EngOpt 2008 - International Conference on Engineering Optimization
Rio de Janeiro, Brazil, 01 - 05 June 2008.

New effective algorithm for Dynamic Resource Constrained Project Scheduling Problem

André Renato Villela da Silva*, Luiz Satoru Ochi and Haroldo Gambini Santos

Universidade Federal Fluminense

Abstract
Many scheduling problems use a solution representation composed of a list of n (number of tasks/activities) values defining when those elements must be scheduled/executed. In resource constrained versions, in addition to dependence constraints, there are other constraints involving the use of an amount of resource to execute the activities. All these constraints may make the solution handling before mentioned very difficult. Many algorithms have troubles to generate feasible solutions, if the instance is hard. This paper proposes a new solution representation that allows a very high degree of freedom at working with generated solutions. A new Evolutionary Algorithm is also proposed to exemplify this capacity.
Keywords: Evolutionary Algorithm, Solution Representation, Project Scheduling Problems, Metaheuristics

1. Introduction

The standard task scheduling problem is composed of a set of n tasks T ($|T| = n$) and a set of m processors P ($|P| = m$). The objective is to assign the tasks on the processors, so that the total processing time, makespan, is minimized. Often, scheduling problems are resource constrained: to activate a task i, we need to pay a cost c_i (retrieved from an amount of available resources). Hence, a new set of objective functions can be elaborated mixing the makespan with the amount of resources. In Resource Constrained Project Scheduling Problems (RCPSP), tasks are replaced by activities (the name "task" will be used for the sake of simplification) there are no processors where activities are assigned; they are executed as a part of a project. There are a lot of models for the RCPSP that use both makespan and the available resources. The reader is referred to [1, 2, 3, 4, 5, 6, 7, 8].

In Dynamic Resource-Constrained Project Scheduling Problem (DRCPSP) model, there is profit p_i, which is an amount of resources, related to the task i that is added to the available resources after the task i activation. This profit will be available for each unit time from the task activation moment until the end of the planning horizon. The DRCPSP, as a generalization of the classical Project Scheduling Problem, belongs to the class of the NP-Hard optimization problems. In this way, the application of exact methods becomes limited, justifying the use of heuristic techniques.

The remainder of this paper is organized as follows. In Section 2, we present the DRCPSP model and existing algorithms to solve it. Section 3 is dedicated to a new solution representation and some algorithms used to solve DRCPSP. The instances used in our experimental tests are outlined in Section 4. Section 5 shows computational results and Section 6 gives the concluding remarks. Future works are presented in Section 7.

2. DRCPSP Model

Let $G = (V, A)$ be a a DAG, where V is the set of vertices and A is the set of arcs that represents the tasks (activities) and their precedences, respectively. Associated to each tasks i there are a activation cost c_i and a profit p_i (positive integer values). There is also a planning horizon represented by a time interval composed of H time units. The objective of the DRCPSP is to maximize the available resources at the end of the planning horizon. In order to earn resources, the only way is by activating some tasks, which generate these resources according to the respective profit of the task (p_i) and the time when the task is activated - a task activated earlier generates more resources. A task remains activated until the end of the planning horizon, thus, the earlier the activation, the greater the amount of generated resources.

*This work was partially supported by CNPq - grant 141074/2007-8

This model has a potential application on industrial production and public service expansion projects which must be done in several stages. Some companies typically expand their services using the profits provided by their own businesses. Hence, it is very important to select which projects are profitable or not.

There are some concepts of the DRCPSP modeling, such as *Activation* that is the entry of a task i in the current partial solution, against payment of a cost c_i. After the activation, a *profit* p_i will be generated at each time unit. *Available task*: a task i is available, if all its predecessor tasks are activated or the task has no precedence. *Planning Horizon* that is a set of time units, $[1..H]$, when the tasks can be activated. *Total profit* (S_i) that is the profit sum of a task i. *Available Resources* (Q_t) that is the amount of resources that can be used to activate tasks, at the end of time unit t. *Time Profit* (P_t) that is the sum of all profits that will be returned at the end of time unit t. These profits will become available resources at time unit $t+1$.

This modeling differs from other resource constrained project scheduling problems because it has a factor called profit that generates resources to be used in future activations. This modeling is called *dynamic* because it is very hard to define, a priori, the amount of available resources at time t.

2.1. Previous Works

The DRCPSP was initially proposed in [9] and was also studied in [10]. In those works, the solution representation was a list of n (number of tasks) non-negative numbers, representing the times when each task was activated. If a task i was not activated, the $i-th$ list position was marked by zero or NULL value. This representation will be called *direct representation* from then on. Once the list is made, the whole scheduling is ready.

A randomized constructive algorithm was also proposed. This algorithm (ADDR) works making a list of available tasks, at each time unit. From this list some tasks are retrieved according to their cost/profit relationship. One task is randomly chosen among then and it is effectively scheduled. Others tasks are chosen until there are no more resources or there are no more available tasks. The amount of resources Q_t and the time profit P_t are computed and algorithm moves to the next time unit.

Using the ADDR, some Evolutionary Algorithms were proposed. The recombination algorithm presented in [10] has better performance than one presented in [9] because that one does not need to check the feasibility of the generated individual. However, it has two serious drawbacks: (i) it must handle two list of available tasks on each parent and (ii) the generated individual tends to be very similar to only one of its parents. This last issue is not completely undesirable, but in most cases, the generated individual is not better than its parents because it takes many tasks from one parent and only few tasks from the other one. This unbalance makes the individual worse than its parents. In other words, the previously proposed recombination algorithm does not make good use of parents scheduling by mixing them.

Others proposed evolutionary operators and techniques are fairly good.

3. New Algorithms to Solve DRCPSP

After studying the behavior of the proposed Evolutionary Algorithms and their operators, we decided to use a more flexible representation that provides us with a real genetic heritage of each parent. In order to do it, the representation must not reflect the final scheduling, but it must indicate the importance of scheduling each task. So, we used an *indirect representation* composed of a list having n continuous values. These values represent the priority degree of each task. An available task with higher priority will be scheduled earlier than a lower priority available task. The priority value is computed as follows, where λ is a random continuous number in (-1,1) - it is done to provide some randomness to generate different individuals.

$$\text{Priority}_i = \frac{p_i}{c_i} + \lambda \qquad (1)$$

Each task receives a priority using the Eq.(1). When we have the entire priority list, it it necessary to know the respective scheduling. For each time unit, new scheduling algorithm sorts the available tasks according to priority list. Now, no random choices are done, but the tasks are scheduled following the

Table 1: Task's informations used in scheduling example

Task	Priority	Cost	Profit
2	2.23	3	2
3	1.78	4	4
4	1.56	1	2
1	0.98	2	1
6	0.45	4	5
5	0.21	2	3

sorted list. If a task has cost greater than available resources, it is not scheduled; otherwise, its cost is retrieved from the amount of resources, its profit is added to time profit and we analyze the next task. When no more tasks are available for scheduling or there are no more resources, this time unit ends and we move to next one. At the end of the algorithm, we have the scheduling provided by priority list and its solution quality. Table 1 and Figure 1 show an example of this new scheduling algorithm, supposing $H = 4$, $Q_0 = 4$, $P_0 = 0$ (simplified notation of Q and P are used in example) and priorities computated just for this example (they are not real values).

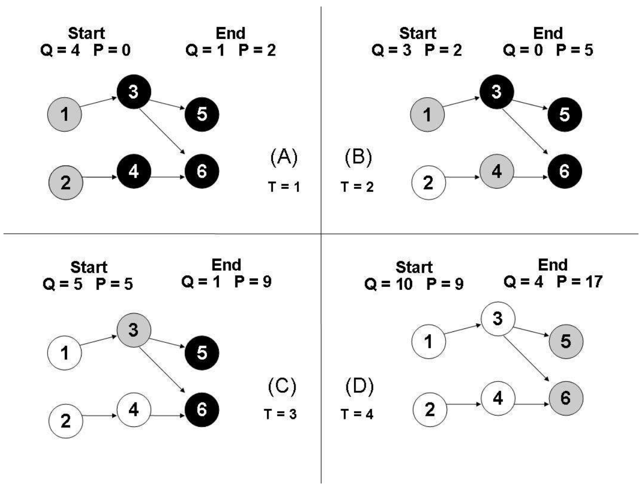

Figure 1: Example of new scheduling algorithm

At start of time $t = 1$ we have two available tasks 1 and 2 (in gray). Other tasks are unavailable (in black) due to precedence constraints. Task 2 has higher priority and, so, it is scheduled first. We need to retrieve 3 units of resource from Q (available resources) and add 2 units to P (profits earned so far). Task 1 is the only available task, but its cost is greater than $Q = 1$. Task 1 can not be scheduled now. Moving to next time $t = 2$, profit P is added to available resources Q, and at this point we have 3 units of resources to spend among tasks. Task 2 is already scheduled (in white) and tasks 1 and 4 are available. Task 4 has higher priority and so, it is scheduled first. We retrieve its cost from Q and add its profit to P. Task 1 can also be scheduled, because we still have 2 units of available resources. Task 1 is scheduled and we finish this time unit having no resources ($Q = 0$), but profit $P = 5$. Moving to next time, we have $Q = 5$, $P = 5$ and only task 3 is available. We schedule it and finish time unit $t = 3$ having $Q = 1$ and $P = 9$. At the last time unit $t = 4$, tasks 5 and 6 are available. We schedule task 6 first and, after it, task 5. Our scheduling finishes and final solution quality is given by the sum of $Q = 4$ and $P = 17$, so 21 units of resource.

3.1. New Effective Evolutionary Algorithm

The use of another representation and another scheduling algorithm demand the use of other genetic operators that make good use of this new situation. In fact, an entire new Evolutionary Algorithm (EA) is necessary. The algorithm proposed in this work is a preliminary version composed of only basic operators.

The initial population is composed of 100 individuals generated according to scheduling algorithm, explained just before. The population is ranked by the individual fitness and divided into three parts: class A has the 20% best individuals; class C has the 20% worst individuals; class B has the other ones.

Crossover operator receives one parent from class A and other from class B. Two children are generated: initially they are identical, and its tasks have $Priority_i$ equal to the average of parent $Priority_i$, for each task. In order to make the two children different, a random number $2 * \lambda$ is added to each priority. From the parent population and the children one, the best 100 individuals are chosen to remain alive, other individuals are discarded.

After the crossover, a mutation algorithm is applied with a chance of 5%. The mutation adds random values to each priority of an individual. These value is computed as follows, where *generation* is the generation number when the mutation is occurring.

$$\text{Value} = \text{generation}^2 * \lambda \tag{2}$$

This formula was made to provide higher perturbation in individuals at later generations. However, such perturbations may make the individual worse. In order to prevent this, we only replace the individual if the mutation improve it; otherwise, the mutated individual is discarded.

The initial stop criterion is 50 generations. In some tests, this criterion is replaced by a time limit. All values and parameter are defined according to preliminary tests.

4. Instances

In [9] two classes (A and B) of instances were used. Class B instances are not hard, because the high number of precedence does not allow many available tasks existing at same time. Class A is composed of 10% of the tasks without any precedence. Others tasks have from 1 to 5 predecessors randomly chosen. The cost of each task c_i is randomly chosen from 1 to 50 and the profit p_i from 1 to 10. Q_0 is randomly chosen from LC to 50, where LC is the lowest cost among the initially available tasks. The time interval H is equal to $\lceil \sqrt{n} \, \rceil$.

These values were defined to provide interesting instances where no trivial solutions are good ones. With these values, we have a large number of tasks that can be activated, and the problem to chose when and what tasks must be activated still remains the most difficult question to answer. These instances are the only ones for this problem.

The instance hardness is proportional to the number of tasks and the planning horizon size (number of time units). In other words, an instance with higher number of tasks is harder than an instance with fewer tasks. In the same way, an instance with a larger number of time units is harder than an instance with fewer time units.

5. Computational Results

All tests were done in the same computational environment used in [9]. In the first test, the new Evolutionary Algorithm was executed 30 times on each instance, using 50 generations as stop criterion. The average results are presented in Table 2.

Table 2: Average Results of Evolutionary Algorithms

Instance	EA3	EA_priority	Improvement
100a	304	303	-0.3%
200a	613	636	3.7%
300a	1734	1958	12.9%
400a	5500	5962	8.4%
500a	11386	11954	5.0%
600a	7097	8616	21.4%
700a	23775	26217	10.3%
800a	31642	32354	2.3%
900a	28379	29912	5.4%
1000a	60751	63512	4.5%

In almost all instance, EA_priority made good improvements. The crossover operator is the main responsible for these improvement. Others tests prove it: the second test consists of running Evolutionary Algorithms by two hours, registering the solution quality of the best individual found. Three instances with 350, 600 and 900 tasks were randomly chosen and the result is showed next.

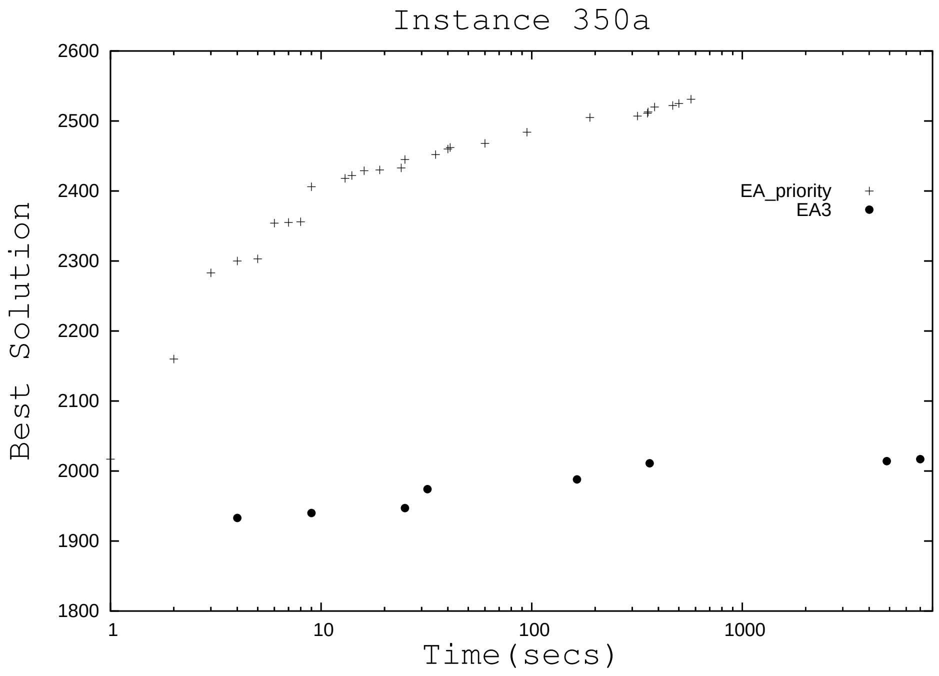

Figure 2: Evolution of Best solution found - instance 900a

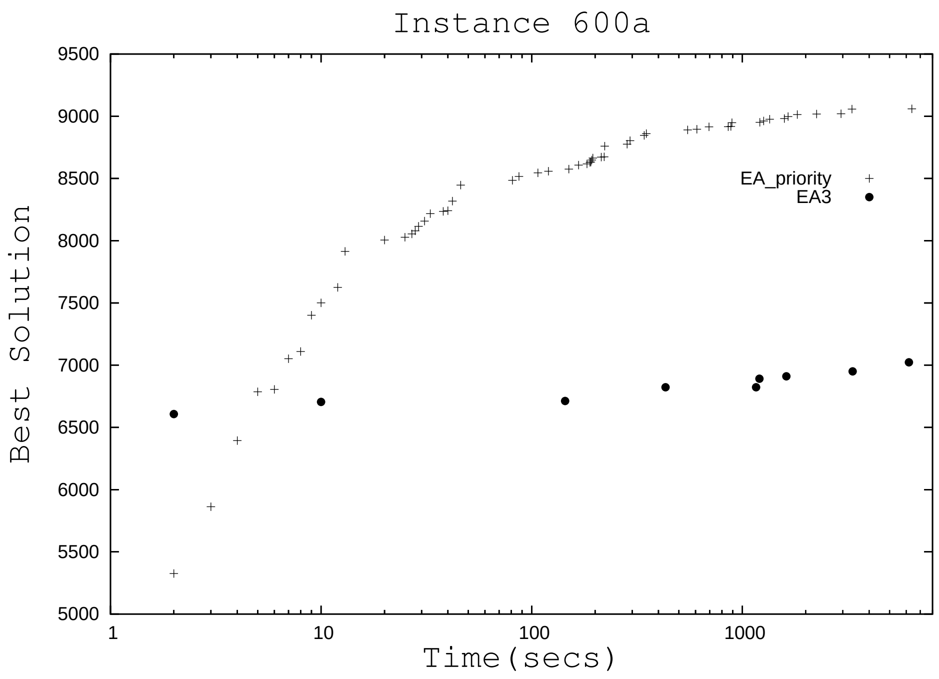

Figure 3: Evolution of Best solution found - instance 600a

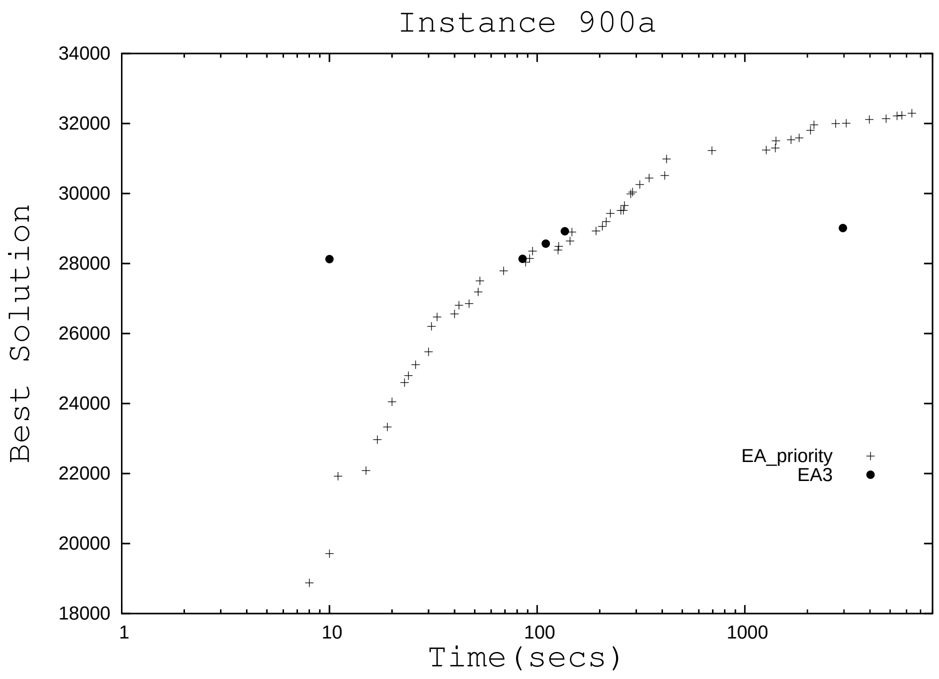

Figure 4: Evolution of Best solution found - instance 900a

These graphs show two main drawbacks of crossover operator of old Evolutionary Algorithm (EA3) used in [10]: (i) it improves the best solution few times and (ii) the improvement obtained is tiny. The new crossover operator, however, makes a lot of improvements. They are also tiny, but at the end of execution, they make a remarkable work. Final best individual generated by EA_priority is 25.5%, 29.0% and 11.3% better than best individual generated by EA3 operator for instances 350a, 600a and

900a, respectively. During EAs executions, others evolutionary operators, like mutation, allow EA3 to generated not so bad results despite crossover. In order to see the real potential of new crossover, a last test was made as follows. An initial population was generated by old scheduling algorithm (Pop. A) and other initial population was generated by new scheduling algorithm (Pop. B). Each population was entirely submitted to both old and new crossover. Average quality of each offspring was compared and the results are presented in Table 3. Table 3 values show how much individuals generated by EA_priority crossover are better than individuals generated by EA3 crossover, in each population.

Table 3: Improvement of new crossover algorithm

Instance	Pop. A	Pop. B
100a	11.5%	-4.9%
200a	14.3%	11.7%
300a	25.9%	95.2%
400a	72.1%	87.7%
500a	61.4%	52.9%
600a	-2.4%	48.2%
700a	42.8%	61.4%
800a	21.1%	38.4%
900a	444.7%	385.1%
1000a	411.3%	272.5%

These old schedulings are converted into priority lists giving a higher priority to tasks activated earlier. When the priority list is made, the crossover is, then, applied on it.

In both initial populations, EA_priority crossover generated individual much better than old crossover. In hard instances, results are great because old crossover could not make good use of parent schedulings. In order to keep feasibility of generated individuals, old crossover generates poor solutions. In new crossover, any scheduling makes a feasible solution, because priority lists do not break any constraints of problem. In order words, any priority list may be mapped into a feasible scheduling. It is very interesting to see that, even in population generated by old scheduling - Pop. A, new crossover obtained very good improvements.

Table 4 shows the same situation in another way: new crossover are much faster than old one because it does not need to manage available task lists from parents and does not care in keeping feasibility. Pop. A and Pop. B have 10000 individuals each one. Old crossover generated 10000 individuals; new crossover generated 20000 individuals due to its building.

Table 4: Average time spent to make each crossover (msec)

	Pop. A		Pop. B	
Instance	EA3 crossover	EA_priority crossover	EA3 crossover	EA_priority crossover
100a	0.36	0.58	0.42	0.31
200a	1.72	1.16	2.63	0.63
300a	4.50	1.90	5.29	1.10
400a	10.84	2.70	12.09	1.43
500a	26.25	3.58	23.24	1.87
600a	31.71	4.61	29.73	2.17
700a	65.34	5.21	59.55	2.91
800a	147.38	5.19	91.61	3.51
900a	102.22	5.85	93.24	4.65
1000a	174.44	7.43	172.24	5.74

6. Concluding Remarks

This paper presented a new way of representing a solution from Dynamic Resource Project Scheduling Problem. One new scheduling algorithm and one new Evolutionary Algorithms are also proposed.

New crossover operator had much better performance than old crossover. This outperformance is very clear when only crossover operator is applied on a population, but we can still see that better performance when a time limit is used as stop criterion.

The results also showed that new representation and new crossover can improve the best solution a lot of times. Otherwise, old representation and crossover can not improve it so frequently.

7. Future Works

The Evolutionary Algorithm proposed in this work still needs a mechanism of avoiding possible premature convergence state or, at least, a diversification algorithm that allows searches in other solution subspaces.

A hybridization technique is also a good possibility, but it must be carefully planned to make good use of the benefits of priority representation.

8. References

[1] J. Blazewicz, J. Lenstra, and A. Rinnooy Kan. Scheduling Subject to Resource Constraints: Classification and Complexity, *Discrete Applied Mathematics*, 1983, 5, 11–24.

[2] C. Boeres, E. Rios, and L. S. Ochi. Hybrid Evolutionary Static Scheduling for Heterogeneous Systems.*Proc. of the IEEE Conf. on Evolutionary Computation (CEC2005)*, 2005, Edinburgh, 1929-1937.

[3] K. Bouleimen and H. Lecocq. A New Efficient Simulated Annealing Algorithm for the Resource-Constrained Project Scheduling Problem and its Multiple Mode Version. *EJOR - European Journal of Operational Research*, 2003, 149, 268-281.

[4] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource-Constrained Project scheduling: Notation, Classification, Models, and Methods. *EJOR - European Journal of Operational Research*, 1999, 12, 3-41.

[5] J. F. Gonçalves, J. J. M. Mendes, and M. G. C. Resende. A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem. *EJOR - European Journal of Operational Research*, 2005, 167, 77-95.

[6] J. F. Gonçalves, J. J. M. Mendes, and M. G. C. Resende. A Genetic Algorithm for the Resource Constrained Multi-project Scheduling Problem. *EJOR - European Journal of Operational Research (accepted)*, 2006.

[7] T. Kalinowski, I. Kort, and D. Trystam. List Scheduling of General Task Graphs Under LogP. *Parallel Computing*, 2000, 26, 1109-1128.

[8] J. Remy. Resource Constrained Scheduling on Multiple Machines. *Information Processing Letters*, 2004, 91, 177-182.

[9] A. R. V. Silva and L. S. Ochi. A Dynamic Resource Constrained Task Scheduling Problem. *Proc. of the Latin-Ibero-American Congress on Operations Research (CLAIO)*, Montevideo, 2006.

[10] A. R. V. Silva and L. S. Ochi. A Hybrid Evolutionary Algorithm for the Dynamic Resource Constrained Task Scheduling Problem. *Proc. of the NIDISC 2007*, LongBeach, 2007.