min. de Educación y Ciencia. Spain-FEDER). FOM-70022/T05 (Min. de Fomento. Spain) and FP6-021235-2. IST-STREP (UE).

References

1. Alcaraz, J., Maroto, C.: A Robust Genetic Algorithm for Resource Allocation in Project Scheduling. *Annals of Operations Research*, vol. 102, pp. 83–109. Kluwer Academic Publishers, Dordrecht (2001)
2. Blazewicz, J., Lenstra, J., Rinnooy Kan, A.H.G.: Scheduling Subject to Resource Constraints: Classification and Complexity. *Discrete Applied Mathematics*, vol. 5, pp. 11–24. Elsevier, Amsterdam (1983)
3. Christofides, N., Álvarez-Valdés, R., Tamarit, J.M.: Project Scheduling with Resource Constraints. A branch and bound approach. *European Journal of Operational Research* 29, 262–273 (1987)
4. Hartmann, S.: A Self-Adapting Genetic Algorithm for Project Scheduling under Resource Constraints. *Naval Research Logistics*, vol. 49, pp. 433–448. John Wiley & Sons, Chichester (2002)
5. Kolisch, R.: Serial and Parallel Resource-Constrained Project Scheduling Methods Revisited: Theory and Computation. *European Journal of Operational Research* 90, 320–333 (1996)
6. Kolisch, R., Hartmann, S.: Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update. *European Journal of Operational Research* 174, 23–37 (2006)
7. Lancaster, J., Ozbayrak, M.: Evolutionary algorithms applied to project scheduling problems—a survey of the state-of-the-art. *International Journal of Production Research* 45, 425–450 (2007)
8. Lova, A., Tormos, P., Cervantes, M., Barber, F.: An Efficient Adaptive Heuristic for the Resource Constrained Project Scheduling Problem. In: *Proceedings CAEPIA 2007*, vol. II, pp. 259–268 (2007)
9. PSPLIB, <http://129.187.106.231/psplib/>
10. Smith, J.: On Replacement Strategies in Steady State Evolutionary Algorithms. *Evolutionary Computation* 15, 29–59 (2007)
11. Tormos, P., Lova, A.: A Competitive Heuristic Solution Technique for Resource-Constrained Project Scheduling. *Annals Of Operations Research*, vol. 102, pp. 65–81. Kluwer Academic Publishers, Dordrecht (2001)
12. Tormos, P., Lova, A.: An Efficient Multi-Pass Heuristic for Project Scheduling with Constrained Resources. *International Journal of Production Research* 41, 1071–1086 (2003a)
13. Tormos, P., Lova, A.: Integrating heuristics for RCPSP: One Step Forward. Technical Report. Universidad Politécnica de Valencia (2003b)
14. Valls, V., Ballestin, F., Quintanilla, M.S.: Justification and RCPSP: A Technique that Pays. *European Journal of Operational Research* 165, 372–386 (2005)
15. Valls, V., Ballestin, F., Quintanilla, M.S.: A Hybrid Genetic Algorithm for the RCPSP. Technical Report. Department of Statistics and Operations Research. University of Valencia (2003)