Aceito para o International Network Optimization Conference (INOC 2007), April 2007, Spa, Belgium.

Effective GRASP for the Dynamic Resource-Constrained Task Scheduling Problem

André Renato Villela da Silva
Luiz Satoru Ochi
*e- mail : {*avillela@ic.uff.br, satoru@ic.uff.br}

Keywords: *meta-heuristic, GRASP, task scheduling problem*

Introduction

The Task Scheduling Problem is subject of many studies in Network Optimization due to its applicability in many areas of engineering and computation. There are many models of environments (homogeneous/heterogeneous), constraints (communications costs, use of resources [i]) and many objectives (reduction of makespan/processor idleness) that can be applied to simulate a real problem. This work analyzes a new modeling that differs from previous ones, because the tasks have a new entity: a profit. After the activation of a task, it generates an amount of resources that must be used to activate other tasks. Therefore, the resource handling is very dynamic because it is not possible (*a priori)* to define how many resources will be available at each time. This problem, proposed in [iii], is called *Dynamic Resource-Constrained Task Scheduling Problem (DRCTSP).*

DRCTSP modeling

Given 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 *i* there is a cost c_i and a profit l_i (positive integer values). There is also a planning process (time interval composed of *H* time units).

The objective of the DRCTSP is to maximize the available resources at the end of the planning process. This model has potential application to 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 *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 until time *H*. *Available task:* a task *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. *Total profit (S_i):* it is the profit sum of a task *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*.

Mathematical Formulation

We describe the DRCTSP as an integer programming problem. The x_{it} binary variable sets the activation of a task *i* at time *t* (= 1) 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 i. The proposed formulation is in following. Line (1) describes the objective function (maximize Q_H+L_H), where *H* is the last time unit. In (2) the

constraints ensure that a task *i* will only be activated at time *t* if all its predecessor tasks are activated, at least *t - 1* time. In (3) the constraints guarantee that the sum of task costs activated at time *t* will be lesser than or equal to the available resources in this time. In (4) the constraints are defined according to the way the available resources change through time. Similarly, in (5) the constraints are defined by the time profit increments. In (6) the constraints ensure that a task *i* will be activated only once, at maximum. Finally, the last two constraints define the variables of the problem.

$$\text{Max}$$

$$Q_H + L_H \tag{1}$$

$$\text{S.t.}$$

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

$$\sum_{i=1}^{n} c_i x_{it} \leq Q_t \quad \forall t = 1,...,H \tag{3}$$

$$Q_t = Q_{t-1} + L_{t-1} - \sum_{i=1}^{n} c_i x_{it} \quad \forall t = 1,...,H \tag{4}$$

$$L_t = L_{t-1} + \sum_{i=1}^{n} l_i x_{it} \quad \forall t = 1,...,H \tag{5}$$

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

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

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

Figure 1: Mathematical formulation of the DRCTSP

A simple constructive example

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

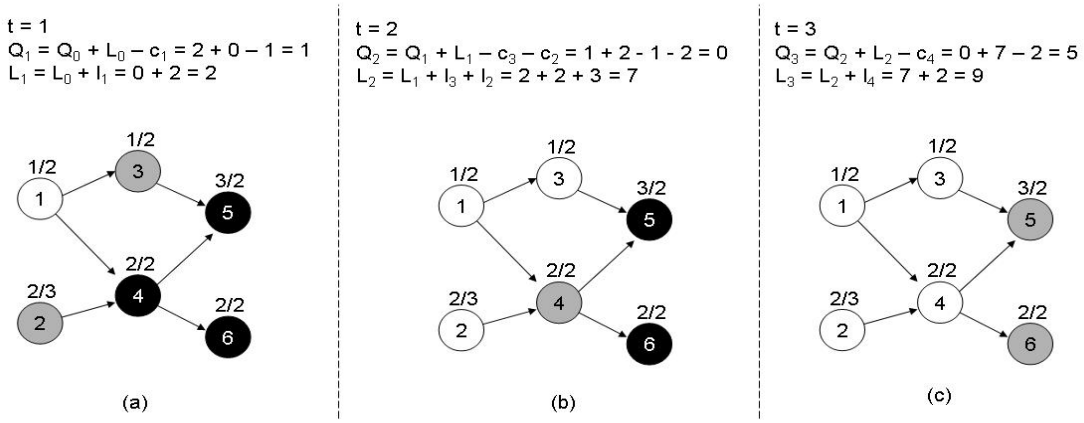

Figure 2: Construction of a simple solution

Basic algorithms to solve DRCTSP

To solve the DRCTSP, one randomized constructive heuristic ADDR was initially proposed. At each time unit *t*, it makes a list of available tasks and sorts it accordingly by using the $\{c_i/l_i\}$ of the tasks. Then, a subset of these tasks (the *p%* of the best candidate tasks using a α parameter as used in GRASP algorithms [ii]) is selected. Among these tasks, one is randomly selected. If its cost is lesser than or equal to the available resources, this task is activated and Q_t and L_t is updated. New random choices are made until there are no available tasks or available resources in this time *t*. The task states are updated, Q_{t+1} and L_{t+1} are computed and we pass to the next time until the final time, *H*.

Enhancement Techniques (ET)

We proposed a set of the enhancement techniques that are methods to try to improve the quality of the solution generated by the ADDR. Two of these techniques try to reduce the feasible solution space, creating constraints on the task activation. The third ET involves pre-processing, based on an optimal scheduling foresight. The last one involves pre-processing to try to reduce the computational time. They are explained in the following.

Cutting Time (*CT*). Its idea is simple: from some planning time on, a task will only be activated if its *Total Profit* (S_i) > c_i. In other words, only if a task generates more resources than its cost can we activate it. It is proposed to do not spend resources on poorly profitable tasks. The *CT* value must be in [1; *H*] of the planning process. If *CT = 1*, we will always use it; if *CT = H*, we will never use it. CT value also can be in uniform range [0; 1].

Relaxation Margin (*RM*). Even so, there are cases in which the activation of a non-profitable task *i* may be a good choice, if any successor task is very profitable. A simple way to allow the activation of this task *i* is to multiply the S_i by a *RM* factor (more than 1.0). However, it does not ensure the activation of the task *i* successors because there are yet resource constraints in future time units.

Previous Weighting (*PW*). In this case, we will not use the $\{c_i/l_i\}$ to sort the available tasks. We will use the $\{c_i/EP_i\}$, where EP_i is the *Expected Profit* of task *i*. It is computed (once) based on the earliest time that a task can be activated (looking only to

the DAG topology). When this earliest time is discovered, we multiply it by the profit l_i to obtain the EP_i.

Arc Removing (*AR*). It is a reduction rule and does not modify 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 path *K* from *r* to *s* in the resulting graph. So the arc *(r,s)* is said to be an *explicit precedence* of an *implicit precedence* (path *K*), and can be removed. Some preliminary tests prove that in most cases it reduces the processing time, mainly in instances where there are a very large number of arcs.

Local Search

Local search (LS) is an improvement procedure that tries to modify an initial solution by changing small segments of it, called neighborhoods. Three local search algorithms were tested: the first one (LS1) analyzes tasks *i* by time. It does the same computation done by *CT*. If task *i* has $c_i > S_i$ it is removed from the current solution. The LS1 analysis begins with the tasks activated later, then back to the tasks activated earlier. The second LS (LS2) is a generalization of the LS1 and works on a set of all successors of a task *i* (F_i) by time. If the total cost of set F_i is greater than the total profit generated by F_i, the whole set F_i is removed from the current solution. The last one (LS3) tries to reconstruct the current solution with a less greedy criteria than ADDR. LS3 randomly chooses a time unit *TL* in the range [*H/4*, *H/2*]. The tasks activated after *TL* are removed and new tasks are activated according to the roulette technique used in genetic algorithms, where a task which is more profitable has a better chance of being chosen, for example. Of course, only available tasks may be chosen at each time unit.

Heuristics

Two versions of the GRASP meta-heuristic (GR1 and GR2) were proposed to solve the DRCTSP. Each GRASP version uses ADDR to generate the solution in the construction phase. In the local search phase, GR1 uses only LS1 while GR2 uses all the proposed local searches. The parameters used in the ETs must be calibrated in order to help ADDR generate improved solutions. This calibration, in each instance, is done by testing some predefined values: 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} (yes or no). The value associated to the best solution is chosen to be used in the iterations of the meta-heuristic. In [iii], two Evolutionary Algorithms (EAs) using very similar techniques were proposed. Those algorithms will be compared to the GRASPs proposed here.

Instances

Two classes (A and B) of instances were used in [iii]. Class *A* is composed of 10% of the tasks without any precedence. The others tasks have from 1 to 5 predecessors randomly chosen. In class *B*, the first task has no precedence. For the others, there is a 20 % chance of having precedence with each previous task. In both classes, the cost c_i is randomly chosen from *1* to *50* and the profit l_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 (planning

process) is the square root of the number of tasks, if it is lesser than or equal to *1000*. If the task number is more than *1000*, the interval is the cube root.

Computational Results

The main objective of these tests is to compare the GRASP versions GR1 and GR2 to the Evolutionary Algorithms EA1 and EA2 previously proposed in [iii]. Hence, the computational environment and the test structures used for EAs and GRASPs are the same. However, distinct seeds were used in the runs of algorithms to ensure their random aspect.

The optimal solution of some small instances can be found easily by the GLPK or CPLEX solver, using the proposed formulation. These optimal values can be used as target values for these instances to test the GRASPs empirical convergence. The first test consists of running GR1 and GR2 over 50 instances made up of 50 tasks, 100 tasks and 150 tasks. Table 1 shows how many times the algorithms found these target values (optimal values).

Class A Instances					Class B Instances				
Size	EA1	EA2	GR1	GR2	Size	EA1	EA2	GR1	GR2
50 tasks	5	**48**	41	47	50 tasks	0	**43**	42	**43**
100 tasks	4	20	21	**26**	100 tasks	0	26	23	**28**
150 tasks	0	8	6	**9**	150 tasks	2	**28**	15	19
Total	9/150	76/150	68/150	82/150	Total	2/150	97/150	80/150	90/150

Table 1: Optimal values found for very small instances

In these very small instances the GRASP versions obtain results very similar to the EAs, but a bit worse. A GR2 Basic version (GR2B), without the enhancement techniques (ETs), was tested too. It obtained, on average, fewer optimal values than GR2. In cases where the optimal value was not found, the average distance to it is 15%, at maximum, for GRASPs and EA2; and 50% for EA1. The main reason for this result is the poor solution quality generated by the combination algorithm used in EA1. EA2 uses the same method, but the ETs get some improvements that are not possible without them.

The computational time spent by GR2 and GR2B is about twice or three times the GR1 computational time. This is due to the LS3 (recreation local search) that spends about one third of the total iteration time of the GRASP. LS2 also consumes a great part of this iteration time, while LS1 consumes less than 2% of the time. However, the improvement obtained by these LSs is not proportional to spent time. LS1 and LS2 contribute (each one) with less than 1% of the total solution quality, while LS3 can contribute up to 10%, in instances having a very high number of activated tasks.

It can be seen in Table 2. The table cells show how the GR2 is better than the other meta-heuristics (in percentage). The small class is composed of 5 instances with the size up to 500 tasks; the medium class is composed of 5 instances with the size between 500 and 1000 tasks; the large class is composed of 10 instances with more than 1000 tasks). All values were computed over the average values of three runs with distinct seeds.

The small and large classes have less activated tasks than medium class due to the instances characteristics. Once the LS3 reconstruction uses a lesser greedy criteria than ADDR, it can search the solution space in a wider manner where there are many activated

tasks. In instances where there are few activated tasks, this search cannot achieve the same results. In some cases (GR1 – large class), it was not necessary to use the reconstruction to achieve good results. In fact, the LS3 was not effective in these instances.

Compared to the GR2B, the GR2 results are better in all class A instances and in almost all class B instances. It is indicative that the ETs actually improve the solution quality, because even between the EAs, the ETs results are better. On average, GR2B was not much worse than GR2 because the LSs (mainly LS1 and LS2) do similar refinements like the ETs, but the LSs spend larger computational time.

Size (# Instances)	EA1	EA2	GR1	GR2B
Small (5)	44,1%	1,4%	1,0%	1,2%
Medium (5)	13,8%	4,0%	3,6%	1,0%
Large (10)	59,4%	0,6%	- 0,1%	1,6%
Average	39,1%	2,0%	1,5%	1,3%

Table 2: Comparison of GR2 results in class A instances

Others tests compare the evolution of the best solution in EAs and GRASP versions. The result is a better performance in GRASP versions due to the independent iterations: a poor solution will be discarded in the next iteration. It does not occur in EAs, where a poor solution (individual) may be constituent of the population by some generations. The other tests in [iii] were repeated with the GRASP versions and they achieved the same or better results than EA2.

Concluding Remarks

This work compared two meta-heuristic approaches intensively studied nowadays: Evolutionary Algorithms and GRASP. The GRASP versions proposed here provide better results than the previously proposed EAs results, in almost all tests. It is due to the GRASP structure that does not force the algorithm to work on a poor solution. The local searches must still be used with caution because, in some instances, they do not improve the solution quality proportionally to the computational time spent. Finally, the Enhancement Techniques (ETs) appear to be very useful in both meta-heuristics, once they allow the construction of better solutions and cause very little overhead in computational time.

References

[i] Brucker, P. et al. Resource-constrained project scheduling: Notation, classification, models and methods. European Journal of Operation Research, v. 12, p. 3-41, 1999.

[ii] 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.

[iii] Silva, A. R. V.; Ochi, L. S. A dynamic resource constrained task scheduling problem. In proceeding of XIII CLAIO, 2006.