Contents

12
Flow shops

Jan Karel Lenstra
Centrum Wiskunde & Informatica

David B. Shmoys
Cornell University

A printing office has to produce a number of books. Each book needs to be typeset, printed and bound, in this order. The typesetter, the printer and the binder can each handle one book at a time. How should they proceed so as to finish the job as fast as possible? Is it evident that the books should be handled in the same ordering at each stage?

This is an example of a 3-machine flow shop. In general, n jobs J_j $(j = 1,\dots,n)$ have to be processed by m machines M_i $(i = 1,\dots,m)$. Each job is first processed on machine M_1, then on M_2, $\dots$, and finally on M_m. The processing time of operation O_{ij} of J_j on M_i is given by p_{ij}. We wish to determine a schedule that minimizes the makespan. In short, $F||C_{\max}$.

12.1. Johnson's algorithm for $F2||C_{\max}$

In one of the first results in deterministic scheduling theory, Selmer M. Johnson gave an $O(n\log n)$ algorithm to solve $F2||C_{\max}$. The algorithm is surprisingly simple: first arrange the jobs with $p_{1j} \leq p_{2j}$, in order of nondecreasing p_{1j} and then arrange the remaining jobs in order of nonincreasing p_{2j}.

The proof that this algorithm produces an optimal schedule is also straightforward. Notice that the algorithm produces a *permutation schedule*, where the order of the jobs on each machine is identical. An easy interchange argument shows that there exists an optimal schedule that is a permutation schedule. For a permutation schedule, it is easy to see that $C_{\max}$ is determined by the processing time of some ℓ

jobs on machine 1, followed by the processing time of $(n+1)-\ell$ jobs on machine 2. Note that this implies that if each p_{ij} is decreased by the same value p, then for each permutation schedule, $C_{\max}$ decreases by $(n+1)p$. Finally, observe that if $p_{1j}=0$, then job J_j is scheduled first in some optimal schedule, and similarly, if $p_{2j}=0$, then J_j is scheduled last is some optimal schedule. Putting these pieces together, we see that an optimal schedule can be constructed by repeatedly finding the minimum p_{ij} value among the unscheduled jobs and subtracting this value from all processing times; if the minimum is zero, then the corresponding job is scheduled. This algorithm is clearly equivalent to the one given above.

12.2. Approximation algorithms: simple results

Given that all but the most restrictive cases of the flow shop scheduling problem are NP-hard, it is natural to consider approximation algorithms for the problem. We will consider algorithms that produce permutation schedules, as well as algorithms that produce arbitrary schedules. In the former case, we must be careful to measure the performance relative to the optimal schedule rather than to the optimal permutation schedule.

In contrast to, for example, $O||C_{\max}$, simple algorithms for $F||C_{\max}$ do not give encouraging results. Consider the permutation schedule produced by a *list scheduling* algorithm (LS); in other words, consider an arbitrary permutation schedule. In such a schedule, at least one operation is being performed at any time, so that $C_{\max}(\text{LS}) \leq \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}$. On the other hand, the optimal schedule length $C_{\max}^*$ must be at least $\sum_{j=1}^{n} p_{ij}$, for any $M_i (i=1,\ldots,m)$. There must be a machine for which this sum is at least $(\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij})/m$, since that is the average machine load. We have obtained the following result.

Theorem 12.1. *For any instance of* $F||C_{\max}$, $C_{\max}(LS) \leq mC_{\max}^*$.

The reader is invited to show that the analysis of algorithm LS is tight; see Exercise 12.1.

Johnson's algorithm for $F2||C_{\max}$ can be used to give simple algorithms with somewhat better performance. For example, suppose that the machines are grouped in $\lceil m/2 \rceil$ pairs: (M_1,M_2), (M_3,M_4), and so forth; if m is odd, then M_m is grouped by itself. We can find an optimal schedule for each pair; clearly, the length of each of these schedules is no more than $C_{\max}^*$. By concatenating the $\lceil m/2 \rceil$ two-machine schedules, we obtain an m-machine schedule of length at most $\lceil m/2 \rceil C_{\max}^*$. Note that this is not necessarily a permutation schedule.

A similar sort of *aggregation* algorithm (AG) produces a permutation schedule with the same performance guarantee. Aggregate $M_1,\ldots,M_{\lceil m/2 \rceil}$ into one virtual machine, and $M_{\lceil m/2 \rceil+1},\ldots,M_m$ into another, with processing times $a_j = \sum_{i=1}^{\lceil m/2 \rceil} p_{ij}$ and $b_j = \sum_{i=\lceil m/2 \rceil+1}^{m} p_{ij}$. We can view this as an instance of $F2||C_{\max}$ and apply Johnson's algorithm to obtain a permutation π. This permutation can be viewed as a schedule for the original problem; how good is it?

One important fact for this analysis: the length of the permutation schedule π for the original m-machine problem is no more than the length of π for the aggregated two-machine problem. If we have an aggregated operation on the virtual machine 1 being processed for a_j time units, then allocating this block of time across the first half of the machines, is sufficient to process each of the component operations of job j in the corresponding interval.

We complete the analysis by showing that there is a schedule for the aggregated two-machine problem of length at most $\lceil m/2 \rceil C^*_{\max}$ (and hence the approximation for the m-machine problem is at most this much). Consider an optimal schedule for the m-machine problem. Group together M_i and $M_{\lceil m/2 \rceil + i}$, for $i = 1, ..., \lfloor m/2 \rfloor$; if m is odd, then $M_{\lceil m/2 \rceil}$ is grouped by itself. For each group, partition the schedule for its machines into time intervals in such a way that there is a one-to-one correspondence between the right endpoints of those intervals and the completion times of operations on machines in that group. If the intervals generated for all of the groups are sorted by their right endpoints and concatenated in that order, we obtain a preemptive schedule for the two-machine problem. Note that the sorting guarantees that the preemptive schedule is feasible for the two-machine problem; more precisely, no job is completed on the first virtual machine after it is started on the second. The length of this schedule is obviously no more than $\lceil m/2 \rceil C^*_{\max}$. But it is straightforward to argue that there is a nonpreemptive permutation schedule of the same length. Thus, we have proved the following theorem.

Theorem 12.2. *For any instance of $F||C_{\max}$, the aggregation algorithm AG delivers a permutation schedule with $C_{\max}(AG) \leq \lceil m/2 \rceil C^*_{\max}$.*

Again, the analysis of algorithm AG is tight; see Exercise 12.2. While Theorem 12.2 gives an upper bound on the quality of permutation schedules relative to the overall optimum, there are classes of instances for which the ratio between the lengths of the optimal permutation schedule and the overall optimal schedule is unbounded; see Exercise 12.3. In the next section, we will give a deeper result on the absolute difference between these two optima.

Many more approximation algorithms for flow shop scheduling have been proposed than the ones that we have analyzed in this section. In general, the objective has been to design methods that do well on a set of randomly generated instances, not to establish performance guarantees. In other words, the emphasis has been on empirical rather than worst-case analysis. In the rest of this section, we will mention some of these heuristics and summarize the experimental results that have been reported. We note that all of these methods produce permutation schedules.

We first discuss three algorithms that *construct* a schedule from scratch. The *slope index* rule orders the jobs J_j in order of nonincreasing values $\sum_{i=1}^{m} [i - (m+1)/2] p_{ij}$. It is based on the same idea as Johnson's algorithm: a job whose operations tend to increase in length should be placed at the beginning of the schedule, and a job whose operations decrease in length should be at the end. The *generalized aggregation* rule uses Johnson's method in a more direct way: for $l = 1, ..., m-1$, apply the two-machine algorithm using processing times $\sum_{i=1}^{l} p_{ij}$ and $\sum_{i=m+1-l}^{m} p_{ij}$ ($j = 1, ..., n$), and evaluate the resulting permutation as an m-machine schedule; choose the best of these $m-1$ schedules. Note that this algorithm is not a proper generalization of algorithm AG. The *longest insertion* rule is as follows: start with the empty partial schedule; at each step, select an unscheduled job with the maximum total processing requirement, and insert it in the current partial schedule in the position that minimizes the increase in schedule length.

There is ample empirical evidence to support the statement that the slope index rule is outperformed by the generalized aggregation rule, and that the longest insertion rule does still better. The latter algorithm is the current champion among constructive rules. These increases in solution quality come at the expense of increases in running time, as the reader can easily verify.

Another type of approximation algorithm applies *local search* to a given schedule. Here, one has to define a *neighborhood* for each schedule, i.e., a set of schedules that can be generated by a simple perturbation. Examples of perturbations are a transposition of two adjacent jobs, a transposition of two jobs that are not necessarily adjacent, and a shift of a job to a different position. In the basic variant of local search, the neighorhood of the initial schedule is searched for a better schedule. If such an improvement is found, the process restarts from there, and this continues until a schedule is found that is optimal in its neighborhood. A recent variant of local search is *simulated annealing*, which accepts deteriorations with a small and decreasing probability in an attempt to avoid bad local optima and to get settled in a global optimum. We will discuss this approach in more detail in Chapter 14.

Computational experiments suggest that the adjacent transposition neighborhood is too restrictive. Local search that starts with an arbitrary solution and applies adjacent transpositions tends to take more time and produce worse schedules than longest insertion. A longest insertion schedule can, however, often be improved by applying transpositions and shifts. Not surprisingly, simulated annealing with the shift neighborhood gives even better results but needs more time.

Exercises

12.1. Prove that the bound for list scheduling given in Theorem 12.1 is tight.

12.2. Prove that the bound for the aggregation algorithm given in Theorem 12.2 is tight.

12.3. Consider the following flow shop instance consisting of $2n$ machines and n jobs: $p_{n-j+1,j} = p_{n+j,j} = 1, (j = 1,...,n)$, and $p_{ij} = 0$ for all other (i, j); here, 0 really denotes some arbitrarily small positive value.

(a) Prove that $C^*_{\max} = 2$.

(b) Prove that any permutation schedule has $C_{\max} \geq \sqrt{n}$. (*Hint:* A beautiful theorem of Erdös and Szekeres states that in any sequence of $n^2 + 1$ distinct integers, there is a decreasing subsequence of length $n+1$ or an increasing subsequence of length $n+1$. Use this theorem to identify a time-consuming subsequence of operations in any permutation schedule.)

(c) Use the insight gained in part (b) to construct a permutation schedule with $C_{\max} = 2\lceil\sqrt{n}\rceil$.

12.3. Approximation algorithms: geometric methods

The most significant results on approximation algorithms for flow shop scheduling are based on geometric methods. The main tool we will use is Theorem 12.4, the vector sum theorem, which was proved in the previous chapter.

The following result was already mentioned in Section 13.1. Let $p_{\max} = \max_{i,j} p_{ij}$.

Theorem 12.3. *For any instance of $F||C_{\max}$, a permutation schedule of length at most $C^*_{\max} + m(m-1)p_{\max}$ can be found in $O(m^2n^2)$ time.*

Proof. Consider an arbitrary flow shop instance. For each M_i, let $\Pi_i = \sum_{j=1}^{n} p_{ij}$, and let $\Pi_{\max} = \max_i \Pi_i$. We will show that Theorem 12.4 provides a way to construct a permutation schedule of length at most $\Pi_{\max} + m(m-1)p_{\max}$. Since $\Pi_{\max} \leq C^*_{\max}$, the theorem then follows.

Without loss of generality, we may assume that $\Pi_i = \Pi_{\max}$ for each M_i. If $\Pi_i < \Pi_{\max}$, we can iterate through the operations on M_i, increasing each p_{ij} to at most $p_{\max}$, until the revised total reaches $\Pi_{\max}$.

Consider a permutation π of $\{1,...,n\}$, and suppose that each machine processes the jobs in the order $J_{\pi(1)},...,J_{\pi(n)}$. Let I_{ij} denote the total idle time of M_i up to the starting time of $O_{i\pi(j)}$. For a permutation schedule, it is easy to calculate all of the values I_{ij}. On M_1, we clearly have $I_{1j} = 0$ for all j. On M_i $(i \geq 2)$, operation $O_{i\pi(j)}$ starts as soon as both $O_{i\pi(j-1)}$ and $O_{i-1,\pi(j)}$ are completed. In case the former operation finishes later, we have

$$I_{ij} = I_{i,j-1}.$$

In the latter case, $I_{ij} + \sum_{k=1}^{j-1} p_{i\pi(k)} = I_{i-1,j} + \sum_{k=1}^{j} p_{i-1,\pi(k)}$, or

$$I_{ij} = I_{i-1,j} + p_{i\pi(j)} + \sum_{k=1}^{j} (p_{i-1,\pi(k)} - p_{i\pi(k)}).$$

Suppose that π is such that the following bound holds:

$$\sum_{k=1}^{j} (p_{i-1,\pi(k)} - p_{i\pi(k)}) \leq (m-1)p_{\max}, j = 1,...,n, i = 2,...,m. \tag{12.1}$$

This implies that the idle times are related by the recurrence

$$I_{ij} \leq \max\{I_{i,j-1}, I_{i-1,j} + mp_{\max}\}.$$

With the initial conditions for $i = 1$, we conclude that $I_{mn} \leq (m-1)mp_{\max}$. This yields the desired bound on the length of the permutation schedule corresponding to π:

$$C_{\max}(\pi) = \Pi_m + I_{mn} \leq \Pi_{\max} + m(m-1)p_{\max}.$$

Fortunately, Theorem 12.4 is exactly what we need to compute a permutation that guarantees the inequality (12.1). Define the vectors

$$v_j = (p_{1j} - p_{2j}, p_{2j} - p_{3j}, \ldots, p_{m-1,j} - p_{mj}), j = 1, \ldots, n.$$

The assumption that $\Pi_i = \Pi_{\max}$ $(i = 1, \ldots, m)$ implies that $\sum_{j=1}^{n} v_j = 0$, so we can apply Theorem 12.2. Since $d = m - 1$ and $||v_j|| \leq p_{\max}$, (12.1) immediately follows. This completes the proof. □

It is natural to ask whether the bound given in Theorem 12.3 is tight. This is certainly not the case for $m = 2$, since there is a permutation schedule that is an overall optimal schedule. One might also wonder if $\Pi_{\max} + 2p_{\max}$ is a tight bound for $m = 2$; no, one can show that there always is a (permutation) schedule that achieves $\Pi_{\max} + p_{\max}$, and this is tight. For $m = 3$, there is a permutation schedule that is guaranteed to be no longer than $\Pi_{\max} + 3p_{\max}$, and this is tight; for $m = 4$, the best bound known is $\Pi_{\max} + 9p_{\max}$.

When the number of machines is fixed, Theorem 12.3 can also be used to obtain strong relative performance guarantees. Let ε be an arbitrary positive constant. Call a job J_j *big* if there is an operation O_{ij} with $p_{ij} > \varepsilon\Pi_{\max}/m^2$, and *small* otherwise. Observe that there are less than m^3/ε big jobs. (Otherwise, at least one machine would have to process at least m^2/ε of the expensive operations, which would contradict the definition of $\Pi_{\max}$.) Hence, for fixed m and ε, the number of big jobs is bounded by a constant.

Now, first use the algorithm of Theorem 12.3 to schedule all of the small jobs. The length of this partial schedule is at most

$$\Pi_{\max} + m(m-1)\varepsilon\Pi_{\max}/m^2 < (1+\varepsilon)\Pi_{\max} \leq (1+\varepsilon)C^*_{\max}.$$

Next, consider all possible ways to schedule all of the big jobs at the conclusion of this partial schedule. Since there are only a constant number of big jobs and a constant number of machines, the number of such extensions is also constant (albeit possibly huge), and the best one can be selected in constant time. Since the completed schedule is at most $C^*_{\max}$ longer than the partial schedule, we obtain a schedule of length at most $(2+\varepsilon)C^*_{\max}$.

Theorem 12.4. *For any instance of $Fm||C_{\max}$ (i.e., with m fixed) and any $\varepsilon > 0$, a schedule of length at most $(2+\varepsilon)C^*_{\max}$ can be found in polynomial time.*

Note that the extension need not be a permutation schedule, and hence the completed schedule need not be one. However, if only permutation extensions are considered, then the same scheme delivers a solution of length at most $2+\varepsilon$ times the length of the best permutation schedule. It remains an interesting open question to find algorithms with comparable performance whose running time is also polynomial in m.

Exercises

12.4. Consider $F2||C_{\max}$. Give a polynomial-time algorithm for finding a permutation schedule of length at most $\Pi_{\max} + p_{\max}$. Show that this bound is tight.

Notes

13.1. *The two-machine flow shop.* The two-machine flow shop algorithm is one of the most celebrated results in scheduling theory. It is due to Selmer M. Johnson (1954).

13.2. *Approximation algorithms: simple results.* Potts, Shmoys, and Williamson (1991) exhibit a family of flow shop instances for which the best permutation schedule is longer than the true optimal schedule by a factor of more than $\frac{1}{2}\sqrt{m}$.

The analysis of algorithm LS is due to Gonzalez and Sahni (1978A). They showed that the bound of Theorem 12.1 is tight even for LPT schedules, in which the jobs are ordered according to nonincreasing sums of processing times. They also gave the algorithm that consists of $\lfloor m/2 \rfloor$ applications of Johnson's algorithm. Algorithm AG was proposed and analyzed by Röck and Schmidt (1983).

The slope index rule is due to Palmer (1965), and the generalized aggregation rule to Campbell, Dudek, and Smith (1970). Dannenbring (1977) compared eleven heuristics, including those of Palmer and Campbell et al. as well as two that apply adjacent transpositions. The longest insertion rule was proposed by Nawaz, Enscore, and Ham (1983) and tested by Turner and Booth (1987). Osman and Potts (1989) report on experiments with traditional local search and simulated annealing.

We quote a few results that are not as simple as the title of this section suggests. In a series of papers, Nowicki and Smutnicki (1989, 1991, 1993) analyzed the performance of the algorithms mentioned in the previous paragraph. They are interested in the worst-case ratio of the length of the approximate schedule and the length of the optimal *permutation* schedule. The generalized aggregation rule has a ratio $\lceil m/2 \rceil$. For the slope index rule and for both adjacent transposition algorithms of Dannenbring, the ratio is about $m/\sqrt{2}$ (the precise value is a more complicated expression that depends on the parity of m). These bounds are tight. The worst-case ratio for the longest insertion rule is unknown, but instances exist that achieve a ratio of about $\sqrt{m}/2$.

Finally, Potts (1985B) investigated the performance of five approximation algorithms for $F2|r_j|C_{\max}$. The best one of these, called RJ', involves the repeated application of a dynamic variant of Johnson's algorithm to modified versions of the problem, and satisfies $C_{\max}(RJ')/C^*_{\max} \leq 5/3$; this bound is tight.

13.3. *Approximation algorithms: geometric methods.* The connection between flow shop scheduling and the vector sum theorem, as stated in Theorem 13.3, was discovered independently by Belov and Stolin (1974), Sevastyanov (1974), and Bárány (1981). Theorem 13.4 is implicit in the work of Shmoys, Stein, and Wein (1994).