A GRASP algorithm to solve the problem of dependent tasks scheduling in different machines

Manuel Tupia Anticona
Pontificia Universidad Católica del Perú
Facultad de Ciencias e Ingeniería, Departamento de Ingeniería, Sección Ingeniería Informática
Av. Universitaria cuadra 18 S/N
Lima, Perú, Lima 32
tupia.mf@pucp.edu.pe

Abstract. Industrial planning has experienced notable advancements since its beginning by the middle of the 20th century. The importance of its application within the several industries where it is used has been demonstrated, regardless of the difficulty of the design of the exact algorithms that solve the variants. Heuristic methods have been applied for planning problems due to their high complexity; especially Artificial Intelligence when developing new strategies to solve one of the most important variants called task scheduling. It is possible to define task scheduling as: .a set of N production line tasks and M machines, which can execute those tasks, where the goal is to find an execution order that minimizes the accumulated execution time, known as makespan. This paper presents a GRASP meta heuristic strategy for the problem of scheduling dependent tasks in different machines

1 Introduction

The task-scheduling problem has its background in industrial planning [1] and in task-scheduling in the processors of the beginning of microelectronics [2]. That kind of problem can be defined, from the point of view of combinatory optimization [3], as follows:

Considering M machines (considered processors) and N tasks with Tij time units of duration for each i-esim task executed in the j-esim machine, we wish to program the N tasks in the M machines, trying to obtain the most appropriate execution order,

Please use the following format when citing this chapter:

Anticona, M.T., 2006, in IFIP International Federation for Information Processing, Volume 217, Artificial Intelligence in Theory and Practice, ed. M. Bramer, (Boston: Springer), pp. 325–334.

fulfilling certain conditions that satisfy the optimality of the required solution for the problem.

The scheduling problem presents a series of variants depending on the nature and the behavior of both, tasks and machines. One of the most difficult to present variants, due to its high computational complexity is that in which the tasks are dependent and the machines are different.

In this variant each task has a list of predecessors and to be executed it must wait until such is completely processed. We must add to this situation the characteristic of heterogeneity of the machines: each task lasts different execution times in each machine. The objective will be to minimize the accumulated execution time of the machines, known as *makespan* [3]

Observing the state of the art of the problem we see that both its practical direct application on industry and its academic importance, being a NP-difficult problem, justifies the design of a heuristic algorithm that search for an optimal solution for the problem, since there are no exact methods to solve the problem. In many industries such as assembling, bottling, manufacture, etc., we see production lines where wait periods by task of the machines involved and saving the resource time are very important topics and require convenient planning.

From the previous definition we can present a mathematical model for the problem as in the following illustration and where it is true that:

- X_0 represents makespan
- X_{ij} will be 0 if the j-esim machine does not execute the i-esim task and 1 on the contrary.

Minimize X_0

s.a $$X_0 \geq \sum_{i=1}^{N} T_{ij} * X_{ij} \quad \forall j \in 1..M$$

$$X_0 \geq \sum_{j=1}^{M} X_{ij} = 1 \quad \forall i \in 1..N$$

Fig. 1. A mathematical model for the task-scheduling problem.

1.1 Existing methods to solve the task-scheduling problem and its variants

The existing solutions that pretend to solve the problem, can be divided in two groups: exact methods and approximate methods.

Exact methods [6, 7, 8, 9] try to find a sole hierarchic plan by analyzing all possible task orders or processes involved in the production line (exhaustive

exploration). Nevertheless, a search and scheduling strategy that analyzes every possible combination is computationally expensive and it only works for some kinds (sizes) of instances.

Approximate methods [3, 4 and 5] on the other hand, do try to solve the most complex variants in which task and machine behavior intervenes as we previously mentioned. These methods do not analyze exhaustively every possible pattern combinations of the problem, but rather choose those that fulfill certain criteria. In the end, we obtain sufficiently good solutions for the instances to be solved, what justifies its use.

1.2 Heuristic Methods to Solve the Task-scheduling variant

According to the nature of the machines and tasks, the following subdivision previously presented may be done:

- Identical machines and independent tasks
- Identical machines and dependent tasks
- Different machines and independent tasks
- Different machines and dependent tasks: the most complex model to be studied in this document.

Some of the algorithms proposed are:

A Greedy algorithm for identical machines propose by Campello and Maculan [3]: the proposal of the authors is to define the problem as a discreet programming one (what is possible since it is of the NP-difficult class), as we saw before.

Using also, Greedy algorithms for different machines and independent tasks, like in the case of Tupia [10] The author presents the case of the different machines and independent tasks. Campello and Maculan's model was adapted, taking into consideration that there were different execution times for each machine: this is, the matrix concept that it is the time that the i-esim task takes to be executed by the j-esim machine appears.

A GRASP algorithm, as Tupia [11]. The author presents here, the case of the different machines and independent tasks. In this job the author extended the Greedy criteria of the previous algorithm applying the conventional phases of GRASP technique and improving in about 10% the results of the Greedy algorithm for instances of up to 12500 variables (250 tasks for 50 machines).

1.3 GRASP Algorithms

- GRASP algorithms (for *Greedy Randomized Adaptive Search Procedure*) are meta heuristic techniques. T. Feo and M. Resende developed such technique by the end of the 80's [5] While the Greedy criteria let us select only the best value of the objective function, GRASP algorithms *relax or increase* this criteria in

such a way that, instead of selecting a sole element, it forms a group of elements, that candidate to be part of the solution group and fulfill certain conditions, it is about this group that a random selection of some of its elements.

This is the general scheme for GRASP technique:

GRASP Procedure (Instance of the problem)

1. While <stop-condition is not true> do
   1.1 Construction Phase (S_k)
   1.2 Improvement Phase (S_k)
2. Return (Best S_k)

End GRASP

Fig. 2. General structure of GRASP algorithm

About this algorithm we can affirm:

Line 1: the GRASP procedure will continue while the stop condition is not fulfilled. The stop condition can be of several kinds: optimality (the result obtained presents a certain degree of approximation to the exact solution or it is optimal enough); number of executions carried out (number of interactions); processing time (the algorithm will be executed during a determined period of time).

Lines 1.1 and 1.2: the two main phases of a GRASP algorithm are executed, later: construction stage of the adapted random Greedy solution; and the stage of improvement of the previously constructed solution (combinatorial analyses of most cases).

2 Proposed GRASP Algorithm

We must start from the presumption that there is a complete job instance that includes what follows: a quantity of task and machines (N and M respectively); an execution time matrix T and the lists of predecessors for each task in case there were any.

2.1 Data structures used by the algorithm

Let us think that there is at least one task with predecessors that will become the initial one within the batch, as well as there are no circular references among predecessor tasks that impede their correct execution:

Processing Time Matrix T: (T_{ij}) MxN, where each entry represents the time it takes the j-esim machine to execute the i-esim task.

Accumulated Processing Times Vector A: (A^i), where each entry Ai is the accumulated working time of Mi machine.

P_k: Set of predecessor tasks of J_k task.

Vector U: (U_k) with the finalization time of each J_k task.

Vector V: (V_k) with the finalization time of each predecessor task of J_k, where it is true that $V_k = = \max\{U_r\}, J_r \in P_k$

S_i: Set of tasks assigned to M_i machine.

E: Set of scheduled tasks

C: Set of candidate tasks to be scheduled

Fig. 3. Data structures used by the algorithm

We are going to propose to selection criteria during the development of the GRASP algorithm, what is going to lead us to generate two relaxation constants instead of only one:

- Random GRASP selection criteria for the best task to be programmed, using relaxation constant α.
- Random selection criteria for the best machine that will execute the task selected before, using an additional θ parameter.

2.2 Selection criteria for the *best task*

These criteria bases on the same principles as the Greedy algorithm presented before:

Identifying the tasks able to be programmed: this is, those that have not been programmed yet and its predecessors that have already been executed (or do not present predecessors).

For each one of the tasks able, we have to generate the same list as in the Greedy algorithm: accumulated execution times shall be established starting from the end of the last predecessor task executed.

The smallest element from each list must be found and stored in another list of local minimums. We shall select the maximum and minimum values of the variables out of this new list of local minimums: *worst and best* respectively.

We will form a list of candidate tasks RCL analyzing each entry of the list of local minimums: if the corresponding entry is within the interval [best, best+ α*(worst-best)] then it becomes part of the RCL. A task is chosen by chance out of those that form the RCL.

2. 3 Selection criteria for the *best machine*

Once a task has been selected out of RCL, we look for the best machine that can execute it. This is the main novelty of the algorithm proposed. The steps to be followed are the next ones:

The accumulated time vector is formed once again from the operation of the predecessors of the j-esim task, which is being object of analyses. Maximum and minimum values of the variables are established: worst and best respectively.

Then we form the list of candidate machines MCL: each machine that executes the j-esim task in a time that is in the interval (best, best+ θ*(worst-best)) is part of the MCL. Likewise, we will select one of them by chance, which will be the executioner of the j-esim task.

2. 4 Presentation of the algorithm

GRASP Algorithm_Construction (M, N, T, A, S, U, V, α, θ)

1. **Read and initialize N, M, α, θ, J_1, J_2,...,J_N, T, A, S, U, V**
2. **E = ø**
3. **While** $|E| \neq N$ **do**

   3.1 C = ø

   3.2 best = $+\infty$

   3.3 worst = 0

   3.4 For ℓ: 1 to N do

   If $(P_t \subseteq E)$ ^ $(J_t \in E)$ => $C = C \cup \{J_t\}$

   3.5 Bmin = ø

   3.6 For each $J_t \in C$ **do**

   3.6.1 $V_L = \max_{J_l \in P_k} \{U_l\}$

   3.6.2 $B_{min} = B_{min} \cup Min_{p\in[1,M]}\{T_{pl} + \max\{A_p, V_l\}\}$

   End for 3.6

   3.7 best = $Min\ \{B_{min}\}$ **{Selection of the best task. Formation of RCL}**

   3.8 worst = $Max\ \{B_{min}\}$

   3.9 RCL = ø

   3.10 For each $J_t \in C$ **do**

   If $Min_{p\in[1,M]}\{T_{pl} + \max\{A_p, V_l\}\} \in$ **[best, best + a* (worst-best)] =>**

   RCL = RCL $\cup \{J_t\}$

   3.11 k = ArgRandom$_{JLeRCL}${RCL}

   3.12 MCL = ø {Selection of the best machine}

   3.13 best = $Min_{p\in[1,M]}\{T_{pk} + \max\{A_p, V_k\}\}$

   3.14 worst = $Max_{p\in[1,M]}\{T_{pk} + \max\{A_p, V_k\}\}$

   3.15 For i: 1 to M do

If $T_{ik} + \max\{A_i, V_k\} \in$ **[best, best + 0* (worst-best)] => MCL = MCL** $\cup$ **{Mi}**

3.16 i = $ArgAleatorio_{M_i \in MCL}\{MCL\}$

3.17 $S_i = S_i \cup \{J_k\}$

3.18 E = E $\cup \{J_k\}$

3.19 $A_i = T_{ik} + \max\{A_i, V_k\}$

3.20 $U_k = A_i$

End While 3

4. makespan = $\max_{k \to 1..M} A_k$

5. Return S_i, $\forall i \in [1, M]$

End GRASP Algorithm_construction

Comments on the GRASP Algorithm proposed:

- Line 1: entry of the variables e initialization of the data structures needed for the working instance as N, M, T, A, U, V, E, Si for each machine, α, θ etc.
- Line 2: the process ends when all the programmed tasks of the batch are in group E.
- Line 3.1: the list of apt tasks C is initialized empty.
- Line 3.2: best and worst variables that will be used to work the intervals of the relaxation criteria are initialized.
- Line 3.4: the list of apt tasks C is formed
- Line 3.5: the list of local minimums Bmin is initialized.
- Line 3.6 – 3.6.2: entries corresponding to list V are actualized; list Bmin is formed adding each minor element of the accumulated time list of each task.
- Lines 3.7 – 3.8: maximum and minimum values of Bmin are assigned to worst and best variables respectively.
- Line 3.9: RCL list is initialized empty
- Lines 3.10 – 3.11: RCL list is formed when the condition $Min_{p \in [1..M]}\{T_{pl} + \max\{A_p, V_l\}\} \in$ [best, best + α* (worst-best)] is true, then an element from this list (k) is chosen by chance.
- Lines 3.12 – 3.16: minimum and maximum execution times for the task selected in 3.11 are chosen. MCL will form from the machines that execute such task fulfilling the condition: $T_{ik} + \max\{A_i, V_k\} \in$ [best, best + θ* (worst-best)]. In the end a machine is also chosen in an aleatoric way (variable i).
- Lines 3.17 – 3.20: structures E, A, U and S_i are actualized where it corresponds.
- Line 3: makespan is determined as the major entry of A.
- Line 4: assigned results found are given back.

3. Numeric Experiences

The instances with which the algorithm was tested are formed by an M quantity of machines, N of task and a T matrix of execution times. The values used for M and N, respectively, were:

- Number of tasks N: within the interval of 100 to 250, taking 100, 150, 200 and 250 values as points of reference.
- Number of machines M: a maximum of 50 machines taking 12, 25, 37 and 50 values as point of reference.
- Processing time matrix: it will be generated in a random way with values from 1 to 100 time units[1].

We have a total of 16 combinations for the machine-tasks combinations. Likewise, for each combination 10 different instances will be generated, which gives a total of 160 test problems solved.

3.1 Quality of the GRASP solution compared to a simply Greedy solution

As there is no literature on pre-determined test instances for such problem, we decided to confront the results with those of a Greedy algorithm[2]. In the first summary table average results of Greedy and GRASP algorithms execution (in that order) over the respective instances are shown; CPU times consumed by both algorithms, the quantity of executed iterations in the GRASP Construction phase and the values assigned to constants α and θ; finally the efficiency of the GRASP result over the Greedy result calculated as follows is also shown: 1-(GRASP Result/Greedy Result)

Machine \ Task	Makespan Greedy	CPU Used Time	GRASP			Iterations (Average)	CPU Used Time	Efficient
			Makespan	∪	∪			
100 \ 12	213.3	0.56	193.7	0.26	0.04	3750	102.53	9.189%
100 \ 25	113.7	0.59	108.6	0.24	0.01	3750	115.8	4.485%
100 \ 37	76.5	0.56	74.3	0.155	0.01	3750	127.6	2.876%
100 \ 50	59.1	0.52	57.8	0.145	0.01	3750	140.91	2.200%
150 \ 12	283.1	0.58	258.2	0.25	0.01	3000	151.18	8.795%
150 \ 25	125.2	0.52	114	0.21	0.01	3000	170.49	8.946%
150 \ 37	86.1	0.55	84.1	0.155	0.01	3000	201.7	2.323%
150 \ 50	68.6	0.53	67.4	0.115	0.01	3000	224.35	1.749%
200 \ 12	325.9	0.54	307	0.07	0.01	2500	216.83	5.799%
200 \ 25	127	0.57	117	0.247	0.01	2500	229.2	7.874%
200 \ 37	109.8	0.57	105.2	0.112	0.01	2500	248.75	4.189%
200 \ 50	76.4	0.47	74.6	0.155	0.01	2500	274.58	2.356%
250 \ 12	411.8	0.46	377.1	0.15	0.01	2500	249.2	8.426%
250 \ 25	183.5	0.58	168.2	0.28	0.01	2500	282.34	8.338%
250 \ 37	125.3	0.57	119.6	0.26	0.01	2500	309.03	4.549%
250 \ 50	96.5	0.55	91.1	0.123	0.01	2500	343.32	5.596%
Efficient								5.481%

Table 1. Summary table of GRASP vs. Greedy numeric experiments

[1] : We shall notice that a very high execution time (+ oo) may be interpreted as if the machine does not execute a determined task. In this case an execution time equal to 0 may be confuse as the machine executes the task so fast that it does it instantaneously, without taking time

[2] : In order to have a Greedy behavior it is enough to make constant a equal to 0 without regardless of selection of the best machine; that is why the algorithm is not added.

3.2 Real quality of the GRASP solution confronted with mathematical model's solutions

In order to determine the real quality of the solutions we decided to apply the mathematical model of the problem to linear programming problem solver packages as LINDO tool in its student version, in order to obtain exact solutions and confront them with those of the proposed GRASP algorithm.

The two tables that follow summarize the exact results obtained confronted with the heuristic (voracious solution) and meta heuristic (GRASP solution) results. In addition to this we will present the values of the relaxation constants used for the GRASP construction phase where nearly 7000 iterations for all the cases were carried out.

Table 2. Experimental results for M = 3

Matrix	N	M	Lindo Model	Greedy Algorithm	GRASP Algorithm	% Lindo/Greedy	% Lindo/GRASP
6x3_0	6	3	28	35	35	25.00%	0.00%
6x3_1	6	3	67	67	67	0.00%	0.00%
6x3_2	6	3	75	87	85	16.00%	2.30%
8x3_0	8	3	40	45	44	12.50%	2.22%
8x3_1	8	3	88	102	102	15.91%	0.00%
8x3_2	8	3	64	82	64	28.13%	21.95%
12x3_0	12	3	130	162	162	24.62%	0.00%
12x3_1	12	3	121	134	121	10.74%	9.70%
12x3_2	12	3	80	108	89	35.00%	17.59%
15x3_0	15	3	162	201	168	24.07%	16.42%
15x3_1	15	3	125	137	125	9.60%	8.76%
15x3_2	15	3	164	190	176	15.85%	7.37%
Efficient mean						18.12%	7.19%

Table 3. Experimental results for M = 5

Matrix	N	M	Lindo Model	Greedy Algorithm	GRASP Algorithm	% Lindo/Greedy	% Lindo/GRASP
10x5_0	10	5	45	61	46	35.56%	24.59%
10x5_1	10	5	40	52	40	30.00%	23.08%
10x5_2	10	5	61	74	61	21.31%	17.57%
15x5_0	15	5	59	60	59	1.69%	1.67%
15x5_1	15	5	64	77	64	20.31%	16.88%
15x5_2	15	5	41	65	41	58.54%	36.92%
20x5_0	20	5	88	121	106	37.50%	12.40%
20x5_1	20	5	66	74	66	12.12%	10.81%
20x5_2	20	5	82	115	87	40.24%	24.35%
25x5_0	25	5	96	121	113	26.04%	6.61%
25x5_1	25	5	109	147	129	34.86%	12.24%
25x5_2	25	5	86	96	96	11.63%	0.00%
Efficient mean						27.48%	15.59%

4. Conclusions

Both criteria adapted to the kind of algorithm presented: the criteria were voracious in the case of the voracious algorithm (un-modifiable selection); or adaptable and random as in the case of the GRASP algorithm. In the literature there are not any GRASP algorithm that considers a double relaxation criteria at the moment of making the correspondent selections: conventional GRASP ones, were just part of a RCL list of candidates for the tasks to be executed. In 100% of the cases the result of the GRASP algorithm is better than that of voracious algorithm for high enough test instances (proportion 4 to 1, 5 to 1). In small instances it is at least equal to the voracious solution or it does not reach the same level by a very little percentage. The percentage of advantage of the GRASP algorithm confronted to the voracious algorithm is in average 5%.

GRASP algorithm get much closer to the exact solution for analyzed instances, within an average range of 5% to 9% of those solutions. In multiple cases it equals the solution, behavior that has been seen as we reduce the task-machine proportions, this is, when the number of available machine increases.

REFERENCES

[1] G. Miller, E. Galanter. Plans and the Structure of Behavior. Holt Editorial, 1960.
[2] M. Drozdowski. Scheduling multiprocessor tasks: An overview. *European Journal Operation Research*, 1996, pp. 215 - 230.
[3] R. Campello, N. Maculan. Algorithms e Heurísticas: desenvolvimiento e avaliaçao de performance. Apolo Nacional Editores. Brasil, 1992.
[4] M. Pinedo. Scheduling: Theory, Algorithms and Systems, Prentice Hall, 2002.
[5] T. Feo, M. Resende, Greedy Randomized Adaptive Search Procedure. *In Journal of Global Optimization*, No. 6, 1995, pp. 109-133.
[6] W. Rauch, Aplicaciones de la inteligencia Artificial en la actividad empresarial, la ciencia y la industria - tomo II. Editorial Díaz de Santos. España, 1989.
[7] P. Kumara Artificial Intelligence: Manufacturing theory and practice. NorthCross - Institute of Industrial Engineers, 1988.
[8] A. Blum, M. Furst. Fast Planning through Plan-graph Analysis. *In 14th International Joint Conference on Artificial Intelligence.* Morgan- Kaufmann Editions, 1995, pp. 1636 -1642.
[9] R. Conway. Theory of Scheduling, Addison–Wesley Publishing Company, 1967
[10] M. Tupia. Algoritmo voraz para resolver el problema de la programación de tareas dependientes en máquinas diferentes. *In International Conference of Industrial Logistics (7, 2005, Uruguay)* ICIIL. Editors. H. Cancela, Montevideo – Uruguay, 2005, p. 345.
[11] M. Tupia., D. Mauricio. Un algoritmo GRASP para resolver el problema de la programación de tareas dependientes en máquinas diferentes. In *CLEI (30, 2004, Perú)* Editors. M. Solar, D. Fernández-Baca, E. Cuadros-Vargas, Arequipa – Perú, 2004, pp. 129—139.