A dynamic resource constrained task scheduling problem[†]

André Renato Villela da Silva
Luis Satoru Ochi*
Instituto de Computação - Universidade Federal Fluminense, Niterói, Rio de Janeiro, Brasil

Abstract

The purpose of this paper is to provide a new model for the Resource-Constrained Task Scheduling Problem (RCTSP) defined in this work as the Dynamic Resource-Constrained Task Scheduling Problem (DRCTSP). In the proposed model, we gather resources (profits) accordingly with the moment in which each task is activated during the planning process. To solve the DRCTSP, we proposed a mathematical formulation; *enhancement techniques,* and *evolutionary heuristics.*

Keywords: metaheuristic, evolutionary algorithm, task scheduling problem

1. Introduction

The task scheduling problem, in its former case, is composed by a set of n tasks T ($|T| = n$) and a set of m processors P ($|P| = m$). The objective is to assign the tasks t_i on the processors, such that the total processing time (*makespan*) is minimized. The problem becomes more difficult to solve when the tasks have precedence among then, what generally occurs.

Another constraint commonly used is related to an entity called "resource". To activate a task t_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. There is a lot of modeling for the RCPSP that uses both makespan and the resources. A deeply reading can be made at [1][2][3][4][5][6][8]. In the proposed model, there is profit l_i, related to the task t_i that is added to the available resources after the task t_i activation (this profit will be available for each unit time from this moment until the end of the planning process). Due to its high complexity, this problem (and its derivations) belongs to the NP-Complete class of combinatorial optimization problems. In this way, the application of exact methods becomes limited, justifying the use of heuristic techniques. In this work, we propose a dynamic version of the RCTSP here defined as Dynamic Resource-Constrained Task Scheduling Problem (DRCTSP). For this problem are proposed: (i) enhancement techniques; (ii) a mathematical formulation; and (iii) heuristics using concepts of evolutionary algorithm. In the Section 2, we show more topics about the proposed model, the mathematical formulation and a small constructive sample. The Section 3 is dedicated to the heuristics and the enhancement techniques. The Section 4 shows computational results and Section 5, concluding remarks.

2. The Dynamic Resource-Constrained Task Scheduling Problem

The motivation to propose this dynamic version of the RCTSP (DRCTSP) comes from a computer game (*Civilization III, from FIRAXIS Games Inc.*), where the player must research

* Corresponding author E-mails: {avillela, satoru}@ic.uff.br

some technologies (activate tasks) to help your civilization development. A DRCTSP consists of a DAG (directed acyclic graph) G = (V, A), where V is the set of vertexes (tasks) and A is the set of arcs (precedence among the tasks). Associated to each task t_i there is a cost c_i and a profit l_i (positive integer values). There is too a planning process (time interval composed by H time units).

The objective of the DRCTSP is maximizing the available resources at the end of the planning process. This model has potential application on manufacture expansion projects, where the tasks are expansion steps that can be made separately. Some concepts of the DRCTSP model: *Activation:* it is the entry of a task t_i in the current partial solution. To do this, we need to pay a cost c_i. After the activation, a profit l_i is available to us at each time unit. *Available task:* a task t_i is available, if all its predecessor tasks are activated. A task without precedence is available too. *Planning Process:* it is a set of time units, [1..H], when the tasks can be activated. *Cost (c_i):* it is the amount of resources necessary to activate a task t_i. *Profit (l_i):* it is the amount of resources given by a task t_i, for each time unit, from its activation until the final time unit H. *Total profit (S_i):* it is the profits sum of a task t_i. *Available Resources (Q_t):* it is the amount of resources that can be used to activate tasks, at time unit t. *Time Profit (L_t):* it is the sum of all profits that will be returned at time unit t.

2.1 Mathematical Formulation

We describe the DRCTSP as an integer programming problem. The x_{it} binary variable sets if a task t_i is activated (= 1) at time unit t or not (= 0). The Q_t integer variable defines the amount of available resources at time t. The L_t Integer variable defines the profit at time t. Q_0 is a problem input data and $L_0 = 0$. The P(i) represents the set of predecessor tasks of task t_i. Following, in the Figure 1 is the proposed formulation. The line 2.1 describes the objective function (maximize $Q_H + L_H$), where H is the last time unit. The 2.2 constraints ensure that a task t_i only will be activated at time 1, if it doesn't have any precedence ($|P(i)| = 0$). The 2.3 constraints ensure that a task t_i only will be activated at time t, if all its predecessor tasks are activated, at least t-1 time. The 2.4 constraints guarantee that the sum of task costs activated at time t will be less or equal than the available resources in this time. In the 2.5 constraints are defined how the available resources are changing along the time. Similarly, in the 2.6 constraints are defined the time profit increments. In the 2.7 constraints is ensured that a task t_i will be activated once. Finally, the last two constraints define the variables of the problem.

2.2 A simple constructive sample

To demonstrate these concepts, we present now a small sample of solution construction. Here $Q_0 = 2$ (input data) and $L_0 = 0$. At every task, there are two numbers above then: the first one is its cost and second one is its profit for each time unit. Activated tasks are in white, available in gray and unavailable in black. In the Fig 2 (a), at time unit $t = 1$, there are two available tasks (1 and 2). Let's activate task 1. We need to calculate the $Q_1 = 1$ and $L_1 = 2$, and the available tasks (now, 2 and 3). Let's activate both. In the Fig 2 (b), at time unit $t = 2$, we calculate $Q_2 = 0$ and $L_2 = 7$ and update the task 4 state. Finally, at Fig 2 (c) we activate task 4 and calculate $Q_3 = 5$ and $L_3 = 9$. Then, the final solution value is given by $Q_3 + L_3 = 14$.

3 – Evolutionary Heuristics

To solve the DRCTSP, was initially proposed one randomized constructive heuristic ADDR. At each time unit t, it makes a list of available tasks and orders it accordingly the $\{c_i/l_i\}$ of the tasks. Then a subset of these tasks (the p% best tasks using a α parameter as used in GRASP algorithms [9] is selected). Among these tasks, one is randomly selected. If its cost is less or equal the available

resources, this tasks is activated and Q_t and L_t updated. New random selections are made until there aren't any available tasks or available resources in this time t. The tasks states are updated, Q_{t+1} and L_{t+1} are calculated and we pass to the next time until the final time H.

Max:

$$Q_H + L_H \tag{2.1}$$

S.T.

$$|P(i)|x_{i1} = 0 \quad \forall i = 1, ..., n \tag{2.2}$$

$$|P(i)|x_{it} \leq \sum_{j \in P(i)} \sum_{t'=1}^{t-1} x_{jt'} \quad \forall i = 1, ..., n \quad \forall t = 2, ..., H \tag{2.3}$$

$$\sum_{i=1}^{n} c_i x_{it} \leq Q_t \quad \forall t = 1, ..., H \tag{2.4}$$

$$Q_t = Q_{t-1} - \sum_{i=1}^{n} c_i x_{it} + L_{t-1} \quad \forall t = 1, ..., H \tag{2.5}$$

$$L_t = L_{t-1} + \sum_{i=1}^{n} l_i x_{it} \quad \forall t = 1, ..., H \tag{2.6}$$

$$\sum_{t=1}^{H} x_{it} \leq 1 \quad \forall i = 1, ..., n \tag{2.7}$$

$$x_{it} \in \{0, 1\} \quad \forall i = 1, ..., n \quad \forall t = 1, ..., H \tag{2.8}$$

$$Q_t, L_t \in \mathbb{N} \quad \forall t = 0, ..., H \tag{2.9}$$

Figure1: The mathematical formulation of the DRCTSP

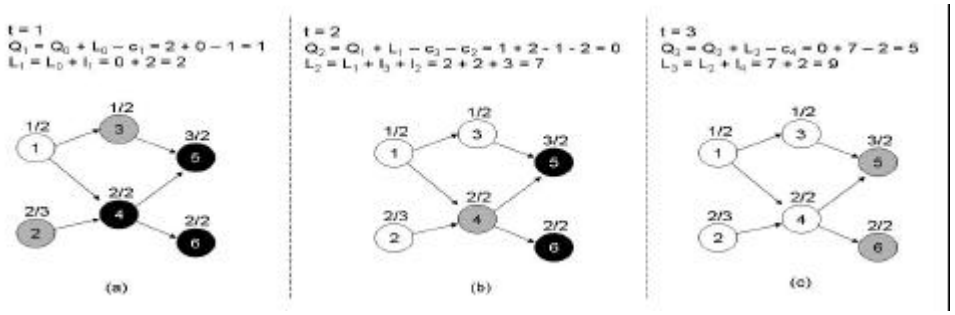

Figure 2: A sample of constructive algorithm

3.1 Enhancement Techniques (ET)

This works also presents a set of enhancement techniques to try to improve the solution quality generated by the propose algorithms. Two of these techniques try to reduce the feasible solution space, creating constraints to the tasks activation. The third ET does a pre-processing based on an optimal scheduling foresight. And the last one does a pre-processing to try to reduce the computational time. They are in following.

ET1: Cutting Time (CT). Its idea is simple: from some planning time on, a task only will be activated if its *Total Profit* (S_i) > c_i. In other words, only if a task generates more resources than its cost we can activate it. It is proposed not to spend resources with a task "*poorly profitable*". The CT must be in [1, H] of the planning process. If CT = 1, we will ever use it, if CT = H we will never use it.

ET2: Relaxation Margin (RM). It is proposed to correct an abuse made by CT. A non-profitable task t_i can be activated due a possible future profitable successor task of it. To do this, we will multiply the S_i by a RM factor (near 1.0 to up). Then, we give a chance to task t_i to be activated even its "*local non-profitable state*". However, we cannot ensure that this successor task will be activated, because it will depend on others situations.

ET3: Previous Weighting (PW). In this case, we will not use the $\{c_i/l_i\}$ to order the available tasks. We will use the $\{c_i/EP_i.\}$ where EP_i is the *Expected Profit of task* t_i. It is calculated (once) based on the early time that a task can be activated (looking only to the graph topology). Once discovered this early time, we multiply it by the profit l_i to obtain the EP_i.

ET4: Arc Removing (AR). It is a reduction rule and doesn't noise the solution quality, but reduces the processing time, by removing some redundant arcs of the input graph. An arc *(r,s)* can be removed if, after this, we can find a path K from r to s in the resulting graph. So the arc *(r,s)* is said an *explicit precedence* of an *implicit precedence* (the path K), and can be removed. Some preliminary tests prove that in the most cases it reduces the processing time, mainly in instances with a very large number of arcs.

3.2 Local Search

Local search (LS) is an improvement procedure that tries to modify an initial solution by changing small segments of it, called neighborhood. Two local search algorithms are proposed: the first one (LS1) analyzes on each task t_i by time. It does the same calculation made by CT. If a task t_i has $c_i > S_i$ it is removed from the current solution. The analysis begins with the tasks activated lately, coming to the task activated early. The second LS (LS2) is a generalization of the LS1 and works on a set of all successors of a task t_i (F_i) by time. If the total cost of the set F_i is more than the total profit generated by F_i, the whole set F_i is removed from the current solution.

3.3 Evolutionary Algorithms

Evolutionary Algorithms (EA) and the Genetic Algorithms (GA), its most popular representative, are heuristics that works simultaneously with a population of solutions, trying to combine them to generate better solutions. Basically we need to generate a population of feasible solutions called initial population. Then, a population subset is chosen to combine and generate a new population. From these two populations, a subset of solutions is chosen to form the new generation of EA. Then, new solutions are chosen to combine and generate a new population and so on. The EA stops when a number of generations are achieved. Readings on EA can be made at [2][5][7][8][10][11]. The EAs have been used in several areas to solve problems considered intractable (NP-Complete and NP-Hard) although its basic versions do not demonstrate too much efficiency in the high complexity problems [10][11]. In order to improve the GA and EA performance researchers have proposed hybrid versions [5][8][10][11].

Two EA (EA1 and EA2) are proposed to solve the DRCTSP. The EA1 is a basic version and the EA2 is a hybrid version including all the enhancement techniques (ET) and the local searches proposed. The Table 1 shows the differences between them. The column *Size* indicates the size of the populations. *Gens* is the maximum number of generations. *Init. LS* shows the Local Searches used after the initial population generation. *Pop. LS* is the local searches used after the generation of a new population. The last column indicates the enhancement techniques used.

EA	Size	Gens	Init. LS	Pop. LS	ETs
EA1	30	50	-	-	AR
EA2	30	50	LS1, LS2	LS1	AR, CT, RM, PP

Table 1: EAs structures

Both EAs generates the initial population using the ADDR heuristic. The α parameter is very important to obtain good solutions. We defined a α range of values ([*0,05 ; 1,0*], with 0,05 increments) within one value is chosen. For each value of α, the ADDR runs *20* times. The chosen α is the value that generates the best solution from them all. Similarly, the CT value ([*0,2 ; 1,0*] with *0,1* increments), the RM value ([*1,0 ; 1,4*] with *0,1* increments) and the PW value ("*use*" or "*don't use*") are chosen. To choose the solutions that will combine, the population is partitioned in three parts. The "*PA*" class is composed by the *20%* best solutions. The "*PC*" class by the *20%* worst solutions. The "*PB*" class by the others solutions. Then to select two parents, a solution from *PA* is chosen randomly and combined with a randomly chosen solution fom *PB*. Thirty new solutions (*offsprings*) are generated for each iteration. From these *60* solutions (*30+30*) only the *30* best

different solutions are chosen to be in the next generation. To combine the parent solutions a heuristic is proposed too. It begins with the tasks without precedence. Let's call the parent solutions S1 and S2. A task t_i is activated in S1 at time 2 and in S2 at time 3. In the new solution t_i will be activated at the earliest time, in this case time 2. If a task can not be activated at the earliest time (because one or more constraint violations), the latest time will be chosen. If even thus the task cannot be activated, then it will not be in the new solution. The algorithm proceeds with the new available tasks (from S1 and S2) until all tasks were analyzed.

4. Computational Results

All tests were done in a 3.0 GHz Pentium 4 HT, with 256 MB and C code over Windows XP. As the DRCTSP is a new modeling, there aren't any benchmark instances to use. Then, two classes of instances were generated to evaluate the proposed enhancement techniques and the evolutionary heuristics. The *A* class is formed by a *10%* of tasks without precedence and the other part with from *1* to *5* predecessors chosen randomly. In the *B* class, the first task hasn't got any precedence. From the second one, there is a 20 % chance of having precedence with each previous task. In both classes, the cost c_i is chosen from *1* to *50* and the profit L_i from *1* to *10*, randomly. The time interval (planning process) is the square root of the number of tasks, if it is less or equal *1000*. If the task number is more than *1000*, the interval is the cube root. The first test was done with very small instances (*50* to *150* tasks). It was possible to find the optimal solution with these instances. Then we created *50* instances for each size, found the optimal solution using the mathematical formulation proposed and the GLPK software and ran the EAs, setting the associated optimal value as target value (stop criteria for EAs). The Table 2 shows how many times each EA found the optimal value. And Table 3 shows the "*average distance*" to the optimal value, when its value was not reached. Moreover, in some instances with size of 150 tasks, to reach the optimal value with an exact algorithm spends *100* times more processing time than an EA.

	A - Instances		B – Instances	
Size	EA1	EA2	EA1	EA2
50	5	48	0	43
100	4	20	0	26
150	0	8	2	28

Table 2: Number of optimal solution found in 50 runs

	A - Instances		B – Instances	
Size	EA1	EA2	EA1	EA2
50	51,0%	14,9%	45,8%	8,1%
100	53,0%	6,8%	51,6%	5,6%
150	54,7%	9,6%	47,5%	6,7%

Table 3: Average distance to optimal value

In the second test done, the EAs ran three times for each instance, initially with the original stop criteria (number of generations), and the average value is calculated. The Table 4 shows how much EA2 outperforms EA1. An example: a value of 100% indicates that EA2 generates a solution twice better than EA1. The instances were partitioned in class (small: <= 500 tasks - 5 instances; medium: > 500 and <= 1000 tasks - 5 instances; large >1000 tasks – 10 instances) in both types *A* and *B*. The small difference showed in the medium instances is due the large number of activated tasks in the solutions. It allows the ADDR without ETs to generate solutions with good quality too, because what only matters is WHEN the tasks will be activated, not WHAT tasks will be activated. The Table 5 shows a similar test, but now, the EAs have a time limit to run. When time is up, we verify the quality of the solution found. Again, the value in the table cells indicates how better EA2 is compared to EA1.

	A - Instances	B - Instances
Small	95,6%	138,9%
Medium	13,7%	14,5%

	A - Instances	B – Instances
Small	101,0%	278,3%
Medium	16,6%	8,9%

Large	145,8%	112,2%

Table 4: Average outperform from EA2 to EA1 without time limit

Large	149,1%	120,3%

Table 5: Average outperform from EA2 to EA1 with time limit

	EA1: A – Instances	EA2: A – Instances	EA1: A – Instances	EA2: A – Instances
Small	0,0%	100,0%	100,0%	100,0%
Medium	0,0%	89,0%	55,6%	100,0%
Large	0,0%	100,0%	73,3%	100,0%

Table 6: Percentage of reached target values

In the last test (Table 6), a target value was defined for each instance used as stop criteria for the EA1 and EA2. This target value is the EA1 result (average of the three runs) obtained in the first test. The cell values indicate the percentage of times that the target was reached. The very bad results obtained in table 6 by EA1 with *A-instances* (compared to the *B-instances)* is due the fact that *A-instances* have less constraints than *B*, once *A-instances* have less precedence among the tasks than *B*). Thus, the space solution of *A-instances* is bigger than *B*, what does *A-instances* more difficult to get an optimal solution.

5. Concluding Remarks

This paper presented a new model of Resource-Constrained Task Scheduling Problem (RCTSP) called DRCTSP. To solve this problem are presented: a mathematical formulation; enhancement techniques-ET and evolutionary heuristics. Initially the tables 2 and 3 show that EA2 using all enhancement techniques (ET) and the two local searches proposed significantly improve the average performance of the standard version of EA (EA1). Additionally, we did a set of computational tests with large scale instances, where the objective was to evaluate the behavior of the hybrid version (EA2) including the ETs and LSs procedures. The average results of the Tables 4, 5 and 6 shows that EA2 also clearly outperforms the EA1 in the large instances.

6. References

[1] Remy, J. Resource constrained scheduling on multiple machines. Information Proc. Letters, v. 91, p. 177–182, 2004.

[2] Boeres, C.; Rios, E.; Ochi, L. S. Hybrid Evolutionary Static Scheduling for Heterogeneous Systems. Proc. of the IEEE Conf. on Evolutionary Computation (CEC2005), Book 3, pp. 1929-1937, Edinburgh, 2005.

[3] Kalinowski, T.; Kort, I.; Trystam, D. List scheduling of general task graphs under LogP. Parallel Computing, 26, p. 1109–1128, 2000.

[4] Bouleimen, K.; Lecoco, H. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. European Journal of Operational Research, v. 149, p. 268–281, 2003.

[5] Gonçalves, J. F.; Mendes, J. J. M.; Resende, M. G. C. A genetic algorithm for the resource constrained multi-project scheduling problem. European Journal of Operational Research (submitted), 2005.

[6] Brucker, P. et al. Resource-constrained project scheduling: Notation, classification, models, and methods. EJOR, v. 12, p. 3–41, 1999.

[7] Holland, J. H. Adaptation in Natural and Artificial Systems. Ann Arbor, MI: [s.n.], 1975.

[8] Gonçalves, J.; Mendes, J.; Resende, M.G.C. A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research, v. 167, p. 77–95, 2005.

[9] Resende, M.G.C.; Ribeiro, C.C. Greedy randomized adaptive search procedures (GRASP). In: Handbook of Metaheuristics [edited by F. Glover and G. Kochenberger], Kluwer Academic Publishers, p. 219-249, 2002.

[10] Santos, H. G.; Ochi, L. S.; Marinho, E. H.; Drummond, L. Combining an Evolutionary Algorithm with Data Mining to solve a Vehicle Routing Problem. To appear in Neurocomputing Journal - ELSEVIER, 2006.

[11] Trindade, A. R.; Ochi, L. S. Um algoritmo evolutivo híbrido para a formação de células de manufatura em sistemas de produção. To appear in Pesquisa Operacional, SOBRAPO, 2006.

[†] Partly supported by FAPERJ: E-26/ 150.575/2004