Sun Yat-sen Management Review
Vol.3, No.1, March 1995

Knapsack-Based Approaches for Scheduling Unrelated Parallel Processors

Chang-Yung Liu
Department of Business Management
National Sun Yet-sen University
Hochi Hwang
Kaohsiang Medical College
(Received January 1995; revised March 1995; accepted April 1995)

Abstract

Parallel processors problems are among the most difficult combinatorial problems to solve. In this paper we study the problem of scheduling n independent jobs on m unrelated parallel processors with the objective of minimizing the makespan. We propose a branch-and-bound algorithm for finding the optimal schedule. This algorithm depends on reducing the set of m constraints to a single diophantine equation, resulting in a knapsack problem. Computational experience indicates that most problems with 5 processors and 30 jobs can be solved in just a few seconds of computer time.

Keywords: Knapsack, Scheduling, Unrelated Parallel Processors.

1. Introduction

This paper presents a branch-and-bound algorithm for the scheduling of a set of n independent jobs on m unrelated parallel processors with the objective of minimizing makespan. The jobs are independent and each must be done by a single processor. Splitting jobs is not allowed and it is assumed that there are no precedence constraints. The processors are nonidentical in the sense that there is no notion of a fast processor always requiring less time than a slow processor, irrespective of that task being executed. Rather, the time required for the execution of a task on a processor is a function of both the task and the processor. An example of this might be in a distributed system where the time requirement of a task on a processor may depend on communication costs.

In [13] this problem is shown to be NP-complete. Hence it seems unlikely that a polynomial time-bounded algorithm, which guarantees an optimal sequence, exists for this problem. The integer programming formulation of this problem is

(P0) Min Z

$$s.t. \sum_{j=1}^{n} t(i,j)x(i,j) \leq Z \quad i = 1, \cdots, m$$

$$\sum_{i=1}^{m} x(i,j) = 1 \quad j = 1, \cdots, n$$

$x(i,j)=\{0,1\}$ for all i and j,

where n is the number of jobs,

m is the number of processors,

t(i,j) is the processing time of job j on processor i,

x(i,j) is an indicator variable that is 1 if processor

i performs job j and 0 otherwise,

Z is the makespan.

Clearly, the exponential-time related enumeration method is the only way of finding the optimal schedule. It is thus worthwhile to design an efficient exact algorithm.

In [11] a polynomial time-bounded algorithm was given to obtain a schedule with the makespan arbitrarily close to the optimal. However, the complexity of the algorithm is $O((1/\varepsilon)n^{2m})$ where ε is the relative error. Ibarra and Kim [12] presented an nlog(n) time-bounded algorithm for m=2 which generates a schedule having a makespan at most $(\sqrt{5}-1)/2$ of the optimal. Lenstra et al. [17] show that no polynomial-time heuristic for this problem has a worst-case performance ratio of less than 3/2 unless P=NP. De and Morton [5] proposed a new heuristic procedure for which the average behavior is within 1.3 percent of the branch-and-bound procedure. Davis and Jaffe [4] also present a number of algorithms with behavior which is $2\sqrt{m}$ times worse than the optimal in the worst case. Liu and Liaw [18] propose two polynomial time bounded algorithms with the addition of an improvement procedure which performs a sequence of job interchanges. Their solution is very close to the optimal.

Although, various heuristic methods for scheduling unrelated parallel processors have been proposed. None of these heuristics dominates all the others in terms of both worst-case performance ratio and time requirements. A report on extensive computational tests to evaluate the effectiveness of these heuristics is provided [10]. Results indicate that, unless an improvement procedure is included, the quality of the heuristic schedules is rather unsatisfactory.

There has been other work on minimizing mean tardiness on parallel processors [9]. Pre-emptive scheduling of tasks on nonidentical processors is studied in [16,8]. In [1] a different optimality criterion is studied. In [20] a man-machine interactive process to swap jobs to minimize the makespan on identical processors is proposed. Dynamic programming and knapsack algorithm approaches are used to solve the identical processor case in [2,21].

In this paper we present a branch-and-bound procedure which finds exact solutions to unrelated parallel processors scheduling problems. Bounding schemes are derived which are motivated by the availability of extremely efficient computing algorithms for the zero-one Knapsack problem. Computational experiments are reported giving indications of limits on problem sizes that can be solved and of how problem parameters influence solution difficulty.

2. The Knapsack Algorithm

The integer programming formulation (P0) indicates that there is at least one of the m processor constraints holds as an equality. Assume it is processor 1, that is
$\sum_{j=1}^{n} t(1,j)x(1,j) = Z$, then (P0) is equivalent to (P1):

$$\text{(P1)} \quad Z = T - \text{Max} \sum_{i=2}^{m} \sum_{j=1}^{n} t(1,j)x(i,j) \tag{1}$$

$$s.t. \sum_{i=2}^{m} \sum_{j=1}^{n} t(1,t)x(i,j) + \sum_{j=1}^{n} t(r,j)x(r,j) \leq T \quad r = 2,\cdots,m \tag{2}$$

$$\sum_{i=2}^{m} x(i,j) \leq 1 \qquad j = 1,\cdots,n \tag{3}$$

$$x(i,j) = \{0,1\} \quad i = 2,\cdots,m \quad j = 1,\cdots,n \tag{4}$$

where $T = \sum_{j=1}^{n} t(1,j)$

Problem (P1) is an integer programming problem and can be solved by a branch-and-bound algorithm. For this approach to be effective, the special structure of the problem must be exploited. If there is a single constraint, then it is a knapsack problem. Current algorithms are capable of solving typical knapsack problems with 60,000 variables in just a few seconds of computer time [3]. Even when the problem has more than one constraint, the knapsack-like structure can be exploited through the use of surrogate constraints.

Let w(2),w(3),...,w(m) be non-negative real numbers, and define

$$t'(i,j) = \sum_{r=2}^{m} w(r)t(1,j) + w(i)t(i,j) \quad i = 2,\cdots,m; \quad j = 1,\cdots,n$$

and $T' = \sum_{r=2}^{m} w(r)T$

The numbers w(2),w(3),....,w(m) are called surrogate multipliers and the

$$\text{theconstraint} \sum_{i=2}^{m} \sum_{j=1}^{n} t'(i,j)x(i,j) \leq T' \tag{5}$$

is called a surrogate constraint. Any solution which satisfies the constraints of (P1) also satisfies (5), but not vice versa. Therefore the problem

$$\text{(P2)} \quad \text{Max} \sum_{i=2}^{m} \sum_{j=1}^{n} t(i,j)x(i,j) \tag{6}$$

$$s.t. \ \sum_{i=2}^{m} \sum_{j=1}^{n} t'(i,j)x(i,j) \leq T' \tag{7}$$

$$\sum_{i=2}^{m} x(i,j) \leq 1 \quad j = 1, \cdots, n \tag{8}$$

$$x(i,j) = \{0,1\} \quad i = 2, \cdots, m; \quad j = 1, \cdots, n \tag{9}$$

provides an upper bound on (P1) [22]. This relaxation is a knapsack problem.

When using (P2) as a relaxation, the tightness of the bound is determined by the surrogate multipliers. The tightest bound is obtained when the surrogate dual is solved, and can be shown to be no worse than the bounds obtained by the linear programming relaxation or the lagrangian relaxation [14]. Methods for finding these multipliers often converge slowly and sometimes exhibit instability [15]. As an approximation, the optimal lagrange multipliers, found by subgradient optimization were used. The interested reader is referred to the paper by Fisher [6] for further details on this procedure.

The linear programming relaxation of (P2) will be used to calculate bounds when solving (P1). Although this will be a weaker bound than solving (P2) itself, the effort required is negligible. Examining more nodes with much less effort per node would likely reduce the solution time. This bound may be tightened by adding any violated constraint of (P1) to (P2), and performing one dual simplex pivot on the resulting two constraint problem. In the spirit of Tomlin [24], this can be done in closed form and the penalty calculated by means of a simple formula.

Note that the best solution to (P2) that is feasible to (P1) is the optimal solution to (P1). Thus, the procedure used to solve (P1) is a modification of the branch-and-bound algorithm of Bulfin, Parker and Shetty [3]. It uses the bounds discussed previously and when a feasible integer solution to (P2) is found, it is checked for feasibility in (P1). If it is feasible to (P1), then rather than fathoming the node, additional variables must be fixed and the procedure continues. In addition, when some variables are fixed at one, all other variables in the same multiple choice constraint can be fixed at zero. This will improve the bound considerably.

To provide an initial incumbent solution or for use by itself if the problem is large, a heuristic is needed. There are two principal objectives that one would like to meet in order to get a quality solution on unrelated processors problem: (1) to make the loads on different processors as equal as possible; (2) to process each job on the processor on which it has a comparative efficiency. Liu & Liaw's heuristic procedure [18] meets both objectives. At the initial stage the heuristic mainly emphasizes efficiency; but as the processors get colse to capacity the emphasis is shifted toward the balance of loads. A reassignment mechanism is added to improve the performance of the heuristic. Therefore, Liu & Liaw's heuristic will be used to calculate the initial incumbent solution for our algorithm.

A statement of the algorithm follows,

Step 0: Apply heuristic procedure of Liu & Liaw to (P1); let Z* be the solution value and X* the value of the variables. Initialize candidate list to consist of (P2), and let all variables be free.

Step 1: If the candidate list is empty, then Stop. The Z* and X* solve (P1). Otherwise choose the problem on the candidate list with the best bound; denote the problem by CP.

Step 2: Solve CPr, the linear programming relaxation of CP, using the algorithm of Sinha & Zoltners [23]. Let the solution value be Zr with variable values Xr.

Step 3: If CPr has no feasible solution or $Zr \leq Z^*$, go to step 2.

Step 4: If some Xr is not integer, go to step 6. If Xr satisfies (7) and (9), go to step 8.

Step 5: Form a 2-constraint linear programming problem by adding the most violated constraint (2) to CPr. A tighter bound than Zr can be obtained by performing one iteration of the dual simplex algorithm [22] on the two constraint problem. If the new bound is no bigger than Z*, go to step 1.

Step 6: Let Z' (Z") be the solution to CPr with the free variable having the largest (smallest) ratio of objective to constraint coefficient fixed at 0 (1). If $Z' < Z"(Z' \geq Z")$, let x(i',j') be the variable with the largest (smallest) ratio.

Step 7: Separate the candidate problem into two new candidate problems. The first consists of CP with the constraints x(i',j')=1 and set x(k,j')=0, k≠i'. The second is CP with x(i',j')=0. These variables are no longer free, but are fixed at the stated value. Place both new problems on the candidate list and go tos tep 1.

Step 8: A new incumbent has been found. Replace Z* by Zr and X* by Xr. Remove all problems from the candidate list with bounds no better than the new Z* and go to step 1.

3. Computatinoal Experience

The previously discussed branch-and-bound algorithm has been tested on a battery of randomly generated problems. Since no computational experience has been reported on this problem, the purpose of this experiment was twofold: to gain some understanding of the computational difficulty of the problem and to examine the performance of the algorithm on problems with various parameters.

This algorithm was tested on problems with 10, 20, 30, 40 and 50 jobs and 2, 5, 10 processors. The solution quality may depend on whether there is any correlation between the rows or columns of the matrix of processing times. To allow for possible variation in performance, the four following classes of test problems were considered, each of which has a different method of generating the processing time t(i,j) of each job j on each processor i.

Uncorrelated (U): t(i,j) is an integer from the uniform distribution [1,100].

processor correlated (P): t(i,j) is an integer from the uniform distribution [a(i)+1,a(i)+20], where a(i) is an integer from the uniform distrubution [0,80].

Job correlated (J): t(i,j) is an integer from the uniform distribution [b(j)+1,b-(j)+20], where b(j) is an integer from the uniform distribution [0,80].

Processor and job correlated (PJ): t(i,j) is an integer from the uniform distribution [a(i)+b(j)+1, a(i)+b(j)+20], where a(i) and b(j) are integers from the uniform distribution [0,100].

For each value of m and n, five problems were generated for each of the four problem classes.

The algorithm was coded in FORTRAN and run on the CONVEX C3840 super computer at the National Sun Yat-sen University. Since a LIFO strategy was used, storage requirements were negligible, and hence computational time is the only criterion reported. Table 1 shows the computational time, in CPU seconds, for each of the sixty problem sets. For each set the minimum, median, and maximum solution times are given. These were chosen since they will provide direct information about the solution difficulty of typical problems with a limited number of sample runs.

Each problem was given a ten second time limit; if the optimal solution was not found, the procedure terminated. On average, the algorithm had no difficulty for any class of 2-processor problems with less than 30 jobs. Problem classes (P) and (PJ) with $m \leq 5$ & $n \leq 30$ or $m \leq 10$ & $n \leq 20$ can be solved in less than 10 seconds.

Table 1 indicates that both correlations influence the computational performance. Problems with correlation between processors (P) are definitely easier to solve. The explanation of this phenomenon, might be that high correlation between the processing times of each processor, tends to generate priority relations between processors that drastically reduce the branches of the tree that must be explored. However, problem class (J) tends to be harder than those without any correlation between jobs. The explanation of this phenomenon might be that a high correlation between the processing times of each job tends to increase the similarity between processors, which will generate an enormous number of feasible schedules with a makespan very close to the optimal and increase the number of nodes of the search tree. The parameter constants (1,100,20,80,etc.) influence the degree of correlation between processors and jobs. Even though problem class (PJ) tends to be more easier to solve than problem class (U), we can not conclude that processor correlated factor (P) is more significant than job correlated factor (J).

4. Concluding Remarks

Unrelated parallel processor problems are among the most difficult combinatorial problems to solve. No direct algorithm has been developed for calculating the optimal makespan or for constructing an optimal schedule. In this paper we have presented a branch-and-bound algorithm for finding the optimal schedule with minimum makespan in an unrelated parallel processors problem. This algorithm is based on the reduction method in which the constraints of an integer programming model is reduced to a single diophantine equation, resulting in a knapsack problem. Because there exists extremely efficient computing algorithms for the zero-one knapsack problem, the proposed approach calculates lower bounds in only a limited time.

Table 1
Computational Time for the Knapsack-Based
Branch-and-Bound Algorithm (in CPU seconds)

	m		2			5			10	
class	n	Min	Median	Max	Min	Median	Max	Min	Median	Max
	10	.001	.003	.008	.010	.022	.132	.072	.273	1.22
	20	.007	.012	.038	.231	.876	2.11	2.31	8.34	–
(U)	30	.012	.090	.286	3.32	6.87	9.33	–	–	–
	40	1.10	2.11	–	–	–	–	–	–	–
	50	–	–	–	–	–	–	–	–	–
	10	.001	.001	.002	.003	.010	.087	.033	.083	.121
	20	.003	.009	.023	.026	.049	.128	.481	.912	2.02
(P)	30	.012	.062	.103	.198	.332	.980	3.99	–	–
	40	.033	.112	.301	1.65	4.48	7.85	–	–	–
	50	.211	.438	1.07	7.44	–	–	–	–	–
	10	.002	.005	.020	.045	.175	1.12	.250	.772	1.11
	20	.023	.082	.217	.880	1.78	2.76	1.87	–	–
(J)	30	.327	1.29	7.45	2.56	6.34	–	–	–	–
	40	4.89	–	–	–	–	–	–	–	–
	50	–	–	–	–	–	–	–	–	–
	10	.001	.003	.009	.009	.020	.980	.065	.123	1.09
	20	.009	.014	.033	.187	.675	1.97	2.23	5.88	9.87
(PJ)	30	.010	.087	.239	1.44	3.44	5.72	–	–	–
	40	.354	.987	5.33	5.98	9.05	–	–	–	–
	50	6.32	–	–	–	–	–	–	–	–

Computational experience indicates that twenty job problems can be solved, and that most problems with 5 processors and 30 jobs are within 10 seconds of computational limits. This is somewhat surprising since these problems have 5^{30} feasible schedules. The results also indicate that processor correlated probllems are easier to solve. Further extension of this research may involve the investigation of the effect of methods of finding surrogate multipliers on the computational efficiency.

References

1. Bruno, J., Coffman, E.G. and Sethi, R., "Scheduling Independent Tasks to Reduce Mean Finishing Time," Commun. ACM, 17, 1974, pp.382–387.
2. Bulfin, R.L., and Parker, R.G., "Scheduling Jobs on Two Facilities to Minimize Makespan," Management Science, 11, 202–213 (1980).
3. Bulfin, R.L., Parker, R.C. and Shetty, C.M., "Computational Results with a Branch-and-Bound Algorithm for the General Knapsack Problem," Naval Research Logistics Quarterly 26, 1979, pp.41–46.
4. Davis, E. and Jaffe, J.M., "Algorithms for Scheduling Tasks on Unrelated Processors," Journal of ACM, 28, 1981. pp.721–736.
5. De, P. and Morton, T.E., "Scheduling to Minimize Makespan on Unequal Parallel Processors," Decision Sciences, 11, 1980, pp.586–602.
6. Fisher, M., "The Lagrangian Relaxation Method for Solving Integer Programming Problems," Management Science, 27, 1981, pp.1–18.
7. Geoffrion, A.M. and Marsten, R.E., "Integer Programming Algorithms: A Framework and State-of-Art Survey," Management Science, 18, 1972, PP.465–491.
8. Gonzalez, T. and Sahni, S., "Preemptive Scheduling of Uniform Processor Systems," Journal of ACM, 25, 1978, pp.92–101.
9. Ho, J.O. and Chang, Y.L., "Heuristics for Minimizing Mean Tardiness for m Parallel Machines," Naval Research Logistics Quarterly, 38, 1991, pp.367–381.
10. Hariri, A.M.A. and Potts, C.N.,"Heuristics for Scheduling Unrelated Parallel Machines," Computers and Operations Research, 18, 1991, pp.323–331.
11. Horowitz, E. and Sahni, S., "Exact and Approximate Algorithm for Scheduling Nonidentical Processors," Journal of ACM, 23, 1976, pp.317–327.
12. Ibarra, O.H. and Kim, C.E., "Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors," Journal of ACM, 24, 1977, pp.280–289.
13. Karp, R.M., "educibility among Combinatorial Problems," In Complexity of Computer Computations, R.E. Miller and J.W. Thatcher, Eds., Plenum Press, New York, 1977, pp.85–103.
14. Karwan, M.H. and Rardin, R.L.,"Some Relationships Between Lagrangian and Surrogate Duality in Integer Programming," Mathematical Programming,17, 1979, pp.320–334.
15. Karwan, M.H. and Rardin, R.L.,"Searchability of the Composite and Mul-

tiple Surrogate Dual Function," Operations Research, 28, 1980, pp.1251–1257.

16. Lawler, E.L. and Labetoulle, J., "On Preemptive Scheduling of Unrelated Parallel Processors by Linear Programming," Journal of ACM, 25, 1978, pp.612–619.
17. Lenstra, J.K., Shmoys, D.B. and Tardos, E., "Approximation Algorithms for Scheduling Unrelated Parallel Machines," Mathematical Programming, 46, 1990, pp.259–271.
18. Liu, C.Y. and Liaw, C.F., "Heuristic Algorithm for Minimizing Makespan on Nonidentical Parallel Processors," Pan-Pacific Conference, Taipei, Taiwan, Proceedings, 1987, pp.859–863.
19. McNaughton, R., "Scheduling with Deadlines and Loss Functions," Management Science, 1, 1959, pp.1–12.
20. Nichols, R.A., Bulfin, R.L. and Parker, R.G., "An Interactive Procedure for Minimizing Makespan on Parallel Processors," International Journal of Production Research, 16, 1978, pp.77–81.
21. Sahni, S., "Algorithms for Scheduling Independent Tasks," Journal of ACM, 23, 1976, pp.116–127.
22. Shapiro, J., Mathematical Programming: Structures and Algorithms, New York, John Wiley and Sons, (1979).
23. Sinha, P. and Zoltners, A., "The Multiple Choice Knapsack problem," Operations Research, 27, 1979, pp.503–515.
24. Tomlin, J.A., "An Improved Branch and Bound Method for Integer Programming," Operations Research, 19, 1971, pp.1070–1075.