J.K.A.U.: Sci., vol. 3, pp. 201-209 (1411 A.H. / 1991 A.D.

A Branch and Bound Algorithm for Job-Shop Scheduling

A.M.A. HARIRI and C.N. POTTS
Department of Statistics, Faculty of Science,
King Abdulaziz University, Jeddah, Saudi Arabia
and
Faculty of Mathematical Studies,
University of Southampton, U.K.

ABSTRACT. The problem of scheduling jobs in a general job-shop to minimize the maximum completion time is considered. A branch and bound algorithm is proposed. The lower bound is obtained from the preemptive schedules which form solutions of single machine subproblems. In the branching rule, a set of operations which each require the same machine is selected and branches of the search tree corresponding to the possibilities that an operation of this set is sequenced before (or after) the others. Computational experience with a variety of test problems is reported.

Introduction

In the job-shop problem, there are m machines (numbered $1, \ldots, m$) which are used to process n jobs (numbered $1, \ldots, n$). Each machine can handle at most one job at a time. Job i($i = 1, \ldots, n$) consists of a sequence of operations $O_{i1}, \ldots, O_{in_i}$ which are to be performed on machines $m_{i1}, \ldots, m_{in_i}$ and which require positive processing times $p_{i1}, \ldots, p_{in_i}$ respectively. Processing starts at time zero. There are technological constraints which specify that operation O_{ij} cannot start until O_{ij-1} is completed ($i = 1, \ldots, n; j = 2, \ldots, n_i$). Preemption of operations is not allowed so that once an operation has started, it must be completed without interruption. The objective is to determine a processing order of operations on each machine which minimizes the maximum completion time.

For many years, the branch and bound algorithm of McMahon and Florian[1] was considered to be the most effective. The enumeration scheme in this algorithm is based on the generation of active schedules to give a search tree for which each node has a corresponding initial partial sequence of jobs on each machine. Lower bounds are obtained from the solution of single machine subproblems in which each job has a release date, a processing requirement on the particular machine under consideration and requires a further specified time for its completion. This single machine problem with release dates is equivalent to the problem of minimizing the maximum lateness and is known to be NP-hard[2]. Nevertheless, McMahon and Florian propose a branch and bound algorithm which solves it fairly efficiently to give a lower bound for the job-shop problem. Many attempts to improve upon this algorithm have ended in failure. For example, the algorithm of Lageweg *et al.*[3], in which at each node of the search tree the branching rule fixes the order between a pair of operations that require the same machine, appears less effective than the active schedules branching. Also, attempts to obtain improved lower bounds using surrogate duality approaches provide, at best, only marginal increases to the single machine bounds at considerable computational expense[4].

Barker and McMahon[5] propose a branch and bound algorithm in which a heuristic method is used to find a feasible schedule at each node of the search tree. A set of consecutively processed operations on some machine is found from this feasible schedule. In an attempt to improve this schedule, either the first job in this set is scheduled to start earlier or an alternative job is sequenced last in this set. Branches of the search tree are constructed for each of these possibilities. For the first set of branches, each job of the set is constrained to be sequenced before the others is given a suitable earliest start time to ensure that processing for this set starts earlier than in the heuristic schedule. For the second set of branches, each job other than the original last job is constrained to be sequenced after the others. The single machine bounds of McMahon and Florian are computed at each node. Computational results indicate that for some problems, this algorithm of Barker and McMahon is more effective than that of McMahon and Florian but for other problems, the McMahon-Florian algorithm yields better results.

Recently, Carlier and Pinson[6] describe a branch and bound algorithm which solves a well-known test problem with 10 jobs and 10 machines: previously, this problem had resisted the attempts of many researchers to solve it. The success of this algorithm is due to the branching rule which selects a set of operations that require the same machine and branches on which operation is sequenced before (or after) the others in this set. However, dominance rules which, at each search tree node, fix the order between various pairs of operations that require the same machine also influence the efficiency of the algorithm.

There has also been much research on heuristic methods which generate approximate solutions. The most notable contribution here is that of Adams *et al.*[7] who obtain high quality solutions to some test problems.

Our aim is to devise a new, more effective, branch and bound algorithm, based on the following observations. Since the single machine bounds of McMahon and Florian are often not close to the optimum, it seems preferable to use a slightly weaker but more quickly computed lower bound. In particular, the preemptive version of the single machine subproblem is easily solved and, in many cases, its value is close to the non-preemptive bound of McMahon and Florian. In terms of a branching rule, the active schedule generation method of McMahon and Florian is rather inflexible since the important decisions that affect the maximum completion time may occur towards the end of the schedule and, hence, their effect will not be apparent until the lower levels of the search tree are reached. This inflexibility is overcome in the schemes of Lageweg *et al.*, Barker and McMahon, and Carlier and Pinson. However, in the first two of these, the set of jobs that determines which branches are added to the search tree does not depend on any lower bounding calculation. We believe that a successful branching rule should be capable of adapting itself to the particular problem under consideration and, furthermore, the components of the schedule which are fixed at any branching should be aimed at increasing the lower bound. This latter effect is most easily achieved by using the results of the lower bounding calculation to decide upon which components should be fixed next.

The paper is organized as follows: Section 2 describes how lower bounds are computed. A complete description of our branch and bound algorithm is given in Section 3, with special emphasis placed upon the branching procedure. Section 4 reports on computational experience with the algorithm and some concluding remarks are contained in Section 5.

Lower Bounds

Our branch and bound algorithm uses a branching rule that fixes the ordering of certain pairs of operations in the search tree. Before describing the lower bounding computations, it is convenient to introduce some terminology. An operation O_{ij} is a predecessor of operation O_{kl} if O_{kl} cannot start until O_{ij} is completed. The technological constraints define some predecessor operations, while the ordering fixed within the search tree provide others. The use of transitivity generates other predecessors. If O_{ij} is a predecessor of O_{kl}, then we refer to O_{kl} as a successor of O_{ij}.

For each operation O_{ij}, we can compute a release date r_{ij} which represents the earliest time that O_{ij} can start and a tail q_{ij} which represents the minimum time that must elapse between the completion of O_{ij} and the completion of all operations. Each operation that has no predecessors is assigned a zero released date. Thereafter, release dates are assigned to operations O_{ij} whose predecessors B_{ij} already have release dates using

$$r_{ij} = \max \left\{ \max_{O_i \setminus j \setminus \in B_i} \left\{ r_i \setminus \setminus + p_i \setminus \setminus + \min_{O_i \setminus j \setminus \setminus \in M_{ij} \cap B_{ij}} \left\{ r_i \setminus \setminus j \setminus \setminus + \sum_{O_i \setminus \setminus \in M_{ij} \cap B_{ij}} p_i \setminus i \setminus \right\} \right\} \right. \quad (1$$

where M_{ij} is the set of operations that require machine m_{ij}. The first term in (1) uses the observation that operation O_{ij} cannot start before all its predecessors are completed, while the second term deems the start time of O_{ij} to be not less than the sum of the earliest start time of some predecessor operation that uses the same machine m_{ij} and the total processing time of such operations. It is clear from (1) that corresponding to r_{ij} is a sequence of predecessor operations such that r_{ij} is equal to the sum of their processing times.

Analogously, each operation which has no successors is assigned a zero tail. We then compute the tails of other operations O_{ij} whose successors A_{ij} already have tails using

$$q_{ij} = \max \left\{ \max_{O_i \setminus_j \setminus \epsilon A_{ij}} \{ q_i \setminus_j + p_{i \ j} \setminus \}, \min_{O_{i \ j} \setminus\setminus \epsilon M_{ij} \cap A_{ij}} \{ q_i \setminus\setminus_i \setminus\setminus + \sum_{O_i \setminus_j \setminus \epsilon M_{ij} \cap A_{ij}} p_i \setminus_j \setminus \} \right\} \quad (2)$$

We observe from (2) that q_{ij} is equal to the sum of processing times of a sequence of operations which are successors of operation O_{ij}.

Let $\{ O_{i_1 j_1}, \ldots, Q_{i_t j_t} \}$ represent the set of all operations that are required to be processed on some machine k. Consider the subproblem in which capacity constraints on all machines other than k are relaxed. For the resulting single machine subproblem, operation $O_{i_s j_s}$ $(s = 1, \ldots t)$ has a release date $r_{i_s j_s}$ when it becomes available for processing, requires a processing time $p_{i_s j_s}$ on the machine and requires a further time $q_{i_s j_s}$ for its completion.

To obtain a lower bound LB_k based on machine k, the preemptive single machine subproblem is solved in which the processing of any operation $O_{i_s j_s}$ $(s = 1, \ldots, t)$ may be interrupted and resumed at a later stage. This preemptive problem is solved by the forward scheduling algorithm of Baker and Su[8] in which no unnecessary machine idle time is allowed. When there is a choice of operations to process, the one having the largest tail is chosen. Also, if during the processing of an operation another operation with a larger tail becomes available, then processing is interrupted at the release date of the latter operation. Having computed all single machine bounds, the value $\mathrm{LB} = \max \{ \mathrm{LB}_1, \ldots, \mathrm{LB}_m \}$ is used as a lower bound in the search tree.

The Branch and Bound Algorithm

We first describe the branching rule that is used in the branch and bound algorithm. Branching aims to enforce feasibility either through the elimination of preemption or through the elimination of overlaps (an overlap occurs when two operations which require the same machine are processed at the same time) in the single machine subproblem based on a machine k for which $\mathrm{LB} = \mathrm{LB}_k$. The branching rule selects a set of operations S which require the same machine and constructs branches according to which operation of S is sequenced before (or in some cases after) the others. Thus, it remains to specify how the set S is chosen and whether the first or the last operation of S should be fixed by branching.

To determine S, if no preemption occurs in the solution of the single machine subproblem based on machine k, then any branching is based on the elimination of overlaps. Otherwise, if some job is preempted, we first attempt to find a non-preemptive schedule for the subproblem based on machine k in which all jobs are completed by time LB. The following forward sequencing heuristic of Schrage[9] is applied in which no unnecessary machine idle time is allowed. When there is a choice of operations, one with the largest tail is scheduled. If some job is completed after time LB in the heuristic schedule, then branching is aimed at the elimination of preemption in the preemptive schedule for this subproblem. Let O_{ij} denote the first operation that is preempted in this single machine subproblem and let S denote the set of operations, including O_{ij} itself, which are scheduled on machine k after the start but before the completion of O_{ij}. Branching fixes which operation of S is sequenced before the others.

If there is no preemption or if Schrage's heurisitic produces a non-preemptive schedule for which the value LB is achieved, the set S is based on the following concept of overlaps. Implicit in the non-preemptive single machine schedule is a schedule for the job-shop problem which, in general, is not feasible. In this job-shop schedule, the completion times $C_{i_s j_s}$ of operations $O_{i_s j_s}$ $(s = 1, \ldots, t)$ on machine k are defined by the single machine schedule. Operations O_{ij} which do not require machine k are assigned completion times as follows. Firstly, temporary release dates $r_{i_s j_s} = C_{i_s j_s} - j_{i_s j_s}$ are assigned to operations $O_{i_s j_s}$ $(s = 1, \ldots, t)$. Then, for operation O_{ij} which has the property that there is some later operation O_{il} $(l > j)$ that requires machine k, its (earliest) completion time is determined using

$$C_{ij} = r_{ij} + p_{ij}$$

Similarly, all other operations which do not require machine k are assigned (latest) completion times using

$$C_{ij} = \text{LB} - q_{ij}$$

If we are fortunate, these completion times define a feasible job-shop schedule. However, often, there is infeasibility because more than one operation is processed on a machine in some time interval. Infeasibility is detected if, for operations O_{ij} and $O_{i'j'}$ which require the same machine, their completion time satisfy $C_{ij} + p_{i'j'} > C_{i'j'}$ and $C_{i'j'} + p_{ij} > C_{ij}$. We define the overlap between these operations as $\min \{ C_{ij}, C_{i'j'} \} - \max \{ C_{ij} - p_{ij}, C_{i'j'} - p_{i'j'} \}$; for pairs of operations which are not processed at the same time, the overlap is defined as zero. Overlaps can be regarded as a measured of infeasibility of the schedule.

To determine S when branching is based on overlaps, assume that jobs are renumbered so that the sequence of operations on machine k is the non-preemptive schedule is $(O_{i_1 j_1}, \ldots, O_{i_t j_t})$. Let the lower bound be given by

$$\text{B} \quad r_i \qquad \Sigma \quad p_i \quad + q_{i_v j_v}$$

for some u and v satisfying $1 \leq u \leq v \leq t$. If possible, we select from the sequence of operations corresponding to $r_{i_u j_u}$, an operation O_{ij} which has a positive overlap with at least one other operation and set S to be the set of operations which has a positive overlap with O_{ij}. Otherwise, this attempt to find S is repeated, based on overlaps with an operation from the sequence corresponding to $q_{i_v j_v}$. If this fails to generate S, a similar procedure is applied which searches for overlaps with one of the operations in the sequence corresponding $r_{i_{u-1}, j_{u-1}}$ (if $u \neq 1$), $r_{i_{u+1}, j_{u+1}}$, $q_{i_{v+1}, j_{v+1}}$, (if $v \neq t$) and $q_{i_{v-1}, j_{v-1}}$. If S is still not determined, we choose S to be the set of operations which have a positive overlap with some operation Q_{ij}, where O_{ij} is chosen to give $|S|$ as large as possible: if there are no overlaps, then the schedule is feasible. If S is generated from overlaps with an operation in the sequence corresponding to $q_{i_{v-1}, j_{v-1}}$, q_{i_v, j_v} or $q_{i_{v+1}, j_{v+1}}$, then branching fixes the operation which is sequenced last amongst those of S; otherwise, branching fixes the operation which is sequenced first amongst those of S.

We next describe the remaining details of our branch and bound algorithm. At each node of the search tree a release date r_{ij} and a tail q_{ij} are computed using (1) and (2) for each operation O_{ij}. If it is found that $r_{ij} + p_{ij} + q_{ij} \geq \text{UB}$, where UB is an upper bound provided by the best feasible solution currently found, then the node is discarded. Otherwise, for any pairs of operations O_{ij} and $O_{i\backslash j\backslash}$ requiring the same machine that satisfy $r_{ij} + p_{ij} + p_{i\backslash j\backslash} + q_{i\backslash j\backslash} \geq \text{UB}$, where O_{ij} is not a predecessor of $O_{i\backslash j\backslash}$ then $O_{i\backslash j\backslash}$ is deemed to be a predecessor of O_{ij} at this node. Two heuristic solutions are computed at each node of the search tree. A forward sequencing heuristic schedules operations starting at time zero. The operation to be scheduled next has its predecessors already scheduled and is chosen from those which do not leave any unnecessary machine idle time so that its tail is as large as possible. A backward sequencing heuristic first finds the problem inverse by interchanging release dates and tails and interchanging predecessor and successor operations; the forward sequencing heuristic is then applied to the inverse. A lower bound LB on the maximum completion time is then computed as described in Section 2. If $\text{LB} < \text{UB}$, then the set S of operations is found on which branching is based.

Our search strategy is one that, in the initial stages of branching, selects a node with the smallest lower bound from which to branch. However, since this approach requires much computer storage space, a newest active node search is adopted when available storage is used. The newest active node search branches from the most recently created active node.

Computational Experience

Test problems were generated for various values of m and n. In these sets of problems, there are 6 in which $m \in \{4, 6, 8\}$ and $n \in \{20, 40\}$ where m is small compared with n, there are 3 in which $m = n = 6$, $m = n = 8$ and $m = n = 10$ and there are 9 in which $m \in \{20, 30, 40\}$ and $n \in \{4, 6, 8\}$ where m is large compared with n. For all problems, each job is processed once on each machine. Processing times p_{ij} ($i = 1, \ldots, n$; $j = 1, \ldots, m$) were generated from the uniform distribution [1,100]. For job i($i = 1, \ldots,$

n), the machines $m_{i1}, \ldots, m_{im}$ required for operations $O_{i1}, \ldots, O_{im}$ were obtained from a randomly generated permutation of the integers $1, \ldots, m$. For each value of (m, n), 10 problems were generated.

The algorithm was coded in FORTRAN 77 and run on a CDC 7600 computer. Whenever a problem was not solved within the time limit of 250 seconds, computation was abandoned. Computational results listing median computation times in seconds, median numbers of nodes and numbers of unsolved problems for each (m, n) are shown in Table 1. For the (m, n) values (8, 20), (10, 10), (30, 8) and (40, 8), there are at least 5 unsolved problems, so the medians cannot be computed.

We first observe from Table 1 that our algorithm solves over 75% of the 180 test problems within the time limit. The performance of the algorithm is reasonable for small m and large n and for small n and large m. It is apparent, however, that problems in which $m = n$ are extremely difficult: when $m = n = 10$, none of the test problems are solved within the time limit. There is some evidence to suggest that for fixed m, where $m < n$, problems become easier as n becomes larger.

In addition to the computational tests described above, the algorithm was applied to the two test problems in[10] with $m = 5$ and $n = 20$, and with $m = n = 10$. The performance of our algorithm on these problems is disappointing since neither was solved within 1000 seconds. The best solution found for the 20-job problem has a

TABLE 1. Computational results.

m	n	MCT	MNN	NU
4	20	0.03	1	1
4	40	0.07	1	0
6	20	8.10	89	4
6	40	0.96	3	0
8	20	–	–	7
8	40	5.42	10	2
6	6	0.36	62	0
8	8	31.93	1890	0
10	10	–	–	10
20	4	0.14	14	0
20	6	2.51	80	0
20	8	75.72	1541	2
30	4	0.22	12	0
30	6	4.76	103	0
30	8	–	–	5
40	4	0.49	19	0
40	6	9.11	149	0
40	8	–	–	8

MCT: median computation time in seconds.
MNN: median number of nodes.
NU : number of unsolved problems.

maximum completion time of 1215, whereas the optimal value is 1165. For the 10-job problem, the best solution found has value 981 compared with the optimal value of 930. However, an alternative algorithm in which the branching rule is based on the division of time intervals within which an operation must be processed (details of this approach are given by Carlier[11]) solves the 20-job problem using 782 search tree nodes and generates a solution of value 946 for the 10-job problem. Due to poor results obtained from initial experiments with other test problems, this alternative approach was abandoned.

Concluding Remarks

Our proposed branch and bound algorithm attempts to make the decisions which have a major effect on the maximum completion time towards the top of the search tree. For a reasonable number of test problems this strategy successfully limits the search tree to a few nodes. Even with this improved branching rule, however, there are a significant number of test problems that remain unsolved.

It appears that a new approach is required to obtain much tighter lower bounds which effectively restrict the search for the harder classes of problems in which m does not differ greatly from n. The commonly used lower bounds obtained from single machine subproblems are far too weak for these harder classes of problems.

Acknowledgement

The research by the first author was supported by a grant from King Abdulaziz University, Jeddah, Saudi Arabia.

References

[1] **McMahon, G.B.** and **Florian, M.**, On scheduling with ready times and due dates to minimize maximum lateness, *Oper. Res.* **23:** 475-482 (1975).

[2] **Lenstra, J.K.**, **Rinnooy Kan, A.H.G.** and **Brucker, P.**, Complexity of machine scheduling problems, *Ann. Discrete Math.* **1:** 343-362 (1977).

[3] **Lageweg, B.J.**, **Lenstra, J.K.** and **Rinnooy Kan, A.H.G.**, Job-shop scheduling by implicit enumeration, *Management Sci.* **24:** 441-450 (1977).

[4] **Fisher, M.L.**, **Lageweg, B.J.**, **Lenstra, J.K.** and **Rinnooy Kan, A.H.G.**, Surrogate duality relaxation for job-shop scheduling, *Discrete Appl. Math.* **5:** 65-75 (1983).

[5] **Barker, J.R.** and **McMahon, G.B.**, Scheduling the general job-shop, *Management Sci.* **31:** 594-598 (1985).

[6] **Carlier, J.** and **Pinson, E.**, An algorithm for solving the job-shop problem, *Management Sci.* **35:** 164-176 (1989).

[7] **Adams, J.**, **Balas, E.** and **Zawack, D.**, The shifting bottleneck procedure for job-shop scheduling, *Management Sci.* **34:** 391-401 (1988).

[8] **Baker, K.R.** and **Su, Z.S.**, Sequencing with due-dates and early start times to minimize maximum tardiness, *Naval Res. Logist. Quart.* **21:** 171-176 (1974).

[9] **Schrage, L.**, *Obtaining optimal solutions to resource constrained network scheduling problems*, unpublished manuscript (1971).

[10] **Fisher, H.** and **Thompson, G.L.**, Probabilistic learning combinations of local job-shop scheduling rules, *in:* **Muth, J.F.** and **Thompson, G.L. eds.**, *Industrial Scheduling*, Prentice-Hall, Englewood Cliffs, New Jersey (1963).

[11] **Carlier, J.**, Scheduling jobs with release dates and tails on identical machines to minimize makespan, *European J. Oper. Res.* **29:** 298-306 (1987).

أسلوب[1] فرعٍ وحَدٍّ لمشكلة تنظيم الأعمال في ورشة تعاقدية

أحمد محمد حريري و كريس ن. بوتس

قسم الإحصاء ، كلية العلوم ، جامعة الملك عبد العزيز ، جدة ، المملكة العربية السعودية

و كلية الدراسات الرياضية ، جامعة ساوثمبتون ، المملكة المتحدة

المستخلص . المشكلة محل البحث عبارة عن تنظيم الأعمال في ورشة عمل تعاقدية ، حيث يتوافر عدد n من الأعمال المطلوب إنجازها على M آلة . كل عمل من الأعمال يمر على الآلات بترتيب محدد قد يختلف عن غيره من الأعمال ، ولمدة زمنية محددة على كل آلة منها ، ليتم إنجازه عليها . اقترحنا أسلوب فرعٍ وحَدٍّ لحل هذه المشكلة . الحد الأدنى المستخدم تم الحصول عليه بحل مشكلة تنظيمٍ على آلة واحدة في حين يعتمد أسلوب التفريع المتبع على اختيار مجموعة من العمليات (كل عملية تمثل انجاز عمل على آلة معينة) التي تتطلب نفس الآلة لإنجازها . كل عقدة في شجرة التفريع تمثل احتمال إنجاز أحد العمليات في هذه المجموعة قبل بقية العمليات في نفس المجموعة . قمنا كذلك بكتابة برنامج كمبيوتر بلغة فورتران لاختبار فعالية أسلوب الفرع والحد المقترح على مشاكل اختبار مختلفة .

(١) ترجمت كلمة (Algorithm) بكلمة (الخوارزمة) في مجمع اللغة العربية ، معجم مصطلحات الحاسب الآلي (الكمبيوتر)