A dynamic resource constrained project scheduling problem

André Renato Villela da Silva*

Computing Department,
Institute of Science and Technology (PURO),
Federal Fluminense University,
Rua Recife s/n – Jardim Bela Vista,
28890-000, Rio das Ostras, Rio de Janeiro, Brazil
E-mail: avillela@ic.uff.br
*Corresponding author

Luiz Satoru Ochi

Institute of Computer Science,
Federal Fluminense University,
Rua Passo da Pátria,
156 – Bloco E – 3° andar – Boa Viagem,
24210-240, Niterói, Rio de Janeiro, Brazil
E-mail: satoru@ic.uff.br

Abstract: In this paper, we present a variant of classical scheduling problems which deal with restricted resources. Unlike other models studied in the literature, the proposed model has no upper bound for the amount of available resources. The objective of this variant is to find a solution that maximises this amount at the end of the planning horizon. A hybrid evolutionary algorithm is proposed for this problem presenting very promising computational results.

Keywords: project scheduling problem; evolutionary algorithms; mathematical formulation; metaheuristics.

Reference to this paper should be made as follows: da Silva, A.R.V. and Ochi, L.S. (2013) 'A dynamic resource constrained project scheduling problem', *Int. J. Data Mining, Modelling and Management*, Vol. 5, No. 4, pp.370–379.

Biographical notes: André Renato Villela da Silva received his PhD in Computer Science in 2010 from the Federal Fluminense University, Niterói – Brazil. Currently, he is a Professor at the Computing Department (PURO), Federal Fluminense University, Rio das Ostras – Brazil. His research interests include heuristic and hybrid methods, integer programming and project scheduling problems.

Luiz Satoru Ochi received his PhD in System Engineering and Computing at the Federal University of Rio de Janeiro, Rio de Janeiro, Brazil. He is a Full Professor at the Federal Fluminense University, Niterói, Brazil. His research interests include metaheuristics, routing, clustering and scheduling problems.

1 Introduction

The dynamic resource-constrained project scheduling problem deals with an uncommon kind of resource called dynamic resource. Unlike classical project scheduling problems (Chen and Shahandashti, 2009; Damak et al., 2009; Homberger, 2007) where resources may be renewable or non-renewable (both with a bounded quantity), the DRCPSP allows an unbounded amount of these resources because they are consumed when a task is executed, but are produced by tasks after their activations.

DRCPSP can be described as follows. Let $G = (V, A)$ be a directed acyclic graph (DAG), where V is the set of vertices and A is the set of arcs that represent the tasks (activities) and their precedence, respectively. There are an activation cost c_i and a profit p_i (positive integer values) associated to each task i. There are also a planning horizon represented by a time interval composed of H time units, an amount of initial resources Q_0 and an initial accumulated profit $P_0 = 0$. Variables Q_t and P_t represent the available resources and the accumulated profit at time t.

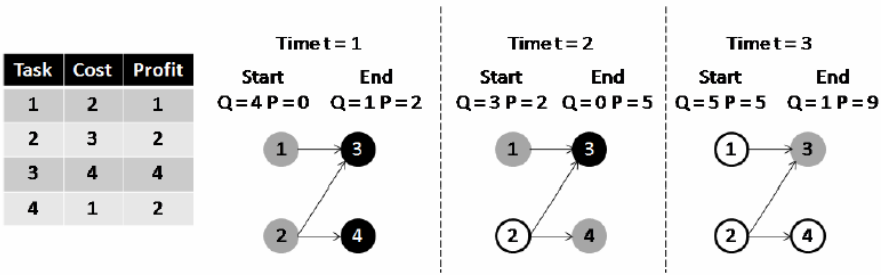
When a task i is activated at time t, it is necessary to pay its cost retrieving c_i from Q_t. From the end of time t to the end of time H (planning horizon size), p_i units of resources are generated by activated task i at each of these time units. In other words, p_i units are added to the accumulated profits from t to H. When the scheduling moves to the next time unit, all accumulated profit becomes available resources and can be used to activate more tasks. The objective of DRCPSP is to maximise the resources ($Q_H + P_H$) at the end of time H. Basically, two constraints are imposed to activate a task i at time t:

1 there must be enough available resources Q_t to activate the task

2 a task may be activated only if all predecessors of i are activated before t.

Figure 1 shows an example of a scheduling, supposing $H = 3$, $Q_0 = 4$, $P_0 = 0$. For the sake of convenience, we will use a simplified notation of Q and P. At the start of time $t = 1$ there are two available tasks 1 and 2 (in grey). Task 2 is scheduled first. We need to retrieve three units of resource from Q (available resources) and add two units to P (profits earned so far). Moving to the next time $t = 2$, profit P is added to available resources Q. Task 2 is already scheduled (in white) and tasks 1 and 4 are available and will be scheduled. We finish this time unit without any resources ($Q = 0$), but profit $P = 5$. Moving to the next time, we have $Q = 5$, $P = 5$ and only task 3 is available. We schedule it and finish time unit $t = 3$ with $Q = 1$ and $P = 9$. Our scheduling finishes and the value of the final solution is given by the sum of $Q = 1$ and $P = 9$ (ten units of resource).

Figure 1 Example of scheduling in DRCPSP



DRCPSP is a NP-hard because it can be seen as a generalisation of the knapsack problem (Poirriez et al., 2009) as follows. Suppose the specific case where there is no precedence among the tasks and $H = 1$. The formulation proposed in the next section can be easily turned into the classical knapsack problem. So, if this specific case is NP-hard, the problem must be also considered NP-hard, for a general instance.

The DRCPSP has a specific application in commercial and industrial environments where expansion plans might take place. Suppose that a company wishes to open new branches in order to increase its financial gains. Each candidate city or place has a building cost related and it is supposed to obtain some sustained profit after its opening. Due to logistic issues (warehouses capacity, transport services or other management issues) it is not possible to open branches anywhere, anytime. Therefore, branches must be opened with some precedence. In general, a fixed budget is available to the expansion plan and all other monetary resources must be acquired by the business profits. Expansion plans like this have a time period to be projected and executed and the main objective is to obtain the maximum amount of monetary resources as possible at the end of the planning horizon.

The remainder of this paper is organised as follows. Section 2 presents a mathematical formulation for the DRCPSP. Heuristic methods are described in Section 3. Section 4 introduces the proposed evolutionary and hybrid algorithms. Computational results are showed in Section 5. Finally, the conclusion is showed in Section 6.

2 Mathematical formulation

The DRCPSP can be written as a mixed integer programme. MIP Binary variables x_{it} indicate if the instant of activation of task i is t (= 1) or not (= 0). Integer variables Q_t and P_t define, respectively, the amount of available resources and the accumulated profit at time t. Constant Q_0 is a problem input data and we consider $P_0 = 0$. $Pred(i)$ represents the set of predecessor tasks of task i.

The objective function (1) seeks to maximise the resources at the end of time H. Constraints (2) ensure that a task i can be activated only if all its predecessor tasks are activated previously. All tasks are inactive at the start of scheduling (3). Constraints (4) show the available resources at time t. Constraints (5) state that the accumulated profit at time t is equal to the accumulated profit at prior time plus the profit of all activated tasks at this time. Constraints (6) ensure that a task i is activated once, at most. Constraints (7) and (8) define the domain of all variables.

$$Max \quad Q_H + P_H \tag{1}$$

s.t.

$$x_{it} \leq \sum_{t'=1}^{t-1} x_{jt'} \quad \forall i = 1,\ldots,n \;\; \forall t = 1,\ldots,H \;\; \forall j \in Pred(i) \tag{2}$$

$$x_{i0} = 0 \quad \forall i = 1,\ldots,n \tag{3}$$

$$Q_t = Q_{t-1} + P_{t-1} - \sum_{i=1}^{n} c_i x_{it} \;\; \forall t = 1,\ldots,H \tag{4}$$

$$P_t = P_{t-1} - \sum_{i=1}^{n} p_i x_{it} \quad \forall t = 1,\ldots,H \tag{5}$$

$$\sum_{t=1}^{H} x_{it} \leq 1 \quad \forall i = 1,\ldots,n \tag{6}$$

$$x_{it} \in \{0,1\} \quad \forall i = 1,\ldots,n \quad \forall t = 1,\ldots,H \tag{7}$$

$$Q_t, P_t \in N \quad \forall t = 0,\ldots,H \tag{8}$$

3 Constructive and local search heuristics

This section presents the constructive method and three local searches which are proposed in order to solve the DRCPSP.

3.1 Constructive method

The proposed randomised constructive heuristic is called ADDR (a randomised version of an ADD-like general constructive heuristic). At each time unit t, the algorithm makes a list of available tasks and sorts it according to c_i/p_i of the tasks. Next, a subset of the p% best tasks is selected (just as the α parameter used in GRASP algorithms by Feo and Resende, 1995). Among these tasks, one is randomly selected. If its cost is not greater than the available resources, this task is activated and Q_t and P_t updated. New random choices are made until there are no available tasks or available resources at time t. The tasks states are updated and scheduling moves to the next time unit until the final time H.

The solution representation is composed of an array of n elements, where n is the number of tasks. Each value of position i represents the time when the task i was activated. If a task was not activated during scheduling its position is marked as NULL (or zero value).

3.2 Enhancement techniques

This work also presents the following set of enhancement techniques (ETs) which try to improve the solution quality generated by the constructive algorithm.

- *Cutting time* (*CT*) – the idea is quite simple: from some time unit on, a task will only be activated if it produces more resources until the end of planning horizon than its cost. The *CT* value must be in the interval [1; *H*] of the planning process. If *CT* = 1, we will always use it; if *CT* = *H*, we will never use it. *CT* value can also be computed in uniform range [0; 1].
- *Relaxation margin* (*RM*) - there are cases where the activation of a non-profitable task *i* might be a good choice: if any successor task *j* is very profitable. However, to assume that a successor task *j* will be activated is very difficult due to problem constraints. Hence, a simple way to allow the activation of this non-profitable task *i*

is to multiply its profit p_i by a factor R and then use it like in CT. Factor R is the relaxation margin.

- *Previous weighting* (PW) – the third ET works as a preprocessing of the graph G. For each task i, its earliest time when that can be activated is computed. This time is multiplied by the profit pi and the product is called *maximum profit* of the task i ($M P_i$). This value is the largest amount of resources that can be generated by the task i in the best case. Therefore, the sorting criterion for the available tasks is now $c_i / M P_i$ instead of the former c_i/p_i. The objective is to give priority to the tasks which can generate more resources.
- *Arc removal* (AR) – it is a reduction rule and does not modify the solution value, but often reduces the processing time, by removing some redundant arcs of the input graph. An arc (i, j) can be removed if there still is a path K from i to j in the resulting graph. AR will be used in all instances.

All ETs were intensively tested and the results obtained showed that they really improved the solution quality. The values used in the ETs must be well calibrated in order to obtain such good results. The calibration is done by running (20 times) each technique, using a set of predefined values. The value that generates the best solution quality is chosen to be used from then on. The tested values are: α – from 0.05 to 0.4 with 0.05 increments; CT – from 0.2 to 0.7 with 0.1 increments; RM – from 1.0 to 1.4 with 0.1 increments and PW – {0, 1} (use or do not use).

3.3 Local search

In order to improve an initial solution three improvement algorithms are presented: the first one (LS1) analyses each task i, performing the same computation made by CT. If task i has a cost greater than the produced resources, it is removed from the current solution. The second improvement algorithm (LS2) is a generalisation of the LS1 and works with a set of all successors of i ($Suc(i)$). If the total cost of set $Suc(i)$ is greater than the total profit generated by it, the whole set is removed from the current solution. The last one, LS3, tries to reconstruct the current solution with a lesser greedy criterion than ADDR. LS3 randomly chooses a time unit TL in the range $[H/4; H/2]$. Tasks activated after TL are removed and new tasks are activated according to the well-known roulette technique (the higher the profit of a task the greater the chance of this task to be chosen). Naturally, only available tasks may be chosen at each time unit.

4 Evolutionary algorithms

Evolutionary algorithms (EA) like those in Gonçalves et al. (2009) and Holland (1975) are heuristics that work with a population of solutions. EAs have been widely used in several areas to solve problems considered intractable although its basic versions do not demonstrate so much efficiency with high complexity problems as shown by Santos et al. (2006).

Three EAs (EA1, EA2 and EA3) are proposed to solve the DRCPSP. EA1 is a basic version including only *arc removal*. EA2 is a more complex version including all the ETs

and proposed local searches LS1 and LS2. EA3 includes all features (techniques, local searches and others mechanisms) proposed in this work.

The initial population is created by ADDR. All populations are always composed of distinct individuals and they are sorted by the fitness (decreasingly). They are also divided into three classes: (A: 20% best individuals; C: 20% worst individuals; B: other individuals). When two parents are chosen to generate offspring, one parent always comes from class A and the other one always comes from class B. This rule was imposed to allow the selection of two fairly distinct parents.

EA1 and EA2 use a recombination algorithm called *best parent* (BP). For each task of the children, its activation time will be the lowest value from the parents. If this value makes the child infeasible, the other value is chosen. If the child remains infeasible, the task is removed (zero value is given). EA3 uses another recombination algorithm called *unique list* (UL). Instead of choosing the earliest activation time from each parent, this algorithm makes a UL of available tasks from each parent at each time. From this list some tasks are chosen like in ADDR. The list of parent available tasks is updated according to the tasks chosen for the child. The algorithm continues until the time unit H. LS1 is applied to each individual generated by the recombination algorithm in EA2 and EA3.

The traditional mutation operator is replaced by the application of LS2 over the offspring population in EA3 only. The best individual of this population has 20% chance of being executed; the second best individual has a 19% chance and so on. After this operator, we have two populations: the parents one and the offspring one, but only the best individuals will be kept to the next generation.

This natural selection might guide the EA to a premature convergence state. EA3 was provided with a mechanism that tries to avoid this undesirable situation. This mechanism is triggered if there are k (in tests, $k = 4$) successive generations without any improvement in the best solution found. At first, the whole population is discarded and LS3 is applied to best individual found so far, as many times as needed, in order to generate another population. The idea is to intensify the searching operations around the best solution. This first attempt is called intensification phase. If the best solution is not improved during other k successive generations all the individuals are removed and a whole new population is generated by ADDR, like in the initial population. This last procedure is applied only if the intensification does not work. We believe that an attempt of a radical diversification can work better than maintaining the stagnant population. This second attempt is called diversification phase. As before, if the best solution is not improved in more k generations the mechanism returns to the first phase.

4.1 The hybrid algorithm

We also propose a hybrid algorithm that uses both heuristics and exact methods. The objective is to get a very good partial solution and use it along with the evolutionary algorithm EA3 in order to improve it. We will reduce the planning horizon to its first half, solve this subproblem using the proposed formulation with CPLEX optimiser and give this partial solution to EA3. The metaheuristic will create some individuals, using the partial solution as fixed gens. The other individual gens are provided by the EA according to the ADDR behaviour. As the partial solution provided by CPLEX may be very good, it is expected that individuals based on it should be better too. In order to

reduce the planning horizon, we have to insert this set of constraints into the mathematical formulation.

$$\sum_{t=PT+1}^{H} x_{it} = 0 \quad \forall i = 1,\ldots,n \tag{9}$$

Constraints (9) ensure that the activations must be done until the time unit partial time (PT), where PT is in [1; $H - 1$], because after this time no more activations are allowed. The PT value was equal to $\lceil H/2 \rceil$ for all executions, because this is a good splitting point for instances with a large amount of tasks (more than 1,000 tasks).

5 Computational results

All tests were performed in an Intel Pentium D with 2 GB of RAM, running Linux (Ubuntu 8.10) and CPLEX 10. Heuristic were coded in C. Instances have been retrieved from LABIC project (http://www.ic.uff.br/~labic) and all have non-trivial solutions. Graphs are composed of 10% of tasks without precedence, other tasks have up to 5 predecessors. Costs and profits are randomly chosen in range [1; 50] and [1; 10], respectively, in order to produce interesting instances. Higher costs would make only few tasks to be activated; lower values would provide very easy instances where all tasks could be activated. Planning horizon size was defined according to n (number of tasks) as square root of n (for instances with $n \leq 1{,}000$) or cube root of n (for larger instances). The initial amount of resources Q_0 is enough to activate at least one task without precedence. P_0 is always equal to zero. Instances from other project scheduling problems are not suitable for the DRCPSP because they do not have profit values. Introducing artificial values in those instances and using them to compare the proposed formulation are not the objective of this paper. Other scheduling algorithms would also need drastic modifications in order to be used with this proposed model.

The instance hardness is proportional to the number of tasks and the planning process size (number of time units). The instances tested by the hybrid version algorithm (CPLEX + EA3) have low or medium hardness – they have either few tasks or few time units – because CPLEX cannot solve the hardest instances in acceptable computational time. However, in the tests with only EAs, the hardest instances are used. In fact, an instance with 1,000 tasks is harder than a task with 2,000 because the former has $H = 32$ (square root) while the latter has $H = 13$ (cube root). For exact algorithms, the size of H is main factor to be taken into account and it is directly related to the time spent to solve these instances.

The first test concerns in to find the optimal solution value in three sets of small instances with 50, 100 and 150 tasks (each set has 50 instances and each one was tested 30 times). In Table 1, we can see that EA3 found more optimal solutions and took almost the same computational time. We need to take into account that EA3 has more features than others versions, so the difference in time was already expected. CPLEX was very fast in solving these easy instances.

Table 1 Number of optimal solutions found and time spent (secs.) in small instances – time limit of 400 seconds for each instance

Optimals found (% of attempts)				*Average time spent*			
Size	*EA1*	*EA2*	*EA3*	*CPLEX*	*EA1*	*EA2*	*EA3*
50	9.9%	*95.5%*	93.2%	0.01	0.36	0.22	0.16
100	7.8%	38.2%	*51.0%*	0.26	1.96	1.73	1.89
150	0.1%	15.3%	*23.7%*	3.38	4.92	5.48	5.87

In fact, EA1 is the worst algorithm because the ADDR without the ETs generates very poor solutions. On average, the solutions generated by ADDR in EA2 and EA3 are almost three times better than solutions generated by ADDR in EA1. Table 2 shows a comparison between EA2 and EA3 using other instances. Second, fourth, sixth and eighth columns present the average improvement (of 30 runs) obtained by EA3 compared to EA2. The results obtained by EA2 are not as good as the EA3 results. In almost all instances, but mainly in instances where there are so many time units and many tasks (700 to 1,000 tasks), the mechanism to avoid premature convergence was more effective because to combine two large solutions generating good results is very difficult.

Table 2 EA3 results compared to EA2

Size	*Avg. improv.*	*Size*	*Avg. improv.*	*Size*	*Avg. improv.*	*Size*	*Avg. improv.*
100	0.0%	600	0.9%	1,100	0.2%	1,600	0.5%
200	1.7%	700	10.5%	1,200	0.1%	1,700	0.5%
300	1.3%	800	3.4%	1,300	0.4%	1,800	0.8%
400	5.8%	900	5.7%	1,400	0.7%	1,900	0.3%
500	0.7%	1,000	0.9%	1,500	–0.2%	2,000	0.5%

Another test involves the hybrid algorithm composed of CPLEX and EA3. The first half of the planning horizon is solved by CPLEX using all its default parameters. EA3, however, has its parameter ranges changed. A modified ADDR begins using a solution provided by CPLEX and only activates tasks after *PT* time units. Thus, a cutting time lesser than 0.5 does not make sense. The α values may be greater than original ones, because a good partial solution is already found and providing more greedy behaviour to ADDR may be more positive (new range varies from 0.5 to 0.8). Other parameters values are kept. In the diversification algorithm the ADDR uses the partial solution to construct other population, but in intensification algorithm EA3 will choose a time unit earlier than *PT* in order to reconstruct solutions from the best individual.

The instances used with hybrid algorithm have more than 1,000 tasks. These instances were chosen because they have a large number of tasks, but do not have a large number of time units ($H = \sqrt[3]{n}$, for them). Table 3 is composed of four columns: the first one shows the size of instance; the second one shows the average results (of 30 runs) using only the EA3 for the whole scheduling; the third one shows the average results using the hybrid method; the last one shows how much the average results from hybrid algorithm are better than EA3 itself.

Table 3 Average EA3 results compared to CPLEX + EA3

Size	*EA3 avg.*	*Hybrid avg.*	*Improv.*	*Size*	*EA3 avg.*	*Hybrid avg.*	*Improv.*
1,100	2,461	2,498	1.1%	1,600	3,287	3,341	1.4%
1,200	2,968	3,147	6.1%	1,700	4,445	4,631	4.6%
1,300	2,730	2,867	4.8%	1,800	3,877	4,082	5.0%
1,400	2,388	2,517	5.8%	1,900	3,911	4,019	2.6%
1,500	3,704	3,886	3.9%	2,000	5,563	5,914	6.1%

On average, the hybrid algorithm CPLEX + EA3 obtained results 4.2% better than the results generated by EA3. These average results are very good because they illustrate that both CPLEX and EA3 can collaboratively run their half scheduling. EA3 improved the partial results generated by CPLEX about 11%, on average.

These results show that this CPLEX + EA combination is a very promising technique. Other optimal solutions from other instances are still needed, in order to verify the average result of this hybrid algorithm in a larger set of instances. However, good improvements can be seen in almost all of the tested instances.

6 Concluding remarks

This work presented a new problem: DRCPSP – dynamic resource constrained project scheduling problem. It makes use of an uncommon kind of resource called dynamic resource that is quite different from Renewable Resources used in many other scheduling problems. For the DRCPSP, a mathematical formulation, a randomised constructive algorithm, four ETs, three local searches, three versions of evolutionary algorithms and a hybrid version were proposed.

The ETs significantly improved the solutions generated by ADDR up to three times without so much computational time. As it seems, its use is essential in order to provide better solutions. EA1 obtained very poor results and its tests were discontinued. EA2 and EA3 presented better results, but they also presented different behaviours. EA2 has a serious problem: premature convergence. The BP algorithm consumes a lot of computation in order to test the solution feasibleness which slows down the EA2. In addition, the fitness of the offspring rarely was better than its parent fitness.

On the other hand, intensification/diversification mechanism reached its objective enabling EA3 to avoid premature convergence state. UL algorithm has an important contribution because it is faster than BP and generates better offspring more frequently.

This work also presented some results of a hybrid algorithm that is composed of an exact method (CPLEX) and a metaheuristic (EA3). This combination obtained good results in a set of large instances and it is very promising due to the fact that, in almost all tested instances, significant improvements were achieved. This integration of mixed integer programming methods and metaheuristics is very important because it makes good use of each paradigm in order to obtain near optimal solutions in acceptable computational time. Nevertheless, there are some hard instances where this partition (into two halves) is not enough for CPLEX to solve its half scheduling. In these instances, heuristic methods are more suitable to solve them.

Acknowledgements

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

References

Chen, P-H. and Shahandashti, S.M. (2009) 'Hybrid of genetic algorithm and simulated annealing for multiple project scheduling with multiple resource constraints', *Automation in Construction*, Vol. 18, No. 4, pp.434–443.

Damak, N., Jarboui, B., Siarry, P. and Loukil, T. (2009) 'Differential evolution for solving multi-mode resource-constrained project scheduling problems', *Computer and Operation Research*, Vol. 36, No. 9, pp.2653–2659.

Feo, T.A. and Resende, M.G.C. (1995) 'Greedy randomized adaptive search procedures', *Journal of Global Optimization*, Vol. 6, No. 1, pp.109–133.

Gonçalves, J.F., Mendes, J.J.M. and Resende, M.G.C. (2009) 'A random key based genetic algorithm for the resource constrained project scheduling problems', *Computer & Operation Research*, Vol. 36, No. 1, pp.92–109.

Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI.

Homberger, J. (2007) 'A multi-agent system for the decentralized resource-constrained multi-project scheduling problem', *International Transactions in Operational Research*, Vol. 14, No. 6, pp.565–589.

Poirriez, V., Yanev, N. and Andonov, R. (2009) 'A hybrid algorithm for the unbounded knapsack problem', *Discrete Optimization*, Vol. 6, No. 1, pp.110–114.

Santos, H.G., Ochi, L.S., Marinho, E.H. and Drummond, L. (2006) 'Combining an evolutionary algorithm with data mining to solve a vehicle routing problem', *Neurocomputing*, Vol. 70, No. 1, pp.70–77.