Ideal Preemptive Schedules on Two Processors

E. G. Coffman, Jr.* J. Sethuraman† V. G. Timkovsky‡

November 8, 2002

Abstract

An ideal schedule minimizes both makespan and total flow time. It is known that the Coffman-Graham algorithm [*Acta Informatica* **1**, 200–213, 1972] solves in polynomial time the problem of finding an ideal nonpreemptive schedule of unit-execution-time jobs with equal release dates and arbitrary precedence constraints on two identical parallel processors. This paper presents an extension of the algorithm that solves in polynomial time the preemptive counterpart of this problem. The complexity status of the preemptive problem of minimizing total flow time has been open.

1 Introduction

We describe a preemptive algorithm for scheduling n unit-execution-time (UET) jobs with equal release dates on two identical parallel processors under a general partial order specifying precedence constraints. The algorithm is preemptive in that it is allowed to interrupt an executing job and resume it later on the same or the other processor; such algorithms are not restricted in the number of times they can preempt jobs. An *ideal* schedule simultaneously minimizes both makespan and total flow time. Our main contribution is a preemptive algorithm for finding an ideal schedule in polynomial time. Note that the complexity status of the preemptive problem of minimizing total flow time has been open.

An algorithm is said to be non-preemptive if all jobs must run to completion without interruption. The earliest polynomial-time algorithms for scheduling UET jobs with general precedence constraints on two identical processors were devoted to minimizing makespan. These are the preemptive Muntz-Coffman (MC) algorithm [11] and the nonpreemptive Coffman-Graham (CG) algorithm [4]. These algorithms bear directly on the problem here; polynomial-time algorithms were obtained later for the more general problems of minimizing the maximum lateness [5, 7]. Other cognate problems include the preemptive and nonpreemptive

*Department of Electrical Engineering, Columbia University, New York, NY 10027, egc@ee.columbia.edu.

†Department of Industrial Engineering & Operations Research, Columbia University, New York, NY 10027, jay@ieor.columbia.edu (corresponding author).

‡CGI Group Inc., 1 Dundas St. W., Ste. 2700, Toronto, Ontario M5L 1G1, timko@cgi.ca; and Department of Computing & Software, McMaster University, Hamilton, Ontario L8S 4L7.

versions of minimizing the number of late jobs, total tardiness, and total weighted flow time which are all known to be NP-hard even for chain-like precedence constraints [9, 10, 16, 1].

The MC algorithm schedules jobs level-by-level highest level first, where the level of a job is the number of edges in a longest path from that job to a job having no successors. If the currently highest level has more than two jobs, then they are scheduled by the preemptive wrap-around rule. If there are exactly two jobs at the highest level, they are run nonpreemptively in parallel. If there is just one, it is run nonpreemptively with a highest (necessarily lower) level available job, if any. Figure 1 shows that the MC algorithm does not always minimize total flow time.

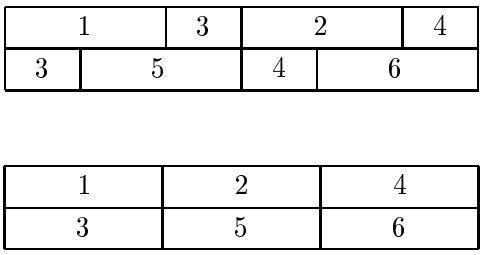


Figure 1: For the instance with six jobs and precedence constraints $1 \to 2$, $3 \to 4$, $5 \to 6$, the MC algorithm finds the upper schedule with total flow time $1 + 2 \cdot 1.5 + 2.5 + 2 \cdot 3 = 12.5$ applying the wrap-around rule to each of the job sets $\{1, 3, 5\}$ and $\{2, 4, 6\}$. The lower schedule, however, has a smaller total flow time $2 \cdot 1 + 2 \cdot 2 + 2 \cdot 3 = 12$.

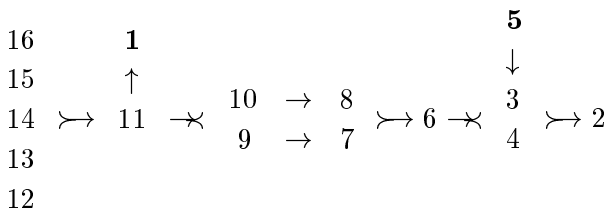
The above algorithms suggest two natural approaches to finding an ideal preemptive schedule: Attempt to extend the preemptive MC algorithm so that it minimizes total flow time while preserving minimum makespan, or attempt to introduce preemptions into the nonpreemptive CG algorithm so that it minimizes total flow time and reduces makespan to that of the MC algorithm. The process of creating a preemptive ideal schedule appears to be much easier starting from a CG schedule than from an MC schedule partly because, as observed by M. R. Garey (private communication), the CG algorithm minimizes total flow time for the nonpreemptive case. Although this property has been known for a long time, we are not aware of any published proof. In this paper, we sketch a proof, leaving routine details to the reader.

The main body of the paper is devoted to an extension of the CG algorithm that yields ideal preemptive schedules. In Section 2, we discuss properties of CG schedules, in particular, the partition of CG schedules into components called *follow-set* schedules which must appear in sequence. In Section 3, we give an algorithm for finding an ideal preemptive follow-set schedule. By appropriately iterating this algorithm on an entire CG schedule, we will obtain in Section 4 an algorithm that computes ideal preemptive schedules. Concluding remarks outline related unsolved problems.

2 CG schedules

CG schedules are ideal schedules for the non-preemptive version of the problem studied here; their structure will play a critical role in the design and proof of correctness of an algorithm for finding ideal preemptive schedules. For this reason, we first review the CG algorithm and note some useful properties.

Given an instance containing n jobs, the CG algorithm first labels each of the jobs using the integers $1, 2, \ldots, n$ as follows: Suppose labels $1, 2, \ldots, k$ have already been assigned, and let S be the subset of unlabeled jobs, all of whose successors have been labeled. Then, assign label $k+1$ to a job in S having the lexicographically smallest decreasing sequence of immediate-successor labels. (Break ties arbitrarily.) The CG schedule is obtained by scheduling, at each point in time, the highest-labeled available job. Since each job is executed for one time unit, processors 1 and 2 both become available at the same time. By convention, we assume that processor 1 is scheduled before processor 2; thus, at any given time, the label of the job executing on processor 1 is greater than the label of the job executing on processor 2. An idle slot, necessarily on processor 2, is assumed to have an artificial job with label 0. Except for these jobs with label 0, jobs are identified uniquely by their labels. See Figure 2 for examples.



16	14	12	11	10	8	6	4	2
15	13	**5**		9	7	**1**	3	

Figure 2: A precedence graph with 16 jobs, where "$\rightarrowtail\!\!\!\!\prec$" and "$\succ\!\!\!\!\rightarrowtail$" denote a set of edges with a common source and a common sink, respectively, and a related CG schedule with four follow-sets: $F_3 = \{16, 15, 14, 13, 12\}$, $F_2 = \{11\}$, $F_1 = \{10, 9, 8, 7, 6\}$, $F_0 = \{4, 3, 2\}$, and two fill-jobs: $E_3 = 5$, $E_1 = 1$ (shown in bold).

To prove that the CG schedule has minimum makespan for the non-preemptive version of the problem, Coffman and Graham [4] determined sets of jobs F_k, $F_{k-1}, \ldots, F_0$ such that *every* job in F_i precedes *every* job in F_{i-1}, for $i = 1, 2, \ldots, k$. Thus, any feasible schedule must finish all jobs in F_i before starting any job in F_{i-1}. The F_i are the *follow-sets* alluded to at the end of the last section; the term is borrowed from Sethi [13]. The collection of follow-sets is such that any schedule that executes these follow-sets in order will trivially be

at least as long as the CG schedule.

The follow-sets of a CG schedule are associated with pairs (D_k, E_k), $(D_{k-1}, E_{k-1}), \ldots,$ (D_0, E_0), where D_i is a job executing on processor 1 and E_i is either a gap in parallel with D_i on processor 2, or a job executing in parallel with D_i on processor 2. The pairs are identified by working backward from the end of the schedule. Let (D_0, E_0) be the last pair of the CG schedule on processors 1 and 2 respectively. Suppose (D_{i-1}, E_{i-1}), $(D_{i-2}, E_{i-2}), \ldots,$ (D_0, E_0), have been defined; then E_i is the latest job on processor 2 smaller than D_{i-1} and scheduled earlier than E_{i-1}; and D_i is the job on processor 1 that is scheduled at the same time as E_i. Using these pairs, follow-set F_i consists of D_i, and all jobs scheduled after the pair (D_{i+1}, E_{i+1}) and before the pair (D_i, E_i); in this context, when E_i is a job, and not a gap, it is referred to as a *fill-job* accompanying follow-set F_i; it fills what would otherwise be a gap created by the precedence relation. See Figures 2 and 2 for an illustration. Trivially, there is always at least one follow-set, but the last one, F_0, in contrast to all others, can have no job with a successor not in F_0.

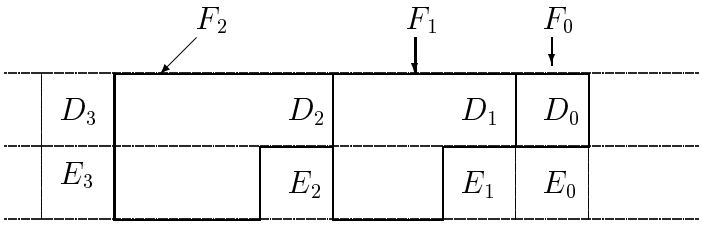


Figure 3: Follow-sets in CG schedules.

We record some useful observations about the CG schedule in the following lemma; all the statements in the lemma are easy to verify and are present in some form in Coffman and Graham [4].

Lemma 1 *Consider a CG schedule with the job pairs* $(D_k, E_k), (D_{k-1}, E_{k-1}), \ldots, (D_0, E_0)$ *determining follow-sets* $F_k, F_{k-1}, \ldots, F_0$. *Then for* $i = 1, 2, \ldots, k$,

(a) job E_i *is smaller than any job in* F_{i-1}, *so it does not precede any job in* F_{i-1}.

(b) every job in F_i *precedes every job in* F_{i-1};

(c) for each F_i *there exist integers* ℓ *and* u, $(u \geq \ell)$ *such that* F_i *consists of* **all** *jobs from some level* u *to some level* l $(l \leq u)$ *that were not already scheduled as fill-jobs; also,* $|F_i|$ *is odd.*

We conclude this section with a proof that a CG schedule has minimum total flow time. The notion of compressed schedule is useful for this purpose. A schedule for an instance of a scheduling problem is *compressed* [14] if, among all schedules for the instance, it contains

the minimum total idle time in the interval $[0, t]$, for every integer $t \geq 0$. In the case of nonpreemptive scheduling of UET jobs, the minimum total idle time in $[0, t]$ is achieved by a schedule that maximizes the number $N[t]$ of jobs scheduled in $[0, t]$.

The total flow time of a conservative nonpreemptive schedule of UET jobs (one that never allows a machine to be idle if there is a job ready to execute) is given by

$$\sum_{t=1}^{n} t \cdot (N[t] - N[t-1]) = n^2 - \sum_{t=1}^{n-1} N[t],$$

where $N[0] = 0$ and $N[n] = n$. Hence, a compressed schedule obviously has the minimum total flow time because it maximizes $N[t]$ for every $t = 1, \ldots, n-1$. Informally, the property of being compressed means that gaps are shifted to the right as much as possible, and each gap is unavoidable. It is intuitively clear, and not difficult to prove by induction on the number of gaps, that CG schedules are compressed [14].

3 Contracting follow-sets

Using a single preemption, a makespan-2 nonpreemptive schedule of three independent jobs can be shortened to a makespan-3/2 preemptive schedule. This creation of a flat preemptive schedule will hereafter be called a *contraction*; it always involves just three jobs. Note for future reference that the sum of the completion times is unchanged by a contraction (see Figure 4).

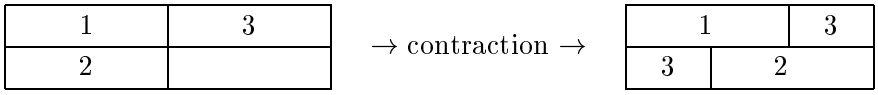


Figure 4: A makespan-2 nonpreemptive schedule and a makespan-3/2 preemptive schedule of jobs 1, 2, 3.

Contraction can be extended to the follow-sets of CG schedules not accompanied by fill-jobs. After showing how this is done, we will show how to generate ideal preemptive schedules using this contraction operation as a primitive.

In a given CG schedule, consider a follow-set whose last job, say J, executes in parallel with a gap, and suppose the follow-set is scheduled in $[t_0, t]$. The following algorithm contracts the schedule for the follow-set, thus reducing its makespan by 1/2, if and only if it is possible to do so without violating precedence constraints. For concreteness, let F_i be the follow-set, so that $J = D_i$

> **Follow-set Contraction (FC) Algorithm.** The algorithm first determines whether J is the only job in F_i, i.e., F_i is a *singleton follow-set*. If so, it halts since the schedule trivially cannot be contracted. If not, then it next determines

whether J is the only level-l job in F_i; if it is, the algorithm halts with the decision that F_i cannot be contracted. To see that this is the correct decision, it is enough to note that all of the jobs scheduled in $[t_0, t-1]$ must have J as their unique level-l successor; otherwise, another level-l job would be available for scheduling by the CG algorithm in $[t-1, t]$, and this contradicts our assumed gap in $[t-1, t]$.

The algorithm next checks the two jobs scheduled in $[t-2, t-1]$. (Two such jobs must exist, since F_i must have at least three jobs.) If these jobs are both level-l jobs, then they and J comprise three independent jobs and the schedule for these jobs is contracted as described earlier. This schedule is appended to the nonpreemptive schedule in $[t_0, t-2]$ to produce a flat schedule for F_i, whereupon the algorithm halts.

The only remaining case to consider is when F_i has more than one level-l job, but there are not two of them in $[t-2, t-1]$. In this case, the algorithm finds the latest time interval $[t'-1, t']$ with $t' < t$ in which a level-l job K is paired with a job K' at a level l' for some $l' > l$. (Such an interval must exist, because there are at least two level-l jobs.) It is easy to verify that all of the jobs scheduled in $[t', t-1]$ will have to precede job J (as J is the only level-l job scheduled after t'), and succeed K'. Thus, an attempt to contract the schedule for F_i can do no better than to move the level-l job K from $[t'-1, t']$ to $[t-1, t]$, and apply the algorithm recursively to the schedule in $[t_0, t']$. That is, letting the level-l' job K' play the role of J, and replacing t with t', the algorithm then proceeds recursively on the smaller schedule in $[t_0, t']$.

Continuing then, if K' is the only level-l' job in F_i, then F_i cannot be contracted. If the two jobs scheduled immediately prior to K' are also level l' jobs, then F_i can be contracted by contracting the schedule for the three jobs in $[t'-2, t']$, and then moving up by 1/2 the jobs (including K now) scheduled after t'. Otherwise, the algorithm finds a level-l' job paired with a level-l'' job, and recurses again.

By convention, if the FC algorithm concludes that a follow-set F_i cannot be contracted, we set its output to be F_i itself. See Figure 5 for illustrations.

The following result is an easy consequence of the comments given in the FC algorithm description.

Lemma 2 *In a given CG schedule, let F_i be a follow-set whose last job executes in parallel with a gap. Applying the FC algorithm to F_i contracts F_i if and only if it is possible to do so within the precedence constraints. Moreover, the FC algorithm produces an ideal preemptive schedule for F_i.*

Proof. First we note that preemptions are useless in the segments of the schedule that the FC algorithm could not contract; such segments contain $2(t-t')$ jobs that must precede a

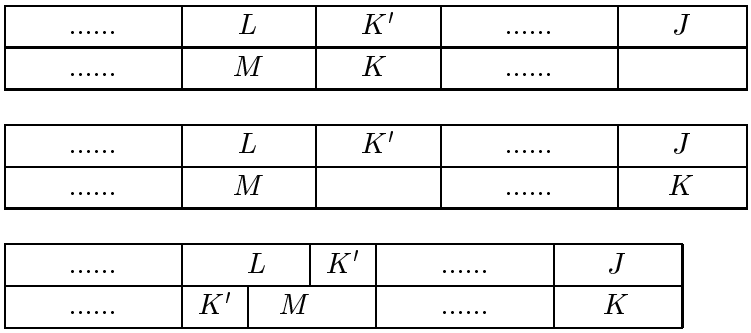


Figure 5: Contracting a follow-set.

job scheduled at time t and succeed a job scheduled at time t'. The FC algorithm therefore produces a minimum-makespan preemptive schedule.

For total flow time, suppose $|F_i| = 2m - 1$. A simple calculation shows that total flow time of F_i in the CG schedule is m^2, which is minimum (see McNaughton [12]). Since the output of the FC algorithm is a contracted version of F_i or F_i itself, and since contraction preserves total flow time of the jobs in the follow-set, the outcome of the FC algorithm is a schedule with total flow time m^2. ■

Note that if J is the last job in follow-set F_i, and if F_i cannot be contracted, then J must be a last job in every feasible schedule of F_i. This will be useful in proving the correctness of the algorithm for finding an ideal preemptive schedule, which is described below.

4 Contracting schedules

This section proves that the Schedule Contraction (SC) algorithm presented below produces preemptive ideal schedules. The SC algorithm proceeds by contracting follow-sets whenever that is possible and, at the same time, helps in eliminating a gap.

> **The SC Algorithm.** Suppose the CG schedule for some instance has follow-sets $F_k, F_{k-1}, \ldots, F_0$, with corresponding fill-jobs or gaps $E_k, E_{k-1}, \ldots, E_0$ occupying the respective final time units on processor 2 in the CG schedule. The SC algorithm begins by searching for an earliest gap. If the schedule has no gap, then the algorithm halts, so assume that there is at least one gap and let E_j be the earliest one. Then the algorithm searches for an earliest follow-set that can be contracted and is scheduled no later than F_j. (The FC algorithm is invoked for this purpose.) Suppose such a follow-set exists, let it be F_i, $i \geq j$, and let its last job finish at time t. The algorithm first creates a *reduced* problem, which consists of the fill-job E_i, the jobs in follow-sets $F_{i-1}, F_{i-2}, \ldots, F_0$, and their corresponding fill-jobs $E_{i-1}, E_{i-2}, \ldots, E_0$. The SC algorithm starts out by scheduling $F_k, \ldots, F_{i+1}$ along with their fill-jobs just as the CG algorithm does. It then appends a flat schedule

for F_i. Finally, the SC algorithm finishes off the schedule with an ideal preemptive schedule for the reduced problem, this last part of the schedule being found by a recursive application of the SC algorithm.

If no follow-set F_i, $i \geq j$, can be contracted, the reduced problem identified by the algorithm consists of the jobs in follow-sets $F_{j-1}, F_{j-2}, \ldots, F_0$, and their corresponding fill-jobs $E_{j-1}, E_{j-2}, \ldots, E_0$. The SC algorithm proceeds just as the CG algorithm up to the time instant at which follow-set F_j completes; the SC algorithm appends at this point an ideal preemptive schedule for the reduced problem found as before by a recursive application of the SC algorithm.

In either case, the algorithm halts when the reduced problem is empty.

Since any contracted follow-set contains at least three jobs and only one preempted job, the total number of preemptions the algorithm makes is at most $n/3$. Our main result is the following:

Theorem 1 *For any given instance, the SC algorithm produces an ideal preemptive schedule.*

Proof. The proof is by induction on the number of gaps in the SC schedule S for instance I. The basis of the induction, which proves that SC schedules with no gaps are ideal, presents the most difficulty, so we will begin with the induction step.

Induction step: The argument hinges on the following "separator" lemma.

Lemma 3 *Let the SC schedule S have at least one gap, suppose the earliest ends at time t, and denote the job executing in $[t-1, t]$ by J. Then J is a separator in the sense that it must be scheduled in $[t-1, t]$ in every ideal preemptive schedule.*

Remark: Given this lemma, the induction step is relatively easy to establish, as follows. Let I' denote the sub-instance consisting of just those jobs scheduled after t in S plus the precedence relations restricted to these jobs. By the inductive hypothesis, the preemptive schedule of the jobs executing after time t in S is ideal in the sense that the schedule from t onwards is an ideal preemptive schedule for I'; since every job of I' must succeed J, and since J must be executed in $[t-1, t]$, the schedule from $(t-1)$ onwards is an ideal preemptive schedule for $I' \cup \{J\}$. Now, consider the schedule of the set of jobs in $[0, t-1]$ in S; this is a schedule for the $2(t-1)$ jobs in the set $I'' \equiv I \setminus \{I' \cup \{J\}\}$. By the induction basis, it is an ideal preemptive schedule for I''. (It trivially has a minimum makespan as it is gap free, but we need the inductive hypothesis for total flow time.) Scheduling a job in I'' after time $(t-1)$ would only increase total flow time, and may increase makespan of the schedule as well; this shows that S is an ideal preemptive schedule for I. ■

Proof of Lemma 3. We begin with a couple of observations. The SC algorithm does not create gaps. A gap G in a SC schedule was the earliest gap found in the CG schedule for

the original instance or for some subsequent reduced instance, depending on the level of the recursion when G was processed; G could not be eliminated at that time because no follow-set in parallel with or earlier than G could be contracted. (G remains thereafter since the next reduced problem is drawn from the schedule to the right of G.)

Consider the iteration of the SC algorithm at level $\ell+1 \geq 1$ of the recursion, and let $F_i^{(\ell)}$ and $E_i^{(\ell)}$ denote the follow-sets and fill-jobs in the reduced instance defined at level ℓ (in the original instance if $\ell = 0$). Suppose $E_j^{(\ell)}$ is the earliest gap, and let $F_i^{(\ell)}$, $i \geq j$, be the earliest follow-set that can be contracted. Then this reduced instance consists of the job $E_i^{(\ell)}$, the jobs in $F_{i-1}^{(\ell)}, F_{i-2}^{(\ell)}, \ldots, F_0^{(\ell)}$, and their corresponding fill-jobs $E_{i-1}^{(\ell)}, E_{i-2}^{(\ell)}, \ldots, E_0^{(\ell)}$. It is easy to see that the CG schedule for this instance will have the following structure:

- The $i-j$ sets $F_{i-1}^{(\ell)}, F_{i-2}^{(\ell)}, \ldots, F_{j+1}^{(\ell)}, F_j^{(\ell)}$, are scheduled in that order, with the last job of each set accompanied by one of the $i-j$ jobs in $\{E_i^{(\ell)}, E_{i-1}^{(\ell)}, \ldots, E_{j-1}^{(\ell)}\}$. Indeed, we know that, since a fill-job can not precede any job in the follow-set scheduled right after it, the fill-jobs $\{E_i^{(\ell)}, E_{i-1}^{(\ell)}, \ldots, E_{j-1}^{(\ell)}\}$ can each be slid down to the next follow-set, with the last one being slid into the gap $E_j^{(\ell)}$.

- The j sets $F_{j-1}^{(\ell)}, F_{j-2}^{(\ell)}, \ldots, F_0^{(\ell)}$, are scheduled in that order, with the last job of $F_m^{(\ell)}$ accompanied by $E_m^{(\ell)}$, for $m = j-1, j-2, \ldots, 0$.

In particular, note that the earliest gap E_j of the original CG schedule has been eliminated by contracting the follow-set F_i. While this operation results in a reduced problem whose CG schedule is not contained in the original CG schedule, note that the original and reduced CG schedules are identical after the jobs in F_j are scheduled. (Of course, the SC schedule corresponding to the reduced CG schedule starts 1/2 time unit earlier).

We are now ready to prove the lemma. If J is accompanied by a gap in the SC schedule, then we know that J must have been accompanied by a gap in the original CG schedule, and in every reduced CG schedule computed during the course of the SC algorithm. In particular, in reduced problems containing J, all the jobs in every follow-set examined by the SC algorithm for contraction are required to precede J. Since the SC algorithm was unable to eliminate the gap in $[t-1, t]$, we know that for some reduced instance found during the course of the SC algorithm, both of the following must be true: (i) The gap accompanying job J is the earliest gap; (ii) all follow-sets preceding job J (if any) and the follow-set containing job J as the last job can not be contracted. Clearly, every follow-set examined by the SC algorithm involving jobs scheduled earlier was either contracted, or could not be contracted. Since all the jobs in every one of these follow-sets were required to precede J, we conclude, by Lemma 2, that the follow-set containing job J cannot be scheduled to start any earlier; in addition, since J must be part of a follow-set that can not be contracted, J has to be scheduled as a last job in that follow-set. This shows that J cannot be scheduled to start any earlier than $(t-1)$. Delaying the start of job J will increase makespan and total flow time of the overall schedule, so J must be scheduled in $[t-1, t]$ in every ideal preemptive schedule. ■

Induction basis: We now prove the basis of the induction, when the SC schedule S for the given instance I has no gaps. Before getting into details, we will comment briefly on the proof.

The proof is based on the expectation that the precedence constraints involving fill-jobs and those involving jobs in the same follow-set have no effect on total flow time. This key simplification is proved implicitly in the analysis below, which focuses on the instance I^* of the *relaxed* problem in which such precedence constraints have been removed from I. Thus, each fill-job is independent of all other jobs, and the jobs in any follow-set are all independent of each other. (But all jobs of a follow-set must continue to precede all jobs in subsequent follow-sets.)

Lemma 6 below proves the existence of a unique ideal preemptive schedule for I^* that begins with g contracted follow-sets, where g is the number of gaps in the CG schedule for I. This schedule differs from the SC schedule of I only in where contractions are made, not their number. Lemma 4 below verifies that such differences have no effect on total flow time. Lemma 5 is a "normal form" result that simplifies the arguments of Lemma 6.

As before, F_i, $k \geq i \geq 0$, denotes the i-th follow-set of the CG schedule for the original instance I, and $n_i = |F_i|$. The following "interchange" lemma simplifies substantially the arguments in the remainder of the proof.

Lemma 4 *Consider a preemptive schedule for a relaxed instance. Let F_i execute before F_j and let t and t' be the starting times of F_i and F_j, respectively. Suppose F_i executes nonpreemptively with fill-job E and F_j executes preemptively as a contracted follow-set. Note that $t' \geq t+(n_i+1)/2$ and that F_j finishes at $t'+n_j/2$. Assume that all jobs, if any, executing in the interval $[t+(n_j+1)/2, t']$ both start and finish in that interval.*

Now, assuming that F_i can be contracted, interchange the modes of execution: Execute F_i preemptively by contracting it, and de-contract F_j, executing its jobs nonpreemptively along with the fill-job E in the last time unit of the schedule for F_j. (Note that the move of E is allowable since it is independent of all other jobs in the relaxed instance. See Figure 6 for an illustration.) The interchange leaves total flow time of the schedule unchanged.

Proof: The interchange does not affect the completion times of jobs executed before F_i starts or after F_j finishes. In accordance with observations made when introducing contractions, the contraction of follow-set F_i does not change total flow time of its jobs. The jobs executing originally in $[t+(n_i+1)/2, t']$, say there are n of them, all have their completion times reduced by $1/2$. Inspection of Figure 7 shows that the interchange reduces by $1/2$ the completion times of all but one of the jobs in F_j. Thus, among the jobs considered so far, there is a total reduction of total flow time of $n \times 1/2 + (n_j - 1) \times 1/2 = (n + n_j - 1)/2$. It remains to observe that, in moving from F_i to F_j, the fill-job E increases its completion time by $(n + n_j - 1)/2$, as can be seen in the figure. The lemma is proved, as the net effect on

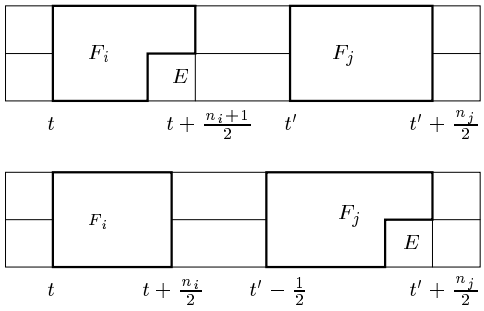


Figure 6: Interchange the modes of execution of F_i and F_j.

total flow time is 0. ■

In what follows, we consider an ideal preemptive schedule for a relaxed instance whose corresponding CG schedule has $k+1$ follow-sets $F_k, F_{k-1}, \ldots, F_0$. We let t_i denote the earliest time at which all jobs of F_i complete, and we let y_i be the total execution time devoted to jobs that execute but do not complete in $[t_{i+1}, t_i]$. The following result narrows down considerably the possible schedules we need to consider.

Lemma 5 *In any ideal preemptive schedule S^* for a relaxed instance, the following holds:*

- *For any i, $k \geq i \geq 0$, and $t \in [t_{i+1}, t_i]$, at least one of the processors is working on a job (either a fill-job or a follow-set job) that completes in $[t_{i+1}, t_i]$.*
- *for all i, $y_i \leq 1$.*

Proof: If the first assertion were violated, then there would be a segment of S^* in some interval $[t_{i+1}, t_i]$ during which no job that finishes in $[t_{i+1}, t_i]$ is executing. Since the instance is relaxed, the fill-jobs executing in this segment are independent of all other jobs, so this segment can be shifted to the end of the interval without increasing any completion times and without violating precedence constraints. But this reduces t_i thus contradicting our assumption that the schedule is ideal and hence minimizes total flow time.

For convenience, assume processor 1 only executes jobs that complete in $[t_{i+1}, t_i]$. To prove the second assertion, suppose $y_i > 1$. All of the jobs executing in $[t_{i+1}, t_i]$ are independent so it is easy to see that the y_i units of partial execution time must be scheduled last on processor 2 (otherwise, we could shift some partial execution time to the right and reduce at least one completion time). But this leads to a contradiction just as before. We can exchange the first unit of partial execution time on processor 2 with the last job, say J, to complete on processor 1. It is readily verified that we can sequence the partially executed jobs so that the exchange gives us a valid schedule with a smaller completion time for J and with all other completion times unchanged. (See Figure 7 for an illustration.) ■

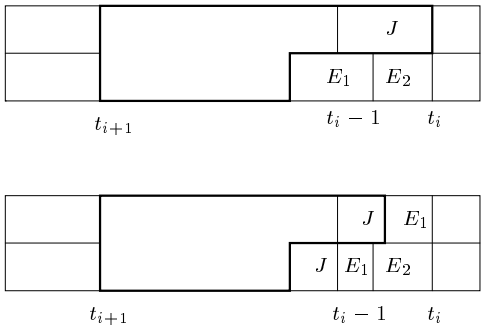


Figure 7: Interchange a portion of J with a portion of E_1.

We are now at the heart of the proof where we show that S^* has a structure similar to that of a SC schedule.

Lemma 6 *For an instance I^* of the relaxed problem whose (non-preemptive) CG schedule has g gaps and whose SC schedule has no gaps, the preemptive schedule that contracts the first g non-singleton follow-sets is ideal and lexicographically minimizes $(t_k, t_{k-1}, \ldots, t_1, t_0)$.*

Remark: Note that the lex-min schedule of Lemma 6 may not be a feasible schedule for the original instance. However, both the lex-min schedule S^* for the relaxed instance and the SC schedule S for the original instance have the same number of contractions. So repeated use of Lemma 4 on S^* shows that it has the same total flow time as S. Thus, once we have proved Lemma 6, we will be done.

We remark that the total flow time of S^* is given by the simple formula $t_0^2 + t_0 + g/4$, which the reader will have no difficulty in deriving.

■

Proof of Lemma 6: We proceed by contradiction and let I^* be an instance with the fewest number of jobs for which the lemma is false. Then, I^* must be such that its first follow-set could be contracted, but is not by any ideal preemptive schedule. Let S^* be an ideal preemptive schedule for I^* that lexicographically minimizes the (decreasing) sequence of follow-set completion times.

Let S_i^* denote the segment of S^* in the time interval $[t_{i+1}, t_i]$. (We set $t_{k+1} = 0$ by convention.) Bear in mind that, because of the precedence constraints, S_i^* cannot contain jobs of F_j, $j < i$, so S_i^* contains all n_i jobs of F_i and some number, say $r_i \geq 0$, of fill-jobs that complete in $[t_{i+1}, t_i]$ plus some set, possibly empty, of jobs that execute only partially during $[t_{i+1}, t_i]$ and take total time $0 \leq y_i \leq 1$ by Lemma 5. All of these jobs are independent. Observe that to prove the claim, it suffices to show that $r_k = 0$ and $y_k = 0$ in S^*, as this implies that F_k is contracted.

For the purposes of the transformation below, we assume for simplicity that there are no

singleton follow-sets; extending the arguments to these uncontractible special cases is easy and left to the reader. Let $F_j,\ j \leq k$, be the earliest follow-set with $r_j \geq 1$. Transform S^* by replacing S_j^* with a segment S_j^{**} constructed as follows (see Figure 8 for an illustration):

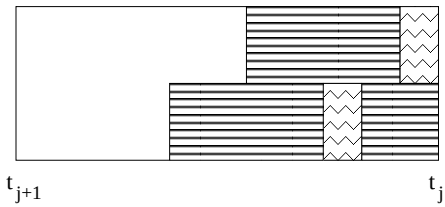


Figure 8: Transformation: the unshaded areas represent the jobs in F_j, horizontal shading represents the fill-jobs that complete by t_j, and the remaining areas represent the partially completed fill-jobs.

If r_j is odd, let processor 1 execute $\lceil n_j/2 \rceil$ jobs of F_j, followed by $\lfloor r_j/2 \rfloor$ fill-jobs, followed by $y_j/2 \leq 1/2$ units of time spent on the partially executed fill-jobs; and let processor 2 execute $\lfloor n_j/2 \rfloor$ jobs of F_j, followed by $\lfloor r_j/2 \rfloor$ fill-jobs, followed by $y_j/2 \leq 1/2$ units of time spent on partially executed fill-jobs, followed by the remaining fill-job completed in $[t_{j+1}, t_i]$. If r_j is even, let processor 1 execute $\lceil n_j/2 \rceil$ jobs of F_j, followed by $r_j/2 - 1$ fill-jobs, followed by $1/2 \leq (1+y_j)/2 \leq 1$ units of time spent on another (complete) fill-job E; and let processor 2 execute $\lfloor n_j/2 \rfloor$ jobs of F_j, followed by $r_j/2 - 1$ fill-jobs, followed by $0 \leq (1 - y_j)/2 \leq 1/2$ units of time spent on the job E, followed by one fill-job, followed by $y_j \leq 1$ units spent on the partially executed fill-jobs.

It is easy to verify that the total flow time of the transformed schedule is no more than that of schedule S^*. Moreover, if y_j is strictly positive or if $r_j > 1$, the transformed schedule is an ideal preemptive schedule with a lexicographically smaller $(t_k, t_{k-1}, \ldots, t_1, t_0)$, contradicting our choice of S^*. Thus, $y_j = 0$ and $r_j = 1$. Let the completing fill-job be E. (We note in passing that it would be easy to show that none of the follow-sets $F_k, \ldots, F_{j+1}$ could be a singleton. For, if there were one, say $F_i,\ i > j$, then we could easily arrange to complete the fill-job E in S_i^* rather than S_j^*.)

Now suppose there is a smallest i larger than j such that $y_i > 0$. Then we modify the schedule so that F_i is contracted (thus reducing t_i), and S_j^* is replaced by S_j^{**} as in the procedure above, with the partially executed fill-jobs being taken from S_i^*. Again, the new schedule contradicts the lex-min property of S^*. Thus, $y_k = y_{k-1} = \cdots = y_j = 0$ and $r_j = 1$.

If $j < k$, then, by definition, $r_k = 0$, so since $y_k = 0$, we are done. If $j = k$, then S^* first schedules F_k along with E, followed by an ideal preemptive schedule for the remaining jobs. By our assumption of a smallest counterexample, a lex-min schedule for the remaining jobs can be found by contracting the next g follow-sets that can be contracted. By contracting F_k and scheduling E with the $(g+1)$-st follow-set that can be contracted, we obtain a schedule with the same total flow time (by Lemma 4), but with a lexicographically smaller $(t_k, t_{k-1}, \ldots, t_1, t_0)$, again contradicting our choice of S^*.

This concludes the proof of the lemma, and hence the theorem.

5 Concluding Remarks

It is readily verified that the time complexity of the SC algorithm is $O(n^2)$. We also note that the SC algorithm solves the non-preemptive two-machine open-shop problem with precedence constraints and UET operations, since it is isomorphic to the problem solved here [2, 16].

The *fractionality* of a preemptive schedule is the greatest reciprocal $\frac{1}{k}$ such that the interval between every two event times (i.e., start times, completion times, or preemption times) in the schedule is a multiple of $\frac{1}{k}$. Theorem 1 shows that there exists an ideal preemptive schedule of UET jobs under arbitrary precedence constraints on two processors with fractionality $\frac{1}{2}$. Hence, the recognition version of the problem is obviously in NP [6]. However, it is unknown whether the recognition version of the corresponding problem on three processors belongs to NP; indeed, it is unknown even if only the makespan or only the total flow time is being minimized. The *fractionality conjecture* [11] states that the problem of minimizing makespan on m processors has a solution of fractionality $\frac{1}{m}$, but it remains unverified even in the case of UET operations. We refer to the extension of this conjecture to the preemptive cases in the scheduling classification [8] as the *extended fractionality conjecture*.

We also mention the NP-*preemption hypothesis* which asserts that the recognition version of any preemptive problem in the scheduling classification belongs to NP. It is obviously true if the extended fractionality conjecture is true. The following gives an idea of how much stronger the extended fractionality conjecture is. Let the recognition version of a preemptive problem belong to NP, and let q be a polynomial upper bound on the size of the problem. Then it has a solution of fractionality $\frac{1}{q!}$ [15]. However, it is unknown whether the extended fractionality conjecture is true for all preemptive problems whose recognition versions belong to NP. If the NP-preemption hypothesis is true, then any preemptive problem on identical parallel processors in the scheduling classification polynomially reduces to a nonpreemptive UET problem [15, 3, 16].

References

[1] Ph. Baptiste, P. Brucker, S. Knust and V. G. Timkovsky, *Fourteen Notes on Equal-Processing-Time Scheduling*, Technical Report, University of Osnabruek, Osnabrueck, 2002.

[2] P. Brucker, B. Jurisch and M. Jurisch, Open shop problems with unit time operations, *ZOR – Methods and Models of Operations Research*, **37**, 59–73, 1993.

[3] P. Brucker and S. Knust, Complexity results for single-machine problems with positive finish-start time-lags, *Computing*, **63**, 299–316, 1999.

[4] E. G. Coffman, Jr. and R. L. Graham, Optimal scheduling for two-processor system, *Acta Informatica*, **1**, 200–213, 1972.

[5] M. R. Garey and D. S. Johnson, Two-processor scheduling with start-times and deadlines, *SIAM Journal on Computing*, **6**, 416–426, 1977.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman, San Francisco, 1979.

[7] E. L. Lawler, Preemptive scheduling of precedence-constrained jobs on parallel machines, in *Deterministic and Stochastic Scheduling*, M.A.H. Dempster, J.K. Lenstra and A.H.G. Rinnooy Kan, eds. Reidel, Dordrecht, 101–123, 1982.

[8] E. L. Lawler *et al.*, Sequencing and scheduling: algorithms and complexity, in *Handbook on Operations Research and Management Science, vol.* 4: *Logistics of Production and Inventory*, S.C. Graves, A.H.G. Rinnooy Kan, and P. Zipkin, eds. Elsevier, 1993, 445–552.

[9] J. K. Lenstra and A. H. G. Rinnooy Kan, Complexity results for scheduling chains on a single machine, *European Journal of Operational Research*, **4**, 270–275, 1980.

[10] J. Y.-T. Leung, and G. H. Young, Minimizing total tardiness on a single machine with precedence constraints, *ORSA Journal on Computing*, **2**, 346–352, 1990.

[11] R. R. Muntz and E. G. Coffman, Jr., Optimal preemptive scheduling on two-processor systems, *IEEE Transactions on Computers*, **C-18**, 1014–1020, 1969.

[12] R. McNaughton, Scheduling with deadlines and loss functions, *Management Science*, **6**, 1–12, 1959.

[13] R. Sethi, Scheduling graphs on two processors, *SIAM Journal on Computing*, **5**, 73–82, 1976.

[14] V. G. Timkovsky, A polynomial-time algorithm for the two-machine unit-time release-date job-shop schedule-length problem, *Discrete Applied Mathematics*, **77**, 185–200, 1997.

[15] V. G. Timkovsky, Is a unit-time job shop easier than identical parallel machines?, *Discrete Applied Mathematics*, **85**, 149–162, 1998.

[16] V. G. Timkovsky, Identical parallel machines vs. unit-time shops and preemption vs. chains in scheduling complexity, *European Journal of Operational Research*, to appear.