A hybrid Evolutionary Algorithm for the Dynamic Resource Constrained Task Scheduling Problem

André Renato Villela da Silva, Luiz Satoru Ochi
Universidade Federal Fluminense
Niterói, RJ - Brasil
{avillela,satoru}@ic.uff.br

Abstract

This work presents a new hybrid Evolutionary Algorithm for the Dynamic Resource Constrained Task Scheduling Problem (DRCTSP). The most important differences between the new EA and the previously proposed EAs are an intensification/diversification mechanism that tries to avoid premature convergence in local optimal solutions and a version combining an exact method (CPLEX) with EAs. Some preliminary tests were done and results are very promising.

1. Introduction

Scheduling problems have been the subject of many studies during the last years due to its large application to many areas of computing and engineering. There are many models of environments (homogeneous/heterogeneous), constraints (communications costs, use of resources [1]) and objectives (reducing makespam/processor idleness) that can be applied to simulate a real problem. We propose a method merging (meta)heuristics and exact algorithms for the Dynamic Resource Constrained Task Scheduling Problem (DRCTSP). This is a new problem that can be applied to project management of companies, in which sub-projects are modeled as tasks. Each task i has a cost c_i that must be paid to activate it. After this activation, this task generates a positive profit l_i, at each time unit, that can be used to activate other tasks. The complete modeling can be found in the next section. This work presents the modeling of a new scheduling problem (DRCTSP), a constructive algorithm (ADDR) for the DRCTSP, some techniques that improve the ADDR solution, a new Evolutionary Algorithm and its hybrid version with an exact method (used by CPLEX solver). Some tests were done and interesting results are obtained. The following section presents the DRCTSP modeling. Some algorithms (constructive algorithm, local searches, Evolutionary Algorithm and its hybrid version) for the DRCTSP are presented in Section 3. Section 4 discusses the instances used in tests. Section 5 presents the obtained computation results. Finally, the conclusions and future works are described in Section 6.

2. The DRCTSP modeling

Given an 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 by H time units). The objective of the DRCTSP is to maximize the available resources at the end of the planning process.

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 at each time unit until the final time unit H. *Available task*: a task i is available, if all its predecessor tasks are activated. A task without precedence is available too. *Planning Process*: 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.

This modeling differs from other resource constrained scheduling problems because it has the entity called profit that generates resources to be used in other activations. Due to it this modeling is called dynamic, once that defining a priori the amount of available resources is not possible.

2.1. Mathematical Formulation

We describe the DRCTSP as an integer programming problem. The x_{it} binary variable sets if a task 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 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 $t = 1$ if it does not have any precedence ($|P(i)| = 0$). In (3) 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 (4) the constraints guarantee that the sum of task costs activated at time t will be lesser than or equal to the available resources at this time. In (5) the constraints are defined according to the way the available resources change through time. Similarly, in (6) the constraints are defined by the time profit increments. In (7) the constraints ensure that a task i will be activated once, at maximum. Finally, the last two constraints define the variables of the problem.

Max

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

S.t.

$$|P(i)|x_{i1} = 0 \quad \forall i = 1, ..., n \tag{2}$$

$$|P(i)|x_{it} \leq \sum_{j \in P(i)} \sum_{t'=1}^{t-1} x_{jt'}$$

$$\forall i = 1, ..., n \quad \forall t = 2, ..., H \tag{3}$$

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

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

$$L_t = L_{t-1} + \sum_{i=1}^{n} l_i x_{it} \quad \forall t = 1, ..., H \tag{6}$$

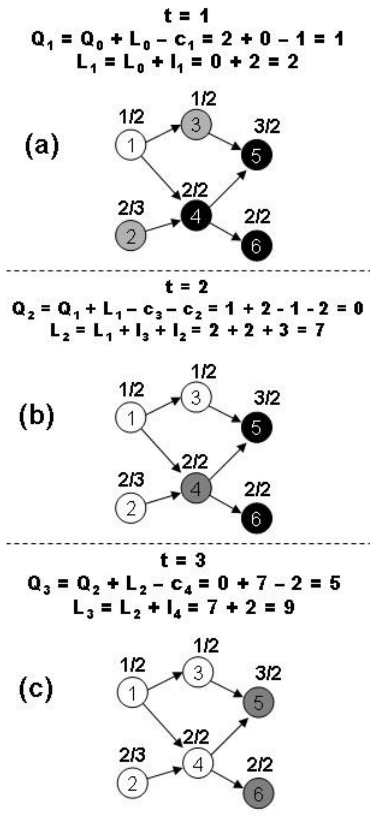
$$\sum_{t=1}^{H} x_{it} \leq 1 \quad \forall i = 1, ..., n \tag{7}$$

$$x_{it} \in \{0, 1\} \quad \forall i = 1, ..., n \quad \forall t = 1, ..., H \tag{8}$$

$$Q_t, L_t \in N \quad \forall t = 0, ..., H \tag{9}$$

2.2. Constructive sample

To demonstrate these concepts, a small sample of solution construction is as follows: $Q_0 = 2$ (input data) and $L_0 = 0$. In Figure 1 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 1 (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 Figure 1 (b), at time unit $t = 2$, we calculate $Q_2 = 0$ and $L_2 = 7$ and update the task 4 state. Finally, in Figure 1 (c) we activate task 4 and calculate $Q_3 = 5$ and $L_3 = 9$. Then, the final solution value is given by $Q_3 + L_3 = 14$.



t = 1
Q₁ = Q₀ + L₀ – c₁ = 2 + 0 – 1 = 1
L₁ = L₀ + l₁ = 0 + 2 = 2
(a)
1/2
1/2
3/2
2/2
2/3
2/2
t = 2
Q₂ = Q₁ + L₁ – c₃ – c₂ = 1 + 2 - 1 -2 = 0
L₂ = L₁ + l₃ + l₂ = 2 + 2 + 3 = 7
(b)
t = 3
Q₃ = Q₂ + L₂ – c₄ = 0 + 7 – 2 = 5
L₃ = L₂ + l₄ = 7 + 2 = 9
(c)
1
2
3
4
5
6

Figure 1. A small constructive sample

3. Algorithms to solve DRCTSP

In [8], a basic constructive algorithm called ADDR was proposed. It makes, at each time unit t, a list containing all available tasks (this list is sorted by the c_i/l_i of each task). Then, a subset of the $p\%$ best candidate task is defined, as the α parameter in GRASP [2]. From this restricted list, a task is randomly chosen. If its cost is lesser than or equal to the available resources at time t (Q_t) the task is activated. Q_t and L_t are updated and new selections are made until there are none available tasks or available resources in this time t. So, we move to the next time unit. At the end of planning process (time unit H), the algorithm outputs the solution and its value. The solution representation is composed by an array of n elements, where n is the number of tasks. Each position represents one task and the value represents the time when the task was activated. If a task wasn´t activated its position is marked as NULL (or zero value).

3.1. Enhancement Techniques

Probably the most important contribution present in [8], the Enhancement Techniques (ETs) try to reduce the number of tasks in candidate list, by adding some constraints to them. For example: a task i will only be activated if its Total Profit $S_i > c_i$. In other words, a task i will only be activated if it is profitable.

This criteria is used in ET1 (Cutting Time - CT) from some planning time (CT) on. However, before this CT time unit (absolute value in [1;H] or normalized value in [0;1]), any task can be activated even it causes resource losses. If CT = 1, we will aways use it; if CT = H, we will never use it.

Even so, there are cases in which the activation of a non-profitable task may be a good choice: if any successor task is very profitable. But supposing that a successor task will be activated is very difficult due to the problem constraints (mainly the available resources needed to activate it in a future time unit). Hence, a simple way to allow the activation of this non-profitable task i is to multiply its profit l_i by a factor and then to compare the product with the cost c_i, like in ET1. This computation is done by the ET2 (Relaxation Margin - RM).

The third ET (ET3) works like a preprocessing of the graph G. For each task i, the earliest time when it can be activated is computed. This time is multiplied by the profit l_i and the product is called Expected Profit of the task i (EP_i). This value is the largest amount of resources that can be generated by the task i in the best case. Therefore, the sorting criteria for the available tasks is now the c_i/EP_i instead of former c_i/l_i. The objective is to give priority to the tasks that can generate more resources. The results will be better or not if the estimation (task activation at the earliest time) is more or less realistic, in other words, if it can or cannot be carried out. This estimation is called Previous Weighting (PW).

The last ET (ET4) is called 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 it 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.

All the ETs were tested in [8] and the obtained results show that they really improve the solution quality. However, the values used in the ETs must be well calibrated to obtain these good results. The calibration is the same used in original paper. A range of values is defined for 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). When the metaheuristic needs to choose the parameter value (at the beginning of the instance reading) the calibration is done: twenty ADDR runs are done with some predefined values in these ranges and the value that generates the best result is chosen. Hence, all four parameter values are defined one-by-one and remain the same until the algorithm ends.

3.2. Local Searches

Local search (LS) is an improvement procedure that tries to modify an initial solution by changing small segments of it, called neighborhood. Three local search algorithms are tested: the first one (LS1) analyzes tasks i by time. It does the same calculation made by CT. If task i has cost $c_i \geq S_i$ it is removed from the current solution. The analysis begins with the tasks activated later, going to the task activated earliest. 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, not proposed in [8], tries to reconstruct the current solution with a lesser 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.

3.3. Evolutionary Algorithms

Evolutionary Algorithms (EA) and the Genetic Algorithms (GA), its most popular representative [4], are heuristics that work with a group of solutions, trying to combine them in order to generate better solutions. Basically, we need to generate a group of feasible solutions called initial population. Then, a group of individuals is chosen to combine and to generate a new population. From these two populations, a subset of individuals is chosen to make the new generation of the EA. Again, new solutions are chosen to combine and to generate a new population until the EA stop criteria is reached. Two EAs (EA1 and EA2) were previously proposed. The basic difference between them is that EA1 does not use the ETs (only uses Arc Removing). Stop criteria (50 generations), population size (30 individuals) and all others parameters are the same for both. EA1 uses LS1 on each individual created. EA2 also uses LS1, but it also uses LS2 on each individual of initial population.

In this work, a new EA (EA3) is proposed. The initial population is created like in EA1 and EA2: the ADDR algorithm (with the ETs) generates individuals until we have a set of 20 (population size) distinct individuals. LS1 and LS2 are applied to these initial individuals, like in EA2. All proposed EAs only work with feasible solutions and the populations are sorted by the individual fitness value.

A new combination algorithm (crossover operator) is proposed too: instead of choosing the earliest activation time from each parent (BP - Best Parent algorithm), like in EA1 and EA2, this new algorithm makes a list with available tasks in each parent (UL - Unique List algorithm). From this unique list some tasks are chosen like in ADDR, at each time unit. The list of parent available tasks is updated in accordance with tasks chosen by the offspring. The algorithm continues until the time unit H. LS1 is applied to each individual generated by the combination algorithm.

The populations are still divided in three classes (A - 20% best individuals; C - 20% worst individuals; B - other individuals). One parent is chosen from class A and the other from class B.

The mutation algorithm, where some perturbations are done in the current solution, is substituted by the application of LS2 to some individuals. However, this application has more chance to occur to the first individual (the best one) than to the second individual. This probability begins at 20% (for the best one) and decreases linearly until the last individual, that has a probability of 1%.

EA3 has a new mechanism to avoid premature convergence of the EA population: if the best solution is not improved in k consecutive generations (where k is an input data; in tests $k = 4$) the population is deleted and reconstructed. This reconstruction is done using the LS3 on the best individual found since the EA start. New individuals created from the best one compose the new population. The idea is to intensify the searching operations around the best solution. Again, if in four generations the best solution continues the same all the individuals are removed and a whole new population is generated by ADDR, like in initial population (recreation algorithm). This last procedure is applied if the intensification does not work, because we believe that an attempt of a radical diversification can works better. Due to its more sophisticated techniques, the EA3 is composed by fewer individuals (only 20) and does less generations (only 40). These values were defined to be fair with the previous EA versions that do not have such mechanisms. Of course, this fairness still remains very subjective, but maintaining the old parameters is more unfair.

Table 1 shows the configuration of the main methods used in the EAs. *PS* means the population size (constant throughout the algorithm). *NG* is the number of performed generations. *ETs* indicates what Enhancement Techniques are used in ADDR algorithm. *ILS* (Initial Local Search) shows the local searches used in the initial population construction; *Comb.* indicates the combination algorithm used: BP - Best Parent; UL - Unique List. *CLS* (Combination Local Search) shows the local search used on the generated individual, after the combination algorithm. *Ex* indicates the extra methods used in EA. Only EA3 uses an extra method: the intensification mechanism that uses LS3.

Alg.	PS	NG	ETs	ILS	Comb.	CLS	Ex
EA1	30	50	AR	–	BP	–	–
EA2	30	50	*	1,2	BP	1	–
EA3	20	40	*	1,2	UL	1	LS 3

Table 1. Configuration of the Evolutionary Algorithms (* means that all Enhancement Techniques are used)

Algorithm 1 EA3

```
1: $Mechanism \leftarrow Intensif$;
2: $CurPop \leftarrow InitPopGeneration(20)$;
3: $NG \leftarrow 1$;
4: while $NG \leq 40$ do
5:     $AuxPop \leftarrow OffspringGeneration(CurPop)$;
6:     $AuxPop \leftarrow LS2(AuxPop)$;
7:     $CurPop \leftarrow NatSelection(CurPop, AuxPop)$;
8:     if Best individual not improved in 4 generations then
9:         if $Mechanism = Intensif$ then
10:            $CurPop \leftarrow Intensification(CurPop)$;
11:            $Mechanism \leftarrow Diversif$;
12:        else
13:            $CurPop \leftarrow Diversification(CurPop)$;
14:            $Mechanism \leftarrow Intensif$;
15:        end if
16:    end if
17:    $NG \leftarrow NG + 1$;
18: end while
19: return Best individual;
```

In Algorithm 1 the initial population (line 2) with 20 individuals is created and LS1 and LS2 are applied to each one. The main loop (lines 4-18) controls the number of performed generations. At each one, an offspring population, with 20 individuals, is generated using the combination algorithm - here, only LS1 is applied to each generated individual. The LS2 application (line 6) occurs accordingly the explained probability. The natural selection chooses the 20 best individuals from each population to compose the next generation. However, if there are 4 generations without any improvement on the best individual the intensification mechanism is activated. In the next time that it occurs the diversification mechanism will be used instead of intensification. In other words, they are alternately activated.

3.4. Exact Algorithms

Like other scheduling problems, the DRCTSP is NP-Complete. The use of exact algorithms is almost prohibitive due to a very large number of solutions that must be analyzed to found an optimal solution. It is the main reason for the use of heuristics and metaheuristics instead of exact algorithms.

In many types of problems, the difference of spent time is low in small instances. But a little increment in the number of tasks, for example, may result in a very large difference in spent time. The instances tested in this work are the same used in [8]. Some instances with 200 tasks spend about 3 minutes to be solved by CPLEX, while the instances with 150 tasks do not spend more than 30 seconds. Instances with more than 250 tasks need many hours to be solved, when it occurs in acceptable computational time. In many instances we have a gap (between lower and upper bounds found by CPLEX) that stays around 20% after many hours of computation.

Traditional techniques to reduce this gap consists of applying integer programming methods like: Lagrangian Relaxation, Column Generation and Cut Generation and others. However, a novel set of techniques has been tested combining exact algorithms with metaheuristics [5][6][7]. A simple example may be seen in recent versions of CPLEX in which the primal bound may be informed by passing to it a feasible solution generated by a (meta)heuristic. Then, the CPLEX uses this solution to speed up the computation, reducing the gap faster.

A technique known as Local Branching [3] is very interesting: some constraint cuts are added to the original mathematical model, enforcing the computation in the neighborhood of the solution. Generally, constraints that cause a search in k-OPT neighborhood are used in this case.

Exact methods used in CPLEX have many difficulties when applied to some scheduling problems. This is the case of DRCTSP in which, as said, many instances spend hours to be solved. Thus, this work proposes a new merging technique to solve the DRCTSP that can be extended to others scheduling problems without so much changes. It is explained in following.

3.5. The proposed hybrid algorithm

Returning to the mathematical model, we can see that the constraints (7) ensure that a task will only be activated once, at maximum. However, some preliminary studies show that the earliest time units are responsible for more than half of the activated tasks at end of scheduling. In other words, good choices in these times provide better solutions, because more profits can be generated along the scheduling.

Thus, the used algorithm must foresees the future time units to select only the best tasks for the whole scheduling. This procedure is very hard to be done with (meta)heuristics due to its random aspect and, mainly, due to the computational complexity need for to do such calculations. In this manner, we propose to use an exact method to solve the first half of the scheduling and, then, a metaheuristic solves the second half, of course, making good use of the partial solution found by the exact method. To do it, the mathematical model need to be changed. Constraints (7) are divided in these two set of constraints (10 and 11).

$$\sum_{t=1}^{PT} x_{it} \quad \leq \quad 1 \quad \forall i = 1, ..., n \tag{10}$$

$$\sum_{t=PT+1}^{H} x_{it} \quad = \quad 0 \quad \forall i = 1, ..., n \tag{11}$$

The constraints (10) continue allowing the activation of a task i. However, this activation must be done until the time unit PT (partial time), where PT is in [1;H-1]. After the time unit PT, the activation is no more allowed - constraints (11). Another way to do the same thing is removing all the x_{it} where $t > PT$ from the original model.

However, an interesting remark must be done: if we remove all variables related to times $t > PT$ or if we reduce the constant H to PT the solution found (and its value) may not be good, because some tasks became unprofitable due to time reduction. Thus, the problem becomes very different and it may not provide good results.

It is necessary, for the good use of this technique, to maintain the variables like in original model. In this way, the exact methods will analyze if a task is necessary for the solution by itself or if it avails other good task (of course, in the range [1;PT]).

The time discretization in scheduling problems provides us with a good way to split the problem in sequentially dependent parts. These parts can be solved by any method and the result is passed to the next problem part until the resolution of the whole problem (this technique may be classified as Sequential Execution in [6]). However, this partition might make the optimal solution of original problem unreachable. Thus, the partial solution(s) and the final solution are only near optimal solutions.

The algorithm proposed in this work is a hybrid version combining CPLEX and EA3 proposed here. The first half of the problem is solved by CPLEX and the second one is solved by EA3, using the partial solution found by CPLEX. Some results are shown in following.

4. Instances

Two classes (A and B) of instances were used in [8]. 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 does not have any precedence. To 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. The 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 $\lceil \sqrt{n} \rceil$, if it is lesser than or equal to 1000 tasks. If the instance size is bigger than 1000 tasks, $H = \lceil \sqrt[3]{n} \rceil$.

The instance hardness is proportional to the product of number of tasks and planning process size (number of time units). In others words, an instance with higher number of tasks is harder than an instance with few tasks. In the same way, an instance with a larger number of time units is harder than an instance with few time units. The instances tested by the hybrid version algorithm (CPLEX+EA) 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.

5. Computational Results

The first test is to found the optimal value in 50 very small instances (instances with 50, 100 and 150 tasks), where CPLEX found the optimal solution in few seconds, at maximum. In Table 2 we can see that EA3 obtained better results in instances with more than 50 tasks and found only 2 results worst than EA2 results in instances with 50 tasks. In instances where the optimal value is not found by EA3 the average distances to this value stay around the same EA2 value (about 10% from optimal value).

	A-instances			B-instances		
Size	EA1	EA2	EA3	EA1	EA2	EA3
50	5	**48**	47	0	**43**	42
100	4	20	**26**	0	26	**33**
150	0	8	**12**	2	28	**29**

Table 2. Number of optimal solutions found from a set of 50 instances for each size

Table 3 shows the average computational time spent by CPLEX and EA3 (in seconds). Both EA2 and EA3 achieved a good number of optimal solutions in small instances. In many times, the CPLEX preprocessing found the optimal value and no branches were needed. Thus, the computational time is very low compared to EA3. Another reason to the high computational time of EA3 is the set of instances where EA3 did not find the optimal value and it stopped when the limit of generations was reached. Instances from class B have more precedences among the tasks than class A instances. These precedences makes the ADDR slower than CPLEX, because ADDR always needs to define what tasks are available at each time unit. Such computation does ADDR for B instances slower than for A instances (compared with CPLEX). The B instances generally need more time to be solved, independently the instance size.

	A-instances		B-instances	
Size	CPLEX	EA3	CPLEX	EA3
50	0.01	0.16	0.02	0.25
100	0.26	1.89	0.27	2.20
150	3.38	5.87	0.60	6.66

Table 3. Average computational time spent (seconds)

Table 4 shows a comparison between EA2 and EA3 in tested instances of the previous paper (the results, here, are presented per instance). The first two column values mean how much EA3 outperforms EA2 (average results of thirty runs).

Again, EA3 shows better results than EA2. In almost all instances, but mainly in instances where there are so many time units and many tasks, the mechanism to avoid premature convergence did more effect because combining two large solutions (of course, maintaining good results) is very difficult. The Unique List algorithm also is answerable for this outperform because it generates no infeasible individuals. In other words, the Best Parent algorithm often generates infeasible individuals that are rejected, once all EAs only works with feasible solutions. These useless generations slow down the EA2 algorithm. Thus, the computational time spent by EA2 is bigger (up to 65%) than EA3 due to it. In addition the Unique List algorithm does not need to test each offspring feasibleness at each algorithm step like in the Best Parent algorithm.

The last two columns from Table 4 show the variance values to these instances. The class B instances have lower variance values (not shown) than class A instances due to the graph topology. These instances have a higher number of precedences that difficulties the tasks activations. Thus the local optimals are more well-defined and the obtained result rarely varies too much. This behavior does not occurs in class A instances (variance values are shown) and, so, the variances values are very higher. The bigger values are from EA2, when comparing the EA2 and EA3 algorithms. It can be explained by the premature convergence avoidance mechanism that tries to escape from these local optimals. Thus, the EA3 results are more uniform, although in some instances, they are very high (instances 400 and 800).

Other tests with target values and time limits were done. EA2 and EA3 have a very similar behavior with the B in-

	Outperform		Variance - class A	
Size	class A	class B	EA3	EA2
100	0.0%	21.7%	0.0	0.0
200	1.7%	0.0%	83.1	60.8
300	1.3%	0.4%	410.5	1445.8
400	5.8%	51.1%	12542.7	2475.7
500	0.7%	2.9%	9718.8	25398.9
600	0.9%	1.1%	5425.3	8206.6
700	10.5%	4.8%	61556.7	757129.2
800	3.4%	5.0%	104042.7	44889.3
900	5.7%	1.0%	76592.7	1316.0
1000	0.9%	2.7%	66641.6	759503.3
1100	0.2%	0.0%	173.2	225.5
1200	0.1%	0.0%	202.3	192.4
1300	0.4%	0.0%	334.7	234.0
1400	0.7%	0.0%	210.0	200.8
1500	-0.2%	0.3%	1028.6	676.6
1600	0.5%	0.0%	210.8	544.3
1700	0.5%	5.2%	344.0	607.0
1800	0.8%	0.0%	1291.9	803.0
1900	0.3%	0.0%	300.4	343.8
2000	0.5%	2.0%	326.6	641.3

Table 4. EA3 results compared with EA2

stances, mainly in tests with a short time limit. In tests with large time limits, EA3 has a better performance because it is faster than EA2 and, so, it generates more populations. However, the best way to see the EA3 and EA2 results is using tests with target values. In the hardest instances (from 700 to 1000 tasks), EA2 did not reach the target value in none of the 30 runs. The target used in these tests was the EA3 average result (of each instance). EA2 reached the target more than 15 times only in two instances.

Another test involves the hybrid algorithm composed by CPLEX and EA3. The H time units are divided in two halves, in other words, $PT = H/2$. The first one is solved by CPLEX with all its default parameters. EA3, however, has its parameter ranges changed. The modified ADDR begins with a solution provided by CPLEX and only activates tasks after PT time units, exactly when CPLEX can not activate anything. 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 behavior to ADDR may be positive (new range varies from 0.4 to 0.8). Other parameters are maintained. In the diversification algorithm the ADDR uses the partial solution to construct other population, but in intensification algorithm EA3 may choose a time unit earlier than PT in order to reconstruct solutions from the best individual.

The instances used here have more than 1000 tasks. These instances were chosen because they have a large number of tasks, but a not so large number of time units. Due to low number of precedences in A instances, almost all tasks can be activated, although neither all of them are in the optimal solution. So, these instances are very good to be tested with the hybrid algorithm. In B instances the partial solution from the first half of the problem corresponds to more then 93% of the complete optimal solution. The Table 5 is composed by five columns: the first one shows the tested instance; the second one shows the average results (of thirty runs) using only the EA3 to the whole problem; the third one shows the variance values from the hybrid algorithm (CPLEX+EA3); the fourth 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 alone.

Inst.	EA3 A.	Hyb V.	Hyb A.	Improv.
1100a	2461	109.8	2498	1.1%
1200a	2968	39.8	3147	6.1%
1300a	2730	349.4	2867	4.8%
1400a	2388	153.2	2517	5.8%
1500a	3704	356.5	3886	3.9%
1600a	3287	131.0	3341	1.4%
1700a	4445	594.6	4631	4.6%
1800a	3877	293.9	4082	5.0%
1900a	3911	413.6	4019	2.6%
2000a	5563	218.2	5914	6.1%

Table 5. Average EA3 results compared with CPLEX+EA3

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

Table 6 shows the best solution found by each method and the known optimal solution values. The Distance column indicates the distance (in percentage) from CPLEX+EA3 to the optimal value. These distances were computed as follows: dist = 1 - (CPX/OPT), where *dist* is the distance value, *CPX* is the CPLEX+EA3 value and *OPT* is the optimal value.

CPLEX+EA3 spent more average computational time than EA3 about 6%. Constructing solutions from a partial solution is faster than constructing from beginning, thus the time spent by CPLEX+EA3 is not so larger than EA3, even with the CPLEX computations.

For instances with more than 1400 tasks, even after more than 12 hours of computational time, the optimal value of whole problem is not found. However, the Table 7 shows

Instance	EA3	Hybrid	Optimal	Distance
1100a	2464	2505	2664	6.0%
1200a	2981	3158	3291	4.0%
1300a	2759	2883	2923	1.4%
1400a	2396	2541	2648	4.0%
1500a	3742	3873	–	–
1600a	3297	3359	–	–
1700a	4465	4668	–	–
1800a	3900	4091	–	–
1900a	3928	4036	–	–
2000a	5606	5944	–	–

Table 6. Best EA3 results compared with CPLEX+EA3

the average time spent (in seconds) by CPLEX and the hybrid algorithm.

Instance	Hybrid	CPLEX
1100a	178	1576
1200a	232	2625
1300a	263	3172
1400a	305	15251

Table 7. average time spent by the hybrid and CPLEX algorithms (in seconds)

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

6. Concluding Remarks

The proposed Evolutionary Algorithm (EA3) obtained better results than the two EAs previously proposed. The EA3 intensification and diversification procedures are, maybe, the main reason to it. The new combination algorithm (Unique List) also has some importance because it can uses more informations from the parents than the old combination algorithm (used in EA1 and EA2).

This work also presented some preliminary results of a hybrid algorithm that is composed by an exact method (CPLEX) and a metaheuristic (Evolutionary Algorithm - 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 nice improvements were achieved. This integration of Linear 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.

6.1. Future Works

The preliminary results shown here may be improved with other techniques that will be tested soon. Another way to use the CPLEX algorithm is dividing the H time units in subranges $S_1, S_2 ... S_n$. The first subrange is solved by CPLEX like in tested way. The partial solution is used by CPLEX, again, to solve the second subrange until all subranges are solved. This technique may speed up the CPLEX computational time on large instances, although the original optimal value may become unreachable.

Better partitions of the time units may provide to CPLEX or EA more chances to improve their partial results. It depends on how many tasks (and, of course, its characteristics) are handled in each partial solution, however more statistical analysis are still needed.

References

[1] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 12:3–41, 1999.

[2] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization 6*, pages 109–133, 1995.

[3] M. Fischetti and A. Lodi. Local branching. In *Proc. of the Integer Programming Conference in honor of Egon Balas*, 2002.

[4] J. Gonçalves, J. Mendes, and M. G. C. Resende. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167:77–95, 2005.

[5] P. Hansen, N. Mladenovic, and D. Urosevic. Variable neighborhood search and local branching. *Computers & Operations Research*, 33:3034–3045, 2006.

[6] J. Puchinger and G. R. Raid. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In J. Mira and J. R. Álvarez, editors, *First International Work-Conference on the Interplay Between Natural and Artificial Computation*, volume 3562, pages 41–53, 2005.

[7] H. G. Santos, L. S. Ochi, and E. B. Uchoa. Combining metaheuristics and integer programming on school timetabling problem. In *Proc. of the 1st Workshop on Mathematical Contributions to Metaheuristics (MATHEURISTICS 2006), Bologna, Italy*, 2006.

[8] A. R. V. Silva and L. S. Ochi. A dynamic resource constrained task scheduling problem. In *Latin-Ibero-American Congress on Operations Research (CLAIO)*, November 2006.