Computers Ops Res. Vol. 21, No. 2. pp. 205–210, 1994
Printed in Great Britain.

0305-0548/94 $6.00 + 0.00

A COMPOSITE HEURISTIC FOR THE IDENTICAL PARALLEL MACHINE SCHEDULING PROBLEM WITH MINIMUM MAKESPAN OBJECTIVE

PAULO M. FRANÇA,[1]† MICHEL GENDREAU,[2]‡ GILBERT LAPORTE[2]§¶ and FELIPE M. MÜLLER[1,2,3]‖

[1]Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, Brazil, [2]Centre de Recherche sur les Transports, Université de Montréal, CP 6128, Succursale A, Montréal, Canada H3C 3J7 and [3]Núcleo Processamento de Dados, Universidade Federal de Santa Maria, Brazil

(Received August 1992; in revised form November 1992)

Scope and Purpose—In several manufacturing processes, it is necessary to determine an assignment of independent jobs to identical parallel machines in order to minimize the total completion time or makespan. There has been a great deal of research on this hard combinatorial problem. One active line of research has been the design of heuristics. This paper proposes a new fast and powerful heurisrtic for this problem.

Abstract—This paper describes a new heuristic algorithm for the problem of scheduling n independent jobs on m identical parallel machines in order to minimize the makespan. The algorithm produces near-optimal solutions in short computation times. It outperforms alternative heuristics on a series of test problems.

1. INTRODUCTION

In this article, we propose a new heuristic procedure for the problem of scheduling n independent jobs $J_1, \ldots, J_n$ on m identical parallel machines $M_1, \ldots, M_m$ without preemption, where $n \geqslant m \geqslant 2$ and each job J_j has a positive processing time p_j. Let C_i be the completion time on machine M_i. Then the problem is to determine a schedule minimizing the makespan $C_{\max} = \max_{i=1,\ldots,m}\{C_i\}$. Using the three-field classification scheme introduced by Graham *et al.* [1], this problem is denoted as $P \| C_{\max}$.

Our purpose is to present 3-PHASE, a composite heuristic consisting of a construction phase, followed by two improvement phases. This algorithm is described in Section 2. Computational results, presented in Section 3, indicate that the proposed algorithm produces near-optimal solutions and compares favorably against alternative heuristics.

The $P \| C_{\max}$ problem is one of the most studied problems in the field of scheduling theory. Surveys have been provided by Lawler *et al.* [2], and by Cheng and Sin [3]. Garey and Johnson [4] have proved that the problem is NP-hard in the strong sense when the number of machines is arbitrary. The problem is solvable in pseudo-polynomial time when the number of machines is fixed and thus NP-hard only in the ordinary sense. Optimal algorithms have been proposed by Blazewicz [5] and by Dell'Amico and Martello [6]. In addition, Bratley *et al.* [7] describe an enumerative procedure for a generalization of the problem in which jobs have earliest start and due date constraints.

Most algorithms developed for $P \| C_{\max}$ are heuristics. These can be broadly classified into constructive heuristics and improvement heuristics. The majority of algorithms belong to the first category and have a known worst-case performance ratio (see, e.g. Graham [8], Coffman *et al.* [9], Friesen [10], Friesen and Langston [11], Sahni [12], Hochbaum and Shmoys [13], and Leung [14]). A well known constructive heuristic is the so-called LPT (longest processing time) method

†P. M. França is Professor at the State University of Campinas, Brazil.
‡M. Gendreau is researcher at the Centre for research on transportation (CRT) in Montreal.
§G. Laporte is Professor at the École des Hautes Études Commerciales de Montréal, and member of the CRT.
¶Author for correspondence.
‖F. M. Müller is a researcher at the Federal University of Santa Maria, Brazil. He is now enrolled as a Ph.D. student at the State University of Campinas, Brazil, and is currently visiting the CRT.

due to Graham [8]. Here the jobs are sorted and relabeled so that $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_n$. Starting with job J_1, all jobs are then successively assigned to the least loaded machine. The worst-case ratio of LPT is equal to $4/3 - 1/(3m)$, and the time complexity of the algorithm is equal to $O(n \log n)$. Numerical tests have shown that LPT has a good empirical performance, especially for large values of n [15]. Another classical constructive heuristic is the multifit (MF) algorithm of Coffman *et al.* [9]. Here, one seeks to determine the smallest machine capacity so that a feasible solution can be found using the LPT scheme. This is achieved by solving heuristically a series of bin packing problems (see, e.g. Coffman *et al.* [16]). It is shown that if k packing attempts are made, MF can run in $O(n \log n + kn \log n)$ time and the worst-case ratio of the algorithm does not exceed $11/9 + 2^{-k}$. In improvement heuristics, job reassignments and interchanges are attempted to improve a feasible solution obtained by any means. The 0/1 interchange algorithm of Finn and Horowitz [15] is of this type; it has a worst-case performance ratio of 2. An improved procedure with a bound of 4/3 has later been proposed by Langston [17].

2. THE 3-PHASE ALGORITHM

The proposed 3-PHASE algorithm can be summarized as follows. In Phase 1, jobs are classified according to their processing times and assigned to machines in order to achieve a fairly balanced load. In Phase 2, the workload balance is improved by repeatedly moving a job from the most loaded to the least loaded machine. Finally in Phase 3, an even better balance is achieved by interchanging jobs between machines. We now describe the three phases.

Phase 1: initial assignment

Contrary to several known constructive algorithm, it is not necessary in this constructive phase to sort the jobs in order of processing times. Only $\underline{p} = \min\{p_j\}$ and $\bar{p} = \max\{p_j\}$ are required. Then, $[\underline{p}, \bar{p}]$ is partitioned into r equal intervals $I_1, \ldots, I_r$, where r is an input parameter of the algorithm. Job J_j is called an ℓ-job if $p_j \in I_\ell$. First initialize $s_1 = \ldots = s_r = 0$, the index of the machine to which the last ℓ-job has been assigned ($\ell = 1, \ldots, r$). Then consecutively consider all jobs J_j: if J_j is an ℓ-job, assign it to machine $s_\ell \pmod m + 1$ and set $s_\ell = s_\ell \pmod m + 1$.

Phase 2: job reassignments

Let $\bar{C} = \sum_{i=1}^{m} C_i/m$ be the average machine completion time; this is independent of the particular assignment of jobs to machines. This step consists of repeatedly moving a job from the busiest machine (with the largest C_i), to the least busy machine in order to reduce the makespan. The procedure is now described.

Step 1. Identify the most and least busy machines and relabel them M_1 and M_m, respectively.

Step 2. Compute $\Delta_1 = C_1 - \bar{C}$, $\Delta_m = \bar{C} - C_m$, $\Delta = \min\{\Delta_1, \Delta_m\}$, $\Delta' = C_1 - C_m$. If $\Delta < \underline{p}$, set $\ell = 0$ and go to Step 5. Otherwise, Δ belongs to some interval I_ℓ, or $\Delta > \bar{p}$, in which case set $\ell = r$.

Step 3. If no job J_j with $p_j \in I_\ell$ and $p_j < \Delta'$ is assigned to M_1, go to Step 4. Otherwise, reassign a job of I_ℓ from M_1 to M_m and go to Step 1.

Step 4. If no job J_j with $p_j \in I_\ell^- = I_1 \cup \ldots \cup I_{\ell-1}$ is assigned to M_1, go to Step 5. Otherwise, reassign a job J_j with $p_j \in I_\ell^-$ from M_1 to M_m and go to Step 1.

Step 5. If $\ell = r$, Phase 2 terminates. Otherwise, scan successively $I_{\ell+1}, I_{\ell+2}, \ldots, I_r$ for a job J_j such that $p_j < \Delta'$. (Note that scanning may stop at the first interval for which all jobs have processing times at least equal to Δ'.) If no such job exists, Phase 2 terminates. Otherwise, reassign job J_j from M_1 to M_m and go to Step 1.

Two remarks are in order. First, the use of $\bar{C}$ as a 'target' to guide the job reassignment is advantageous when compared with the standard procedure used in [15], since machines with completion times in the neighborhood of $\bar{C}$ are unlikely to be selected for reassignments. Second, the fact that jobs are grouped in subsets according to their processing times provides for a more efficient search of candidate jobs for reassignment. Tests have shown that in Steps 3 or 4, there is

no real advantage, when doing reassignment, in looking for the job J_j with the best fit, i.e. with the largest p_j not exceeding Δ; any job with $p_j \leqslant \Delta$ will do.

Phase 3: job interchanges

Again, M_1 and M_m refer to the two machines with heaviest and lightest workload, respectively. (The remaining machines are labeled arbitrarily.) In this phase we attempt to interchange a job from M_1 (job J_j, say) with a job from another machine M_h (job $J_{j'}$, say), starting with $h=m$, $m-1, \ldots$, until an advantageous exchange has been identified, i.e. an exchange such that $\theta = p_j - p'_j > 0$ and $\theta = p_j - p'_j < d = C_1 - C_h$. The idea is to use the value of $d/2$ as a target for the difference in processing times between two interchanged jobs, very much in the way $\bar{C}$ was used in Phase 2. To determine the target value, observe that after interchange, the value of C_1 will decrease to $C'_1 = C_1 - \theta$, and that of C_h will increase to $C'_h = C_h + \theta$, so that $|C'_1 - C'_h| = |C_1 - C_h - 2\theta| = |d - 2\theta|$, and this value is minimized for $\theta = d/2$. It is not necessary to enumerate all possible interchange pairs. Instead, we take advantage of the interval structure induced by the job processing times, as the search is restricted to job pairs belonging to two intervals located approximately $d/2$ units apart. Formally, the procedure can be described as follows.

Step 1. Set $h=m$.

Step 2. Set $t=r$, $f_t = C_1$ and $d = C_1 - C_h$. If $\bar{p} - d/2 \leqslant \underline{p}$, set $s=1$. Otherwise, let s be the index of the interval to which $\bar{p} - d/2$ belongs.

Step 3. If there exist jobs J_j and $J_{j'}$ with $p_j \in I_t$, $p_{j'} \in I_s$, such that J_j is assigned to M_1, $J_{j'}$ is assigned to M_h, and $0 < p_j - p_{j'} < d$, compute $f_t = \min_{J_j, J_{j'}} \{\max[C_1 - p_j + p_{j'}, C_h + p_j - p_{j'}]\}$, otherwise set $f_t = C_1$. If $s=1$, set $t = t-1$ and go to Step 5.

Step 4. Set $s = s-1$, $t = t-1$ and go to Step 3.

Step 5. Set $f^* = \min_{t \leqslant r} \{f_t\}$. If $f^* < C_1$, interchange the two jobs J_j and $J_{j'}$ yielding f^* and go to Phase 2. If $f^* = C_1$, set $h = h-1$. If $h > 1$, go to Step 2. If $h=1$, stop. ■

It is worth observing that the partitioning of jobs into r intervals has a significant impact on the computation of f_t in Step 3, and hence on the overall performance of Phase 3. Assuming uniformly distributed processing times, the average number of jobs per machine with processing time in a given interval is approximately equal to n/rm, which implies that around $(n/rm)^2$ pairs of jobs are considered in the determination of f_t for each value of t. Therefore, the computation of f^* requires approx. $2r(n/rm)^2$ comparisons, as opposed to $2(n/m)^2$ when no partition is used.

Considering the fact that the heuristic may iterate an unspecified number of times between Phases 2 and 3, it is not possible to derive a simple worst-case complexity bound for it. A worst-case performance bound of 2 for the heuristic results from the fact that after the first execution of Phase 2, the Finn and Horowitz [15] stopping criterion is satisfied. The empirical performance of the heuristic on random problems is also very good, as will be illustrated in Section 3.

3. COMPUTATIONAL RESULTS

The 3-PHASE heuristic was compared with three other well known algorithms:

(1) the classical 0/1 interchange heuristic of Finn and Horowitz [15]
(2) the improved 0/1 interchange heuristic of Langston [17]
(3) the LPT heuristic of Graham [8].

The 0/1 interchange heuristic and its improved version are known to have very good computational performances. The LPT heuristic was selected as it is often used as a benchmark for test problems. We did not compare 3-PHASE with the Multifit algorithm of Coffman *et al.* [9] as this algorithm would appear to be computationally comparable to LPT [18].

The four heuristics were compared with each other, and the solution values produced by these algorithms were also compared with the optimal values obtained by solving a sequence of bin packing problems within a bisection search scheme [6]. For the solution of the bin packing problems, we used the MTP code of Martello and Toth [19].

Table 1. Recommended values r^*

n/m	r^*
(1, 10]	2
[10, 50)	5
[50, 100)	10
[100, 200)	15
[200, 400)	20

Table 2. Computational results for $p_j \in [1, 100]$

m	n	$\left(\frac{z-z^*}{z^*}\right)$	$\left(\frac{z_1-z}{z}\right)$	$\left(\frac{z_2-z}{z}\right)$	$\left(\frac{z_3-z}{z}\right)$	CPU time (s)				
						z^*	z	z_1	z_2	z_3
5	10	0.018	0.027	0.027	0.000	0.003	0.002	0.032	0.023	0.032
	50	0.000	0.008	0.008	0.005	0.003	0.002	0.032	0.032	0.035
	100	0.000	0.001	0.001	0.001	0.008	0.007	0.030	0.038	0.030
	500	0.000	0.000	0.000	0.000	0.024	0.023	0.112	0.125	0.063
	1000	0.000	0.000	0.000	0.000	0.063	0.062	0.215	0.217	0.152
10	50	0.002	0.025	0.025	0.013	0.013	0.007	0.035	0.032	0.033
	100	0.002	0.007	0.007	0.003	0.008	0.007	0.038	0.042	0.032
	500	0.000	0.001	0.001	0.000	0.014	0.013	0.137	0.132	0.057
	1000	0.000	0.000	0.000	0.000	0.034	0.033	0.295	0.298	0.102
25	50	0.011	0.008	0.008	0.003	0.017	0.013	0.032	0.038	0.037
	100	0.003	0.004	0.004	0.021	0.023	0.010	0.042	0.045	0.037
	500	0.000	0.002	0.002	0.001	0.026	0.025	0.160	0.157	0.055
	1000	0.000	0.000	0.000	0.000	0.041	0.040	0.425	0.422	0.083

Table 3. Computational results for $p_j \in [1, 1000]$

m	n	$\left(\frac{z-z^*}{z^*}\right)$	$\left(\frac{z_1-z}{z}\right)$	$\left(\frac{z_2-z}{z}\right)$	$\left(\frac{z_3-z}{z}\right)$	CPU time (s)				
						z^*	z	z_1	z_2	z_3
5	10	0.010	0.037	0.037	−0.001	0.002	0.001	0.030	0.033	0.033
	50	0.001	0.004	0.004	0.003	0.092	0.005	0.038	0.033	0.037
	100	0.000	0.002	0.002	0.001	0.022	0.005	0.043	0.037	0.035
	500	0.000	0.000	0.000	0.000	0.021	0.020	0.170	0.165	0.067
	1000	0.000	0.000	0.000	0.000	0.046	0.045	0.527	0.528	0.150
10	50	0.002	0.018	0.018	0.013	3.962	0.002	0.035	0.037	0.033
	100	0.000 (7)	0.007	0.007	0.004	3.524 (7)	0.010	0.040	0.037	0.033
	500	0.000	0.000	0.000	0.000	0.037	0.020	0.178	0.180	0.052
	1000	0.000	0.000	0.000	0.000	0.040	0.038	0.567	0.562	0.100
25	50	0.011	0.075	0.075	0.000	0.017	0.012	0.035	0.033	0.037
	100	0.003 (3)	0.050	0.050	0.028	11.433 (3)	0.015	0.038	0.037	0.032
	500	0.000	0.002	0.002	0.001	0.187	0.035	0.168	0.173	0.050
	1000	0.000	0.000	0.000	0.000	0.064	0.063	0.565	0.567	0.087

Table 4. Computational results for $p_j \in [1, 10{,}000]$

m	n	$\left(\frac{z-z^*}{z^*}\right)$	$\left(\frac{z_1-z}{z}\right)$	$\left(\frac{z_2-z}{z}\right)$	$\left(\frac{z_3-z}{z}\right)$	CPU time (s)				
						z^*	z	z_1	z_2	z_3
5	10	0.013	0.036	0.036	0.003	0.005	0.001	0.033	0.033	0.030
	50	0.000 (7)	0.005	0.005	0.005	111.402 (7)	0.002	0.043	0.035	0.040
	100	0.000 (8)	0.002	0.002	0.001	22.331 (8)	0.012	0.047	0.037	0.037
	500	0.000	0.000	0.000	0.000	0.912	0.035	0.188	0.190	0.067
	1000	0.000	0.000	0.000	0.000	0.053	0.052	0.627	0.628	0.150
10	50	0.004 (5)	0.026	0.026	0.021	239.700 (5)	0.005	0.028	0.032	0.038
	100	0.000 (1)	0.007	0.007	0.003	2.333 (1)	0.008	0.042	0.035	0.035
	500	0.000 (7)	0.000	0.000	0.000	1.238 (7)	0.040	0.178	0.175	0.055
	1000	0.000 (9)	0.000	0.000	0.000	0.066 (9)	0.065	0.627	0.635	0.105
25	50	0.004	0.048	0.048	0.004	0.010	0.007	0.030	0.035	0.043
	100	0.001 (1)	0.047	0.047	0.023	0.100 (1)	0.017	0.040	0.038	0.040
	500	— (0)	0.002	0.002	0.001	— (0)	0.057	0.177	0.182	0.050
	1000	0.000 (4)	0.000	0.000	0.000	0.142 (4)	0.137	0.587	0.592	0.090

Tests were executed on problems with $m = 5, 10, 25$ and $n = 10, 50, 100, 500, 1000$, and processing times in the intervals [1, 100], [1, 1000] and [1, 10,000]. Uniform, as well as truncated exponential and normal distributions were tested, but they exhibit basically the same behavior. All reported results will be for uniformly distributed processing times. We first analyzed the effect of r, the number of intervals for the processing times, on the performance of the algorithm by running several test problems for $r = 1, \ldots, 20$. Basically, a low value of r will produce a good Phase 1 solution, and few passes of Phases 2 and 3; Phase 3 will, however, be computationally expensive. Larger values of r yield worse initial solutions, but will speed up the improvement phases. Overall, we found empirically that the 'best' value of r is related to the ratio (n/m). The 'recommended values' r^* given in Table 1 usually yield a solution value and a computing time very close to the best among all values of r for a given instance.

All problems were then run with $r = r^*$. Unless otherwise indicated, all entries are average values over 10 problems. The only exceptions are when averages are taken over the number of problems solved to optimality and this number is less than 10. The number of successful problems out of 10 is then shown in parentheses in columns 3 and 7 of Tables 3 and 4. The variables used in Tables 2–4 are defined as follows:

z^* : global optimum
z : best solution value found by the 3-PHASE algorithm
z_1 : best solution value found by the 0–1 interchange algorithm [15]
z_2 : best solution value found by the improved 0–1 interchange algorithm [17]
z_3 : best solution value found by the LPT algorithm [8].

The four heuristics were coded in C and all tests were executed on a Sun Sparc 2 station. The MTP bin packing program used as a base for the exact algorithm is written according to PFORT, a portable subset of 1966 ANSI Fortran [20]. A time limit of 1000 s was imposed for the solution of any instance.

Results indicate that the 3-PHASE algorithm produces solution values that are almost always well within 1% of the optimum, when not optimal (see column 3). Moreover, except for one case ($p_j \in [1, 1000]$, $m = 5$, $n = 10$), our algorithm yields average solution values at least as good as those obtained with any of the three alternative heuristics used for the comparison: this is indicated by the nonnegative ratios in columns 4–6. The 3-PHASE algorithm is also always quicker than these heuristics, often by one order of magnitude. Here it is important to point out that the LPT heuristic includes a sorting phase which dominates the execution time; in contrast, 3-PHASE and the two versions of the 0–1 interchange algorithm require no preliminary sorting of processing times. Another observation is that when $p_j \in [1, 100]$, the exact algorithm is just as fast as 3-PHASE; however, this advantage disappears for the other two processing time distributions used in the comparison. In short, the proposed algorithm produces optimal or quasi-optimal solutions in relatively short times and its empirical performance dominates that of the three benchmark heuristics.

Acknowledgements—The work of Paulo M. França was supported in part by Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), and by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil. Michel Gendreau and Gilbert Laporte were founded by the Canadian Natural Sciences and Engineering Research Council under grants OGP0038816 and OGP0039682. Felipe M. Müller was partially supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Brazil. Thanks are due to an anonymous referee for his valuable comments.

REFERENCES

1. R. L. Graham, E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals discr. Math.* **5**, 287–326 (1979).
2. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, Sequencing and scheduling: algorithms and complexity. Report BS-R8909, Center for Mathematics and Computer Science, Amsterdam (1989).
3. T. C. E. Cheng and C. C. S. Sin, A state-of-the-art review of parallel-machine scheduling research. *Eur. J. Opl Res.* **47**, 271–292 (1990).
4. M. R. Garey and D. S. Johnson, Strong NP-completeness results: motivation, examples and implications. *J. assoc. comput. Mach.* **25**, 499–508 (1978).
5. J. Blazewicz, Selected topics in scheduling theory. *Annals discr. Math.* **31**, 1–60 (1987).
6. M. Dell'Amico and S. Martello, Optimal scheduling of tasks on identical parallel processors. Research report OR/90/7. DEIS, University of Bologna (1990).

7. P. Bratley, M. Florian and P. Robillard, Scheduling with earliest start and due date constraints on multiple machines. *Naval Res. Logist. Q.* **22**, 165–173 (1975).
8. R. L. Graham, Bounds on multiprocessing timing anomalies. *SIAM J. appl. Math.* **17**, 416–429 (1969).
9. E. G. Coffman Jr, M. R. Garey and D. S. Johnson, An application of bin-packing to multiprocessor scheduling. *SIAM J. Comput.* **7**, 1–17 (1978).
10. D. K. Friesen, Tighter bounds for LPT scheduling on uniform processors. *SIAM J. Comput.* **16**, 554–560 (1987).
11. D. K. Friesen and M. A. Langston, Evaluation of a MULTIFIT-based scheduling algorithm. *J. Algorith.* **7**, 35–59 (1986).
12. S. K. Sanhi, Algorithms for scheduling independent tasks. *J. assoc. comput Mach.* **23**, 116–127 (1976).
13. D. S. Hochbaum and D. B. Shmoys, Using dual approximation algorithms for scheduling problems: theoretical and practical results. *J. assoc. comput. Mach.* **34**, 144–162 (1987).
14. J. Y.-T. Leung, Bin-packing with restricted piece sizes. *Inform. Process. Letters* **31**, 145–149 (1989).
15. G. Finn and E. Horowitz, A linear time approximation algorithm for multiprocessor scheduling. *BIT* **19**, 312–320 (1979).
16. E. G. Coffman Jr, M. R. Garey and D. S. Johnson, Approximation algorithms for bin-packing—an updated survey. In *Algorithm Design for Computer System Design* (Edited by G. Ausiello, M. Lucertini and P. Serafini), pp. 49–106. Springer, Vienna (1984).
17. M. A. Langston, Improved 0/1 interchange scheduling. *BIT* **22**, 282–290 (1982).
18. D. K. Friesen, Tighter bounds for the MULTIFIT processor scheduling. *SIAM J. Comput.* **13**, 170–181 (1984).
19. S. Martello and P. Toth, *Knapsack Problems*. Wiley, Chichester (1990).
20. B. F. Ryder and A. D. Hall, The PFORT verifier. Computer science report 2, Bell Laboratories (1981).