A Constructive Algorithm to Minimise the Makespan on Identical Parallel Machines

José Luis Jimeno, Ethel Mokotoff and Joaquín Pérez
Dpto. de Fundamentos de Economía e Historia Económica, Universidad de Alcalá, Spain.
josel.jimeno@uah.es, ethel.mokotoff@uah.es, joaquin.perez@uah.es

Abstract
In this paper we present constructive algorithms for the classical deterministic scheduling problem of minimising the makespan on identical parallel machines. Since the problem is known to be NP-hard in the strong sense, the approximate algorithms play a relevant role when solving this problem. The proposed algorithms are based on list scheduling procedures. The algorithms consist in not taking into consideration all the machines at the beginning, but taking into account subsets of machines, step by step, until the full set of machines are employed. Computational results show that the proposed algorithms perform very well, taking less time to run than the seminal Longest Processing Time First Algorithm.

1. Introduction

In this paper we deal with the classical Parallel Machine Scheduling (PMS) problem with n independent jobs and m identical parallel machines. All the jobs have to be carried out on one of the machines without pre-emption. The problem of finding the schedule that minimises the maximum job completion time is considered (denoted by $P//C_{max}$).

Due to the NP-hardness of the problem in hand, heuristic methods have been intensively applied. The only way to secure optimal solutions is by enumerative techniques (for a survey of PMS problems, see Mokotoff [4]).

The aim of this article is to present constructive algorithms based on list scheduling [2]. The basic idea is to consider a partition of the full set of identical machines. The jobs are assigned to a machine from a subset, adding new machine subsets, until all jobs have been assigned to a machine. Computational results show that, to deal with the problem that we are considering, the proposed algorithms (PA) perform very well.

In the following section, the scheduling problem is analysed, and previous results are then recalled. In Section 3, the new algorithms are presented. Computational experiments are described in Section 4. We conclude with a summary.

2. Problem definition and previous results

We will use this notation:

J_i: job i, $i \in J = \{1, 2, ..., n\}$

M_j: machine j, $j \in M = \{1, 2, ..., m\}$

p_i: processing time of job i

C_i: completion time of job i

C_{max}: makespan, the maximum completion time of all jobs i, $C_{max} = \max \{C_1, C_2,..., C_n\}$

Let J be a set of n jobs i to be processed, each of them on one machine, from a set M of m machines. All the jobs can be processed on any of the m machines. We consider the identical machine model, for which the processing time of each job, p_i, is independent of the machine processing it. The objective is to find an appropriate allocation of jobs from set J to machines from set M that would minimise the maximum completion time criterion, called makespan.

The most important constructive algorithms dedicated to the PMS problem can be classified by their design into List Scheduling [2] and Bin Packing, (based on the MultiFit algorithm of

Coffman et al. [1]). List Scheduling (LS) and MultiFit (MF) algorithm performances have a mean of 1 and a variance of 0, for arbitrary distributions and arbitrary LS algorithms. LS algorithms share a similar way of assigning jobs: a list of all the jobs is made, then the uppermost job is assigned to the first available machine and the process continues until all the jobs in the list have been assigned to a certain machine. As regards how to make a list of jobs, there are different kinds of LS algorithms, the Longest Processing Time algorithm consists in making a list according to LPT first order and assigning the uppermost job from the list to a free machine or to that one with the least processing time already assigned to it, until the list is exhausted. This takes $O(nlogn+nm)$, where the first term corresponds to ordering jobs, and the second to the assignment of each job to a machine [3]. For large n, LPT is an excellent heuristic; in fact, C_{max}(LPT)-C^*_{max} converges towards zero when n converges towards infinity, C^*_{max} being the optimal makespan value.

LPT continued to be the best algorithm until 1978, when Coffman et al. [1] presented the MF algorithm. This algorithm takes into account the duality existing between the $P//C_{max}$ and the Bin Packing problem. The MF algorithm performs better than the previous ones by increasing the computational complexity ($O(nlogn+knm)$, where k is the number of iterations and it is recommended that k be at least 7).

3. Proposed Algorithm

We present a heuristic algorithm that can be described as follows: Starting with a list of the jobs in LPT order and a partition of the set of machines in r subsets ($r \leq m$), the algorithm takes the first one of these subsets S_1 and assigns the longest jobs according to the LPT rule, until its completion time is nearest to a fixed limit (for example a lower bound of makespan), without exceeding it. When there is no machine in S_1 to which it is possible to assign a job, without surpassing the limit, a new subset of machines S_2 is added to the first one and the algorithm follows in the same manner with the assignment of jobs that have not yet been assigned. This procedure continues until we have added the last subset of machines. At this point the algorithm follows the LPT rule, assigning the rest of the jobs without taking into account the fixed limit.

LS algorithms are as useful as they are simple. They build a schedule, starting by making a list of jobs, following a given rule of ranking, and the uppermost job from the list is assigned to a selected machine according to a certain assignment rule, until the list of jobs is exhausted, even though in the problem in hand finding the list and the assignment rule that gives the minimum schedule length is *NP-hard*. It seems to be obvious that the allocation must be made by balancing the jobs among the set of machines. The progressive introduction of machines allows equilibrium in the schedule length of all machines to be achieved (to finish with the smallest time difference between them).

In the following, the proposed algorithm is described step-by-step.

Step 1.

List the jobs $J = \{J_1, J_2, ..., J_n\}$ in LPT order. Take an r partition $S = \{S_1 , S_2 , ... , S_r\}$ of the machines set $M = \{M_1, M_2, ..., M_m\} = S_1 \cup S_2 \cup ... \cup S_r, r \leq m$.

Set $L(C_{max}) = max\left\{\frac{1}{m}\sum_{i=1}^{n} p_i, \underset{i}{max}\{p_i\}, p_m + p_{m+1}\right\}$, being $i \in \{1, 2, ..., n\}$, and select a value for $\delta \in [0, 1]$

Step 2.

Set $C_j = 0$, for $j = 1, 2, ..., m$. List the r subsets of machines $S_1, S_2, ...S_r$. Select the set of machines $L = S_1$, and remove S_1 from S.

Step 3.

-If $J = \emptyset$, $C_{max} = \underset{i}{max}\{C_i\}$, returning the objective value of the solution that has been built.

-Otherwise, select the first job from J and remove it from the set.

Step 4.

Set $min\,C = \min_{j \in L}\left\{\max_{i \notin J}\{C_i\}\right\}$ and let t be the corresponding j with minimum $\left\{\max_{i \notin J}\{C_i\}\right\}$

Step 5.

-If $min\,C + p_i \leq \delta L(C_{max})$ or $min\,C = 0$, then assign job i to machine t, and go to Step 3.

-Otherwise:

-If $L \neq M$, select the first subset S_k from S, update $L = L \cup S_k$, and go to Step 4.

-Otherwise, assign job i to machine t, and go to Step 3.

Some open questions remain: what is the appropriate value of δ?, what is the optimal partition of the machine set?. These questions have led us to develop different algorithms. We have tried with δ=1, 0.95, 0.90, 0.85, 0.80, 0.75 and 0.50, and three different machine set partition as described in the following section.

4. Computational Results

To know the efficiency of the proposed procedure described in Section 3 we have compared it with the LPT [2], the algorithms presented by Mokotoff et al. (MJG) [5] and the MF algorithm [1]. We have tested three algorithms with different types of partitioning. As the machines are identical, the only question to be taken into account when making the partition is the size of each subset, so we have the following partition scheme:

$0.5m$, $0.25m$, $0.15m$, $0.1m$, denoting this algorithm as AP_{ni}.

$0.1m$, $0.15m$, $0.25m$, $0.5m$, denoting this algorithm as AP_i.

$0.1m$, $0.1m$, $0.1m$, $0.1m$, $0.1m$, $0.1m$, $0.1m$, $0.1m$, $0.1m$, $0.1m$, denoting this algorithm as AP_{10}.

We carried out the computational experiments by considering two series of test problems obtained by randomly generating the p_i values according to uniform distributions in different ranges. Series 1 and 2 correspond to the ranges [1, 100] and [1, 1000], respectively.

The results have proved that PA actually take much less time to run than MJG and MF. Thus, as regarding the issue of error deviation, PA is only comparable with LPT where it is shown to be clearly better. Due to space limitations we only present a few, but representative results in Tables 1 and 2. For the different values of m and n the entries give, for each algorithm, the average values over 10 problem instances of the makespan's error percentage with respect to McNaughton's lower bound (%Error) and the CPU time required to run them (Time).

Table 1. Series 1 - [1, 100]. Average values over 10 problems. Deviation with respect to $L(C_{max})$

		LPT		MF_7		AP_{10}(δ=1)		AP_{ni}(δ=1)		AP_i(δ=1)	
M	**N**	**%Error**	**Time**	**%Error**	**Time**	**%Error**	**Time**	**%Error**	**Time**	**%Error**	**Time**
10	50	2,088	0,002	1,005	0,002	1,195	0,001	1,533	0,000	1,632	0,001
	100	0,437	0,002	1,298	0,003	0,273	0,000	0,392	0,002	0,316	0,000
	250	0,069	0,003	1,446	0,006	0,038	0,003	0,053	0,002	0,038	0,001
	500	0,000	0,002	1,505	0,008	0,000	0,003	0,004	0,004	0,000	0,002
	1000	0,000	0,006	1,539	0,020	0,000	0,005	0,000	0,006	0,000	0,005
20	50	5,948	0,000	1,865	0,005	5,015	0,001	5,793	0,001	5,948	0,001
	100	2,176	0,002	0,964	0,004	0,955	0,001	1,428	0,001	1,194	0,002
	250	0,253	0,002	1,341	0,007	0,123	0,001	0,232	0,006	0,233	0,002
	500	0,063	0,002	1,437	0,015	0,032	0,003	0,056	0,006	0,047	0,004
	1000	0,008	0,009	1,511	0,033	0,004	0,008	0,004	0,009	0,012	0,009
50	100	2,993	0,002	2,989	0,006	2,993	0,003	2,993	0,002	2,993	0,002
	250	2,182	0,004	0,992	0,017	1,312	0,006	1,865	0,007	1,552	0,004
	500	0,423	0,010	1,244	0,034	0,299	0,010	0,381	0,006	0,302	0,009
	1000	0,089	0,023	1,410	0,344	0,030	0,021	0,069	0,017	0,089	0,019

Table 2. Series 2 - [1, 1000]. Average values over 10 problems. Deviation with respect to $L(C_{max})$

		LPT		**MF$_7$**		**AP$_{10}$(δ=1)**		**AP$_{ni}$(δ=0.75)**		**AP$_i$(δ=0.90)**	
m	**n**	**%Error**	**Time**	**%Error**	**Time**	**%Error**	**Time**	**%Error**	**Time**	**%Error**	**Time**
10	50	2,621	0,002	1,508	0,002	1,527	0,000	1,727	0,001	1,813	0,001
	100	0,490	0,005	1,542	0,005	0,420	0,000	0,435	0,001	0,485	0,003
	250	0,077	0,003	1,553	0,005	0,050	0,001	0,062	0,002	0,088	0,002
	500	0,021	0,005	1,557	0,011	0,016	0,004	0,025	0,006	0,022	0,003
	1000	0,004	0,010	1,560	0,018	0,003	0,010	0,004	0,008	0,004	0,005
20	50	8,497	0,002	2,198	0,003	7,109	0,002	8,497	0,002	8,497	0,002
	100	2,700	0,002	1,505	0,003	1,498	0,001	1,572	0,001	1,605	0,002
	250	0,327	0,006	1,537	0,009	0,199	0,001	0,315	0,001	0,293	0,003
	500	0,086	0,004	1,550	0,018	0,046	0,008	0,051	0,004	0,057	0,006
	1000	0,021	0,009	1,557	0,035	0,012	0,006	0,015	0,011	0,014	0,011
50	100	4,519	0,003	4,821	0,006	4,519	0,002	4,519	0,003	4,519	0,002
	250	2,345	0,006	1,506	0,020	1,463	0,006	1,624	0,008	1,360	0,007
	500	0,446	0,012	1,532	0,040	0,413	0,012	0,361	0,010	0,375	0,012
	1000	0,120	0,028	1,547	0,075	0,090	0,019	0,104	0,021	0,109	0,022

5. Conclusions

In this paper we have proposed a new constructive technique for a practical *NP-hard* scheduling problem, $P//C_{max}$. The approach is based upon a simple and intuitive list-scheduling technique; however, the overall method we have introduced is an original strategy. The strategy consists in considering a progressive introduction of the machines to the problem.

These computational results show that the proposed procedure is promising, especially when time constitutes an important issue. As further research we are exploring other machine set partition schemes. Not only is it interesting as a technique to be included in algorithms appropriate for finding the end-point solution, but also the technique can be used as an initial solution of enumerative methods such as Branch and Bound or Branch and Cut.

Acknowledgements

This research has been partially supported by the Research Project H008/2001, Universidad de Alcalá.

References

[1] Coffman, E.G. Jr., Garey, M.R., and Johnson, D.S. (1978). An application of Bin-Packing to multiprocessor scheduling. *SIAM Journal on Computing* 7, 1-17.

[2] Graham, R.L. (1966). Bounds for certain multiprocessing anomalies. *Bell System Technical Journal* 45, 1563-1581.

[3] Graham, R.L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics* 17, 263-269.

[4] Mokotoff, E. (2001). Parallel Machine Scheduling Problems: A Survey. *Asia-Pacific Journal of Operational Research* 18, 193-242.

[5] Mokotoff, E., Jimeno, J.L. and Gutiérrez, A.I. (2001). List Scheduling Algorithms to Minimize the Makespan on Identical Parallel Machines. *TOP* 9/2, 243-269.