Eliminating Migration in Multi-Processor Scheduling*

Bala Kalyanasundaram† and Kirk R. Pruhs‡

We investigate the power of migration in real-time multi-processor scheduling with preemption. We show that every collection of jobs that can be completed by some schedule S on m processors, can also be completed by a non-migratory schedule S' on $6m-5$ processors. We can conclude from this result that, for many scheduling problems, such as $P \mid r_i, pmtn \mid \sum w_i(1-U_i)$ and special cases thereof, the ability of the scheduler to migrate jobs is of only limited advantage. Our proof is constructive, and can be implemented to run in pseudo-polynomial time (or in polynomial time at the cost of raising the bound from 6m-5 to 12m-5). As an example of the usefulness of this result, we give a polynomial time 6-approximation reduction from the problem $P \mid r_i, pmtn \mid \sum w_i(1-U_i)$ to the problem $1 \mid r_i, pmtn \mid \sum w_i(1-U_i)$. The proof of correctness of this reduction critically uses the fact there is a nonmigratory schedule that closely approximates the optimal migratory schedule.

1. INTRODUCTION

1.1. Motivation

Following up on the work in [11], we investigate the power of migration (the ability to resume a job on another processor after preemption) in problems involving scheduling preemptable jobs on a multiple identical processors. For concreteness consider the problem $P \mid r_i, pmtn \mid \sum w_i(1-U_i)$, that is, the jobs have non-negative benefits, release dates, and deadlines, and the objective function is to maximize the aggregate benefit of those jobs completed by their deadline on m processors. As in [11] we define ω_m as the maximum, over all instances $\mathcal{I}$, of the ratio of the benefit of the optimal *migratory* schedule for $\mathcal{I}$ to the benefit of the optimal *nonmigratory* schedule for $\mathcal{I}$. In a nonmigratory schedule a job can only be run on the processor on which it is first run. Our main motivation for considering

* Supported in part by NSF Grant CCR-9734927.

† Dept. of Computer Science, Georgetown University. E-Mail: kalyan@cs.georgetown.edu

‡ Dept. of Computer Science, University of Pittsburgh. E-Mail: kirk@cs.pitt.edu.

such a ratio comes from difficulties that arise in bounding the performance ratio (or the competitive ratio in an online setting) of an approximation algorithm A for this problem, and variants thereof. For a maximization problem the performance ratio of A is

$$\max_{\text{Instances } \mathcal{I}} \frac{|\text{OPT}(\mathcal{I})|}{|A(\mathcal{I})|}$$

where $A(\mathcal{I})$ is the schedule produced by A on input $\mathcal{I}$, $\text{OPT}(\mathcal{I})$ is the optimal schedule, and $|S|$ is the benefit of the schedule S. The simpler the combinatorial structure for $\text{OPT}(\mathcal{I})$, the easier the task of comparing it against $A(\mathcal{I})$. Note that nonmigratory schedules are generally simpler combinatorially than migratory schedules, and that for scheduling problems such as $P \mid r_j, pmtn \mid \sum(1 - U_j)$ (maximizing the number of jobs completed before their deadlines) there appear to be no nice lower bounds for $|\text{OPT}(\mathcal{I})|$. So if one can design an approximation algorithm A that has a performance ratio c against the optimal nonmigratory schedule, then one knows that the performance ratio of A is at most $c \cdot \omega_m$. We give an example application of this use in section 3.

From a practical point of view, nonmigratory schedules may be preferable to migratory schedules since migration may involve significant overhead. On the other hand, allowing migration may make life easier for the algorithm designer. For example, the standard algorithms for many real-time scheduling problems, such as $P \mid r_i, pmtn \mid L_{\max}$ are relatively easy reductions to linear programming; however, such reductions would seem to be of little use in finding nonmigratory schedules since generally nonmigratoriness can not be expressed by linear constraints. If one has an efficient construction $\mathcal{C}$ that transforms a migratory schedule S into a nonmigratory schedule S' that closely approximates S, then one can in some sense get the best of both worlds. That is, an algorithm designer can design a scheduling algorithm A that uses migrations, knowing that he/she can produce an only slightly worse nonmigratory scheduling by feeding the output of A into $\mathcal{C}$.

Previously the only bounds known on ω_m were for *the uniform density case* $P \mid r_i, pmtn \mid \sum x_i(1 - U_i)$, that is, when the benefit of a job is proportional to its length. The fact that $\omega_m \leq 3$, in the uniform density case, follows immediately from the results in [12]. In [11], it is shown that $\frac{2m-1}{m} \leq \omega_m \leq \frac{1}{1-(1-\frac{1}{2m})^m}$ in the uniform density case. For large m, $\frac{1}{1-(1-\frac{1}{2m})^m}$ is approximately 2.54.

1.2. Our Results

Let $\Phi(m)$ be the collection of instances $\mathcal{I}$ for which there exists a feasible *migratory* schedule $S(\mathcal{I})$ on m processors that completes all the jobs in $\mathcal{I}$.

For $\mathcal{I} \in \Phi(m)$, let $c(\mathcal{I})$ be the minimum integer k, such that there exits a schedule S' on k processors such that

- S' is nonmigratory,
- the earliest execution time of each job J_i in S' is not earlier than the earliest execution time of J_i in S, and
- the completion time of each job J_i is S' is not later than the completion time of J_i in S.

We then define

$$\alpha_m = \max_{\mathcal{I} \in \Phi(m)} c(\mathcal{I})$$

that is, the minimum number of processors required to complete, without migration, all the jobs completed in a migratory schedule on m processors. Our main result is then

THEOREM 1.1. *$\alpha_m \leq 6m - 5$, and hence, $\omega_m \leq \frac{6m-5}{m}$.*

We prove this theorem by giving a pseudo-polynomial time algorithm that transforms a migratory schedule on m processors into a nonmigratory schedule on $6m - 5$ processors. We also give a polynomial time algorithm that transforms a migratory schedule on m processors into a nonmigratory schedule on $12m - 5$ processors. The lower bound example in [11] shows that $\alpha_m \geq 2m - 1$. It is immediately clear that $\omega_m \leq \frac{\alpha_m}{m}$ since the jobs completed on the m processors in the nonmigratory schedule that obtain maximum aggregate benefit, have aggregate benefit at least $\frac{m}{6m-5}$ of the aggregate benefit of the nonmigratory schedule. Hence, we can conclude that $\omega_m \leq \frac{6m-5}{m}$.

Our results fundamentally improve on those in [11]. The most obvious improvement is that our results hold for the general case where the jobs have arbitrary benefits (not just the case where the benefit of the job is proportional to its length). A more subtle improvement is that we obtain a nonmigratory schedule that completes *every* job by its deadline, which may be useful in minimization problems where knowing that ω_n is small is of little direct use.

In section 2, our general strategy for bounding α_m is as follows. In subsection 2.1, we show how to in pseudo-polynomial time transform an arbitrary migratory schedule on m processors into a canonical migratory schedule S on m processors. The transformation will be an arbitrary sequence of local adaptations that try to run jobs with earlier completion times earlier in the schedule. A canonical schedule is one for which there is no applicable local adaptation. In subsection 2.2, we show that canonical schedules can be described by a linear order $\prec_t$ on the jobs that can be run at time t, for each possible time t. In section 2.3, we show how each

$\prec_t$ can be obtained from $\prec_{t-1}$ by a sequence of insertions, deletions, or swapping of two consecutive jobs. With the preliminaries out of the way, we will, in subsection 2.4, prove theorem 1.1 in the following manner. We define the conflict graph G for a schedule S to contain a vertex for each job, and to contain an edge between two vertices J_i and J_j if there exists a time t where both J_i and J_j are run at time t in S. We then show that the density, which is the number of edges divided by the number of vertices, of every induced subgraph of G of a canonical schedule S is at most $3(m-1)$. Since the minimum degree of a vertex in a graph is at most twice the density of that graph, the minimum degree of a vertex in every induced subgraph of the conflict graph G is at most $6(m-1)$. A standard result from extremal graph theory implies that G must then be $6m-5$ colorable. Theorem 1.1 then follows by noting that all jobs with the same color can be scheduled entirely on one processor. We believe that this technique of using the conflict graph will be useful in other scheduling problems. Finally in subsection 2.5, we show how to in polynomial time transform an arbitrary migratory schedule on m processors into a canonical migratory schedule S on $2m$ processors.

In section 3, we prove the following result.

THEOREM 1.2. *If there is a polynomial time algorithm with performance ratio c for the problem $1 \mid r_i, pmtn \mid \sum w_i(1-U_i)$, then there is a polynomial time algorithm with performance ratio $6c$ for the problem $P \mid r_i, pmtn \mid \sum w_i(1-U_i)$.*

This reduction repeatedly applies the single processor algorithm to the remaining unscheduled jobs. This result is an example of the usefulness of knowing a bound on ω_m; that is, the proof of correctness of the reduction critically uses the fact there is a nonmigratory schedule that closely approximates the optimal migratory schedule. We can then conclude that there exists a pseudo-polynomial time 6-approximation algorithm for $P \mid r_i, pmtn \mid \sum w_i(1-U_i)$ since there exists a pseudo-polynomial time algorithm for $1 \mid r_i, pmtn \mid \sum w_i(1-U_i)$ [14]. Obviously this algorithm is then a polynomial time 6-approximation algorithm for the problem $P \mid r_i, pmtn \mid \sum(1-U_i)$. As far as we know this is the first polynomial time approximation algorithm for this problem with a constant performance ratio.

1.3. Related Work

We briefly summarize our knowledge of the current state of affairs of the complexity and approximatablity of real-time scheduling problems involving preemptable jobs. For information on offline scheduling, including the three field scheduling notation, see [4] or [15]. For a general introduction to approximation algorithms for scheduling problems see [8]. For a recent general survey of online scheduling see [17].

We start with single processor results. The offline versions of the following two problems $1 \mid r_i, pmtn \mid \sum w_i(1-U_i)$ and $1 \mid r_i, pmtn \mid \sum w_i U_i$ are NP-hard [5], but can be solved in pseudo-polynomial time using Lawler's dynamic programming algorithm [14]. Hence, Lawler's algorithm runs in polynomial time on the unit benefit versions of these problems. For the online version of $1 \mid r_i, pmtn \mid \sum x_i(1-U_i)$, there is a 4-competitive deterministic algorithm, and this is optimal for deterministic online algorithms [2, 19]. There is no constant competitive deterministic or randomized online algorithm for $1 \mid r_i, pmtn \mid \sum (1-U_i)w_i$ [12]. However, in [9] it is shown that if the online scheduler is given a faster processor than the adversary, then there is a relatively simple deterministic online algorithm that is constant competitive. There is no constant competitive deterministic online algorithm for $1 \mid r_i, pmtn \mid \sum(1-U_i)$. However, there is an $O(1)$ competitive randomized algorithm [10] for this problem. It is well known that if all the jobs can be completed by their deadline then both the Earliest Deadline First (EDF) algorithm and the Least Laxity First (LLF) algorithm will produce an optimal schedule.

We now turn to results for multi-processor scheduling. The problems $P2 \mid pmtn, r_i \mid \sum U_i$ and $P \mid pmtn \mid \sum U_i$ are minimally NP-hard [5]. In the context of the problem $Pm \mid r_i, pmtn \mid \sum U_i$, $m \geq 2$, no online algorithm can finish all the jobs by their deadline even on instances for which there is a schedule that finishes all the jobs by their deadline [6]. In [16] it is shown that LLF and EDF will complete all the jobs by their deadline, on instances where the adversary completes all the jobs, if LLF and EDF are equipped with a processor twice as fast as the adversary's processor. Some improvements on these results can be found in [13].

2. ELIMINATING MIGRATIONS

We begin with some preliminary definitions and concepts. It will be convenient for us to adopt a slightly nonstandard notion of time. Normally when one refers to a time t one thinks of t as a zero dimensional point. In this paper, when we refer to a time t we will be referring to a one dimensional interval of length one. So when we refer to time t, we mean what more traditionally would be referred to as the time interval $(t, t+1)$. Further we assume that time is discrete, so the possible times are the natural numbers.

Throughout this section, we assume that O represents an initial arbitrary migratory schedule for some collection $\mathcal{J}$ of jobs on m identical processors $P_1, \ldots, P_m$. Since we number the jobs, and thus associate a linear order on the jobs, our definition of a schedule is slightly nonstandard. Formally, a

schedule S assigns a job J_j to each time/processor pair (t, P_i) (we say that J_j is run on P_i at time t), subject to the following three conditions:
1. The jobs are numbered $J_1, J_2, \ldots, J_n$.
2. No two jobs are assigned to the same pair (t, P_i).
3. No job is assigned to pairs (t, P_i) and (t, P_j) where $P_i \neq P_j$.

If S does not assign a job to particular time/processor pair (t, P_i), then we say that processor P_i is *idle* at time t. Without loss of generality, we may assume that there are no times where all processors are idle in O.

DEFINITION 2.1.

1. The *length* or *execution time* x_i of a job J_i is the total number of times that J_i was assigned to a processor in O.
2. Let the *completion time* $C_i(S)$ be the last time that J_i was run in the schedule S. Let $F_i(S)$ denote the first time that J_i was run in a schedule S.
3. We say that J_i is legal at time t in a schedule S if $F_i(S) \leq t \leq C_i(S)$.
4. Let $\mathcal{L}_t(S)$ be the collection of jobs that are legal at time t in schedule S.
5. Let $\mathcal{L}_{i,j}(S)$ be the collection of times where both of the jobs J_i and J_j are legal in schedule S.
6. Let $\mathcal{R}_i(S)$ be the collection of times when J_i is run in schedule S.
7. We say that a job J_i *ejects* a job J_j at time t in a schedule S if $t \in \mathcal{L}_{i,j}(S)$, $t \in \mathcal{R}_i(S)$, and $t \notin \mathcal{R}_j(S)$.
8. Let $L(S)$ be the last time that a job is run in S.
9. A linear order $\sqsubset$ on the jobs in S is *completion preserving* if $C_i(S) < C_j(S)$ implies that $J_i \sqsubset J_j$.
10. A linear order $\sqsubset$ on the jobs in S is *tie-breaking* if $C_i(S) = C_j(S)$, and $\mathcal{R}_i(S) \subset \mathcal{R}_j(S)$, implies that $J_j \sqsubset J_i$. (Here $\subset$ denotes proper subset, so it is not the case that $A \subset A$.)

For brevity, whenever the schedule S under consideration is unambiguous, we drop S from our notation.

2.1. The Canonical Schedule Construction

First we define the local adaptation that we use to transform an arbitrary schedule into a canonical schedule.

DEFINITION 2.2.

1. A *switch* is a quadruple (J_j, a, J_i, b) where a and b are times with $a < b$.
2. A switch (J_j, a, J_i, b) is *applicable* to a schedule S if J_j ejects J_i at time a in S, and J_i ejects J_j at time b in S.

TABLE 1.

Summary of Relations

Relation	Where Defined	Comment
$\sqsubset$		Generic relation
$\lhd$	Definition 2.5	Pre-canonical relation
$\ll$	Definition 2.6	Canonical relation
$\prec_t$	Definition 2.7	Canonical schedule's job preference at time t
$\propto_s$	Subsection 2.3	A relation in the decomposition of $\prec$

The result of applying an applicable switch (J_j, a, J_i, b) to S is a schedule where J_i is run in place of J_j at time a and J_j is run in place of J_i at time b. We say that that J_i was *switched earlier* $(b-a)$ positions and that J_j was *switched later* $(b-a)$ positions by this switch.

DEFINITION 2.3. Let $\sqsubset$ be a linear order on the jobs run in a schedule S. A switch (J_j, a, J_i, b) is $\sqsubset$-*applicable* to a schedule S if it is applicable, and $J_i \sqsubset J_j$.

DEFINITION 2.4.

1. A schedule S is *pseudo-canonical* under a linear order $\sqsubset$ on the jobs if there is no $\sqsubset$-applicable switch in S.

2. A schedule S is *pre-canonical* under a linear order $\sqsubset$ on the jobs in S if S is pseudo-canonical under $\sqsubset$, and $\sqsubset$ is completion preserving.

3. A schedule S is *canonical* under a linear order $\sqsubset$ on the jobs in S if S is pre-canonical under $\sqsubset$, and $\sqsubset$ is tie-breaking.

Intuitively, the goal of constructing a canonical schedule is to significantly reduce the number of distinct pairs of jobs that are ever run simultaneously. In order to establish this fact, we consider the preference $\prec_t$ of the canonical schedule on the jobs at time t. We further refine the $\prec_t$ orders to the $\propto$ orders so that the change from each $\propto_{s-1}$ to $\propto_s$ is a single swap. Table 1 may be of use to the reader to help keep these relations straight.

We now introduce a particular completion time preserving linear order that will be useful to us. Recall that a numbering of the job is implicit in every schedule.

Definition 2.5. We define a linear order $\lhd^S$ on the jobs run in a schedule S as follows. For any two jobs J_i and J_k it is the case that $J_i \lhd^S J_k$ if $C_i(S) < C_k(S)$, or $C_i(S) = C_k(S)$ and $i < k$.

An example of a $\lhd$-applicable switch is shown in figure 1 where we assumed $F_i(S) < a$. The cross hatched areas in figure 1 represent times when neither of jobs J_i or J_k are being run on the corresponding processor.

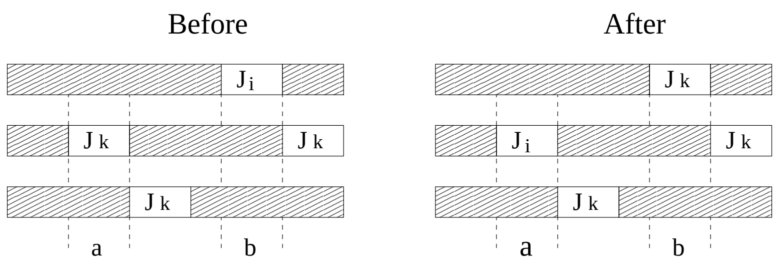

FIG. 1. A $\lhd$-applicable Switch

Swap-Till-You-Drop Procedure: Initially let the schedule S be the initial schedule O. While there exists $\lhd^S$-applicable swap in S, modify S by applying this swap. We retain the original numbering of jobs in the modified S.

Lemma 2.1. *The Swap-Till-You-Drop procedure produces a schedule S, which is pre-canonical under $\lhd^S$, after at most $(nL)^3$ switches, where $L = L(O)$.*

Proof. Let S be the schedule at the start of an iteration of *Swap-Till-You-Drop*. We say a switch (J_j, a, J_i, b) is *good* if it is $\lhd^S$-applicable and it causes the completion time of J_i to decrease. Note that a $\lhd^S$-applicable switch does not increase the completion time of J_j since C_j is strictly after time b, or we would get a contradiction to $\lhd^S$-applicability. Hence, by the definition of L, there can be at most nL good switches.

We now claim that there can be at most $(nL)^2$ switches between any pair of good switches. We now restrict our attention to the period P between two consecutive good switches. For the ease of presentation, without loss of generality, assume that the jobs are numbered such that $J_i \lhd J_{i+1}$, $1 \leq i \leq n-1$. Note by assumption that the completion times of the jobs are invariant during P. Let $s_{i,j}$ be the sum over all switches involving J_i and J_j during P of the number of positions changed during this switch.

We now will show that

$$\sum_{j=i+1}^{n} s_{i,j} \leq iL^2 - \sum_{j=1}^{i-1} \sum_{k=i+1}^{n} s_{j,k} \tag{1}$$

Note that the left hand side of equation 1 is the total number of switches that advance portions of J_i to times earlier in the schedule. Equation 1 is then sufficient to complete the proof of the lemma because the total number of switches during P is

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} s_{i,j} \leq \sum_{i=1}^{n} iL^2 \leq (nL)^2$$

We prove equation 1 by induction on i. Consider the case $i = 1$. The job J_1 can not be switched later by the definition of $\lhd$-applicability. Furthermore, by the definition of L it is the case that J_1 can be switched earlier at most L^2 positions. Hence, the claim holds for $i = 1$ since the double sum in equation (1) is empty. Now consider a general i. By induction

$$\begin{aligned}
\sum_{j=i}^{n} s_{i-1,j} &= s_{i-1,i} + \sum_{j=i+1}^{n} s_{i-1,j} \\
&\leq (i-1)L^2 - \sum_{j=1}^{i-2} \sum_{k=i}^{n} s_{j,k} \\
&= (i-1)L^2 - \sum_{j=1}^{i-2} s_{j,i} - \sum_{j=1}^{i-2} \sum_{k=i+1}^{n} s_{j,k}
\end{aligned}$$

Hence,

$$s_{i-1,i} \leq (i-1)L^2 - \sum_{j=1}^{i-2} s_{j,i} - \sum_{j=1}^{i-1} \sum_{k=i+1}^{n} s_{j,k} \tag{2}$$

Now note that

$$\sum_{j=i+1}^{n} s_{i,j} \leq L^2 + s_{i-1,i} + \sum_{j=1}^{i-2} s_{j,i}$$

since the number of positions that J_i can be switched earlier is at most L^2 plus the number of positions it was switched later. Hence, substituting from equation (2) we get that

$$\sum_{j=i+1}^{n} s_{i,j} \leq iL^2 - \sum_{j=1}^{i-1} \sum_{k=i+1}^{n} s_{j,k}$$

which is what we wanted to establish. ∎

DEFINITION 2.6. Let S be an arbitrary pre-canonical schedule under $\lhd^S$. We define a relation $\ll^S$ on jobs in S as follows (Let J_i and J_j be arbitrary jobs run in S):

1. If $C_i(S) < C_j(S)$ then $J_i \ll^S J_j$.
2. Otherwise if $\mathcal{R}_j(S) \subset \mathcal{R}_i(S)$ then $J_i \ll^S J_j$.
3. Otherwise, if $J_i \lhd^S J_j$ then $J_i \ll^S J_j$.

Observe that $\ll^S$ is almost identical to $\lhd^S$, except when the completion times of jobs are identical.

LEMMA 2.2. *If a schedule S is pre-canonical under $\lhd^S$ then it is canonical under $\ll^S$.*

Proof. Obviously $\ll^S$ is a completion preserving and tie-breaking linear order. To see that there is no $\ll^S$ applicable switch (J_j, a, J_i, b), note that then it must be the case that $J_i \ll^S J_j$ by the definition of $\ll^S$-applicability. But then it must be the case that $J_j \lhd^S J_i$ or S would not be pre-canonical under $\lhd^S$. Hence, since $\lhd^S$ is completion preserving, it must be the case that $\mathcal{R}_j(S) \subset \mathcal{R}_i(S)$ by the definition of $\ll^S$. But then this is a contradiction since J_j could never eject J_i in S. ∎

2.2. Properties of a Canonical Schedule

In this section, let S be an arbitrary schedule that is canonical on m processors under $\ll^S$; all references to completion times, first run times, etc. are in reference to S, unless specified otherwise. Since we have fixed S, we may drop the reference to S in our notations.

In this subsection we show that at any particular time t that we can use S to associate a linear order $\prec_t^S$ on the jobs in $\mathcal{L}_t(S)$. Intuitively, $\prec_t$ denotes the schedule's preferences among the jobs at time t. If at time t it is the case that S runs a job J_i, but not J_j, then intuitively S prefers J_i to J_j. Intuitively, one can interpret $\ll^S$ as the initial order $\prec_0$. It is not hard to see that for some pairs of jobs, say J_i and J_j, it may be the case that the canonical schedule's preference may change over time, that is, $J_i \prec_{t-1} J_j$ and $J_j \prec_t J_i$. However, we will show that for each pair of jobs there is at most one point of time where S's preference among these two jobs changes.

DEFINITION 2.7. The domain of $\prec_t$ are those jobs legal at time t. Let J_i and J_j be jobs such that $J_i \ll J_j$. Let $g_{i,j} = g_{j,i}$ be the earliest time such that, $g_{i,j} \geq F_i$, $g_{i,j} \geq F_j$, and S is running J_j but not J_i at this time. If no such time $g_{i,j}$ exists then we say that $g_{i,j} = +\infty$, or that $g_{i,j}$ is undefined. If $g_{i,j}$ is undefined, then by definition, for all $t \in \mathcal{L}_{i,j}$, it is the case that $J_i \prec_t J_j$. If $g_{i,j}$ is defined, let $f_{i,j} = f_{j,i}$ be the earliest time such that, $f_{i,j} \geq F_i$, $f_{i,j} \geq F_j$, and for times t satisfying $f_{i,j} \leq t \leq g_{i,j}$ it is the case that if $t \in \mathcal{R}_i$ then $t \in \mathcal{R}_j$. Note that if $g_{i,j}$ is defined then it must be the case that $f_{i,j}$ is defined and $f_{i,j} \leq g_{i,j}$. If $t < f_{i,j}$ and $t \in \mathcal{L}_{i,j}$, then by definition it is the case that $J_i \prec_t J_j$. If $t \geq f_{i,j}$ and $t \in \mathcal{L}_{i,j}$, then by definition it is the case that $J_j \prec_t J_i$.

Intuitively, $f_{i,j}$ is the swap over point where S starts prefering J_j to J_i. See figure 2 for illustration of the definitions of $f_{i,j}$ and $g_{i,j}$. We now need to establish some basic properties of $\prec_t$.

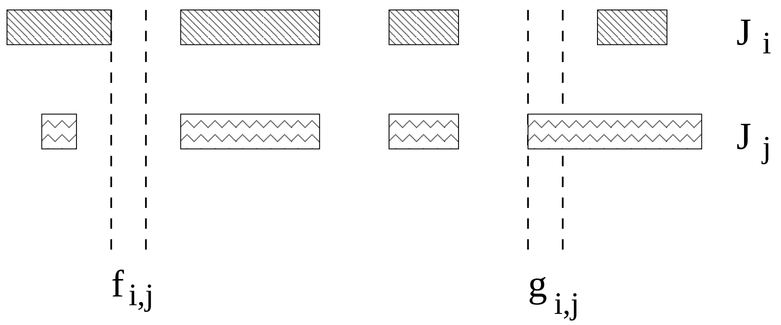

FIG. 2. An example of $f_{i,j}$ and $g_{i,j}$

We now observe a useful property satisfied by $f_{i,j}$.

OBSERVATION 2.1. *Let J_i and J_j be two arbitrary jobs such that $J_i \ll J_j$.*

- *If $f_{i,j}$ is defined then for all $t \in \mathcal{L}_{i,j}$ it is the case that:*
  - *If $t < f_{i,j}$ then $J_i \prec_t J_j$.*
  - *If $f_{i,j} \leq t$ then $J_j \prec_t J_i$.*
- *If $f_{i,j}$ is not defined then for all $t \in \mathcal{L}_{i,j}$ it is the case that $J_i \prec_t J_j$.*

LEMMA 2.3. *At all times t, $t \in [f_{i,j}, g_{i,j})$, $t \in \mathcal{R}_i$ if and only if $t \in \mathcal{R}_j$.*

Proof. Assume to reach a contradiction there exists a $t \in [f_{i,j}, g_{i,j})$ such that S is running exactly one of J_i or J_j at time t. If S is running J_i at time t then we get a contradiction to the definition of $f_{i,j}$, and if S is running J_j at time t then we get a contradiction to the definition of $g_{i,j}$. ∎

LEMMA 2.4. *At all times t, if $t \in \mathcal{R}_i$, and $J_j \prec_t J_i$, then $t \in \mathcal{R}_j$.*

Proof. Assume to reach a contradiction that there is some time t for which $t \in \mathcal{R}_i$, and $J_j \prec_t J_i$, but $t \notin \mathcal{R}_j$. Since $\prec_t$ is defined for the pair of jobs J_j and J_i at time t, it must be the case that these two jobs are legal at time t. First assume that $t < f_{i,j}$ (this includes the case that $f_{i,j}$ is undefined). Then since $t < f_{i,j} \leq g_{i,j}$ and $J_j \prec_t J_i$, it must be the case that $J_j \ll J_i$ by the definition of $\prec_t$. But then we get a contradiction to the definition of $g_{i,j}$ as t satisfies all the conditions required in the definition of $g_{i,j}$.

If $t \in [f_{i,j}, g_{i,j})$, we then get a contradiction to lemma 2.3.

Now assume that $t = g_{i,j}$. Since $J_j \prec_t J_i$, by the definition of $g_{i,j}$, it must be the case that $t \in \mathcal{R}_j$, which is a contradiction.

Now assume that $t > g_{i,j}$. Then since $t > g_{i,j}$ and $J_j \prec_t J_i$, it must be the case that $J_i \ll J_j$ by the definition of $\prec_t$. We now get a contradiction to the canonicalness of S as $(J_j, g_{i,j}, J_i, t)$ is a $\ll$-applicable switch. ∎

LEMMA 2.5. *For all times t, the relation $\prec_t$ is a linear order.*

Proof. By the definition of $\prec_t$, it is obvious that every pair of jobs in $\mathcal{L}_t$ is comparable, and that $\prec_t$ is anti-symmetric. Hence, we need only prove that $\prec_t$ is transitive. To reach a contradiction let t be the smallest time such that $\prec_t$ is not transitive. Consider three distinct jobs J_i, J_j, and J_k, such that $J_i \prec_t J_j \prec_t J_k$, but $J_k \prec_t J_i$ instead of $J_i \prec_t J_k$. Let $u(i,j) \geq t$ be the earliest time such that S is running one of J_i and J_j, but not the other. We allow the possibility that $u(i,j)$ may be strictly greater than the completion time of one of J_i or J_j. If no such time exists, let $u(i,j) = +\infty$. Similarly define $u(j,k)$ and $u(k,i)$. We now break the proof into cases.

Case 1: In this case assume that $u(i,j) = u(j,k) = u(k,i) = +\infty$. Hence, it must then be the case that $C_i = C_j = C_k$ by the definition of the $u(\cdot,\cdot)$'s. We now break this case into subcases.

Subcase 1a: Suppose t is not the first run time of any of the jobs J_i, J_j and J_k. Hence, observe that J_i, J_j and J_k must be legal at time $t-1$. If $J_i \prec_{t-1} J_j \prec_{t-1} J_k \prec_{t-1} J_i$ then we get a contradiction to the selection of t. Hence, two of the jobs must have changed their relationship between time $t-1$ and time t. Without loss of generality, assume that $J_j \prec_{t-1} J_i$. Hence, $f_{i,j} = t$ and so it must be the case that there is a time $t' \geq t$ such that $t' \in \mathcal{R}_i$ and $t' \notin \mathcal{R}_j$. Hence, we get a contradiction to $u(i,j) = +\infty$.

Subcase 1b: Suppose t is the first run time of at least one of J_i, J_j and J_k.

We first argue that the first run time of at least one of these three jobs is strictly before t. Assume otherwise that $t = F_i = F_j = F_k$. Then since $u(i,j) = u(j,k) = u(k,i) = +\infty$ none of J_i, J_j or J_k can eject one of the other two jobs. By the definition of $\prec$ and the fact that $J_i \prec_t J_j \prec_t J_k \prec_t J_i$ it must be the case that $J_i \ll J_j \ll J_k \ll J_i$, which is a contradiction.

Now, without loss of generality, assume that $F_j = t$ and that $F_k < t$. Since $u(i,j)$, $u(j,k)$, and $u(k,i)$ are $+\infty$, and $C_i = C_j = C_k$, we conclude

that J_j can never eject J_k. Therefore, by the definition of $J_j \prec_t J_k$ it must be that case that $J_j \ll J_k$ and $R_j \subset R_k$. But by condition 2. in the definition of $\ll$, it must be the case that $J_k \ll J_j$, which is a contradiction.
Case 2: In this case assume that at least one of $u(i,j)$, $u(j,k)$ or $u(k,i)$ is finite. Without loss of generality assume that $u(i,j) \leq u(j,k)$ and $u(i,j) \leq u(k,i)$. So at least $u(i,j)$ is finite. We now break this case into subcases.
Subcase 2a: In this subcase assume that $u(i,j) \in \mathcal{R}_j$ and $u(i,j) \notin \mathcal{R}_i$. By the definition of $u(i,j)$, during the time interval $[t, u(i,j))$, it is the case that whenever S is running one of J_i and J_j, S is running both of J_i and J_j. Hence, by the definition of $\prec$ it must be the case that $J_j \prec_t J_i$, which is a contradiction.
Subcase 2b: In this subcase assume that $u(i,j) \notin \mathcal{R}_j$ and $u(i,j) \in \mathcal{R}_i$.

Suppose that J_k is not run at time $u(i,j)$. By the definition of $u(k,i)$, during the time interval $[t, u(k,i))$, it is the case that whenever S is running one of J_k and J_i, S is running both of J_k and J_i. Applying the fact that $u(k,i) \geq u(i,j)$, we conclude that during the time interval $[t, u(i,j))$, whenever S is running one of J_k and J_i, S is running both of J_i and J_k. Now apply the fact that J_i is run at $u(i,j)$ and J_k is not run at $u(i,j)$. As a result, by the definition of $\prec_t$, we can conclude that $J_i \prec_t J_k$, which is a contradiction.

On the other hand, suppose that J_k is run at time $u(i,j)$. As a result $u(j,k) = u(i,j)$. By the definition of $u(j,k)$, during the time interval $[t, u(j,k))$, it is the case that whenever S is running one of J_j and J_k, S is running both of J_j and J_k. Applying the fact that J_k is run at $u(j,k)$ and the definition of $\prec_t$, we can conclude that $J_k \prec_t J_j$, which is a contradiction. ▮

2.3. Decomposing Canonical Schedules

Recall that S is the canonical schedule under consideration. Our goal is to show that the number of distinct pairs of jobs that are ever run simultaneously is not large. Since canonical schedules are always trying to run the earliest completing jobs first, one might intuitively expect that the $\prec$ relation is time invariant, that is, $J_i \prec_s J_j$ if and only if $J_i \prec_t J_j$. Unfortunately, $\prec$ may not quite be time invariant. We establish that $\prec$ can vary over time only in a very restricted way. We do so by stating the change from $\prec_{t-1}$ to $\prec_t$ in terms of consecutive swaps of jobs that are consecutive in the $\propto$ order.

We now introduce some definitions. Let $\sqsubset$ be some linear order on the jobs. Let $J_i \sqsubset J_j$ be two consecutive jobs with respect to $\sqsubset$, that is $J_i \sqsubset J_j$ and there does not exist a k such that $J_i \sqsubset J_k \sqsubset J_j$. *Swapping* J_i and J_j results in a linear order that is identical to $\sqsubset$ except that J_i and J_j have changed order, that is, now $J_j \sqsubset J_i$ and the $\sqsubset$ relation of the other jobs did not change. This swap is *legal* if $J_i \ll J_j$. We say that the $\sqsubset$*-rank* of a job J_i is r if there are exactly $r-1$ jobs such that $J_j \sqsubset J_i$. If $J_i \sqsubset J_j$ then we say that J_i has *lower* $\sqsubset$*-rank* than J_j. A job J_i is $\sqsubset$*-prime* if its $\sqsubset$-rank is at most m. A $\sqsubset$*-prime swap* is one that occurs between the mth and $(m+1)$st $\sqsubset$-ranked jobs in the linear order. An *inversion* between linear orders $\sqsubset$ and $\sqsubset'$ is a pair of jobs J_i and J_j such that $J_i \sqsubset J_j$ and $J_j \sqsubset' J_i$.

We now construct a sequence of linear orders $\propto_0, \ldots, \propto_b$ and a sequence of natural numbers $t(1) < t(2) < \ldots < t(L)$, such that $\propto_{t(s)}$ is identical to $\prec_s$, for $1 \leq s \leq L$. Furthermore, each $\propto_t$ derivable from $\propto_{t-1}$ in one of the following ways:

- A job J_i is inserted as a $\propto_t$-prime job into $\propto_t$. Intuitively, this corresponds to the first run of J_i.
- A $\propto_{t-1}$-prime job J_i is deleted from $\propto_{t-1}$. Intuitively, this corresponds to the completion of J_i.
- A legal swap is applied to $\propto_{t-1}$.

Further, no job may be inserted more than once. In each case, no other changes occur from $\propto_{t-1}$ to $\propto_t$.

Constructing the $\propto$ Orders: We construct the $\propto$ relations in stages. Stage s, $1 \leq s \leq L$, intuitively corresponds to the transformation from time $s-1$ to time s in S. To get started, let $t(0) = 0$, and let $\propto_0$ be the empty relation. We now consider stage $s \geq 1$. Let v be the sum of the number of inversions from $\prec_{s-1}$ to $\prec_s$, the number of jobs with first run time s, and the number of jobs with completion time $s-1$. Then $t(s) = t(s-1) + v$. Each stage is divided into three phases: the completion phase, the swap phase, and the first run phase.

Completion Phase: Let $r = t(s-1)$. We iteratively consider the jobs with completion time $s-1$ in S in decreasing $\ll$ order. Assume that we are considering job J_i. We increment r, and let $\propto_r$ be identical to $\propto_{r-1}$ except that J_i is no longer in the domain.

First Run Phase: Let $u = t(s)$, and let the linear order $\propto_{t(s)}$ be identical to the linear order $\prec_s$. We iteratively consider the jobs with first run time s in S in decreasing $\ll$ order. Assume that we are considering job J_i. We decrement u, and let $\propto_u$ be identical to $\propto_{u+1}$ except that J_i is no longer in the domain.

Intuitively, the swap phase can be thought of as a minor variant of selection sort that reorders the jobs from their order in some $\prec_{s-1}$ to their order in $\prec_s$. The swap phase consists of subphases, where the ith subphase identifies the ith lowest ranked job J_i in $\prec_s$. In this subphase, the job J_i is then swapped with its immediate predecessor in the list until J_i is in the i position. Hence, we maintain the invariant that after the ith subphase it is the case that the first i jobs agree with the first i jobs in the $\prec_s$ ranking, and that the relative order of the jobs in positions greater than i is identical to the starting order $\prec_{s-1}$. This property enables us to establish that only adjacent jobs are swapped, and that each pair of items are swapped at most once.

Swap Phase: Observe that $r = t(s-1) + c$, where c is the number of jobs with completion time $s-1$. Similarly, observe that $u = t(s) - f$, where f is the number of jobs with first run time s. Throughout the swap phase the jobs in the domain of the $\propto$ relations are the same as the jobs in the domain of $\propto_r$. We divide each swap phase into sub-phases. Let $w(1) = r = t(s-1) + c$. In the ith subphase, let $J_{w(i)}$ be the lowest $\propto_u$-ranked job such that $J_{w(i)}$'s $\propto_u$ ranking, call it $\alpha(i)$, is different than $J_{w(i)}$'s $\propto_{w(i)}$ ranking, call it $\beta(i)$. If there is no such $J_{w(i)}$ then the swap phase is over. Note that it must be the case that $\beta(i) > \alpha(i)$. We maintain an order-preserving property that for any two jobs J_a and J_b with $\propto_{w(i)}$ ranks between $\alpha(i)$ and u it is the case that $J_a \propto_{w(i)} J_b$ if and only if $J_a \prec_{s-1} J_b$. Notice that the order-preserving property holds for the first phase. Let $J_{\sigma(1)}, \ldots, J_{\sigma(k)}$ be the jobs that are $\prec_{s-1}$-ranked lower than $J_{w(i)}$, and that are involved in some inversion with J_i between $\prec_{s-1}$ and $\prec_s$. Observe that $\beta(i) = \alpha(i) + k$. Assume that $J_{\sigma(1)}, \ldots, J_{\sigma(k)}$ are ordered by decreasing $\prec_{s-1}$ rank. Hence, by our choice of $J_{w(i)}$, the order-preserving property, and the renumbering of the jobs, the jobs in $\propto_{w(i)}$ with $\propto_{w(i)}$-ranking $\alpha(i)$ to $\alpha(i) + k = \beta(i)$ are the jobs $J_{\sigma(k)}, \ldots, J_{\sigma(1)}, J_{w(i)}$. We iteratively consider the jobs $J_{\sigma(1)}, \ldots, J_{\sigma(k)}$ in this order. Assume that we are considering $J_{\sigma(1)}$. Let $\propto_{w(i)+1}$ be identical to $\propto_{w(i)}$ except that $J_{w(i)}$ and $J_{\sigma(1)}$ are swapped. Observe that $\propto_u$ and $\prec_s$ are identical except for those jobs that are introduced during the first run phase. Therefore $J_{w(i)} \prec_s J_{\sigma(1)}$. Applying the order preserving property, we know that $J_{\sigma(1)} \prec_{s-1} J_{w(i)}$. Therefore, the swap between $J_{w(i)}$ and $J_{\sigma(1)}$ is legal. We now repeat this process where $\propto_{w(i)+\ell}$ is identical to $\propto_{w(i)+\ell-1}$ except that $J_{w(i)}$ and $J_{\sigma(\ell)}$ are swapped where $1 \leq \ell \leq k$. This marks the end of ith subphase. We set $w(i+1) = w(i) + k$ and find $\alpha(i+1) > \alpha(i)$ to start the next subphase. It is important to observe that the order-preserving property is satisfied since every swap in the previous phase involves only $J_{w(i)}$ whose $\propto_{w(i+1)}$ ranking is $\alpha(i)$. Finally observe that during an entire swap phase, every swap is legal and every pair of jobs is swapped at most once.

2.4. The Conflict Graph

In this subsection we show that every subgraph of the conflict graph is sparse, and hence that S can be scheduled on few processors. Recall that in the *conflict graph* $G = (\mathcal{J}, E)$ for S there is a vertex for each $J_i \in \mathcal{J}$, and two vertices J_i and J_j are adjacent if there exists a time t where both J_i and J_j are run at time t in S. The subgraph of $G = (\mathcal{J}, E)$ induced by a set $\mathcal{I} \subseteq \mathcal{J}$ is $G[\mathcal{I}] = (\mathcal{I}, (\mathcal{I} \times \mathcal{I}) \cap E)$. The *density* of the subgraph $G[\mathcal{I}]$ is $|(\mathcal{I} \times \mathcal{I}) \cap E|/|\mathcal{I}|$. Graph G is α *density bounded* if every induced subgraph of G has density at most α. Let $\delta(G[\mathcal{I}])$ be the minimum degree of a vertex in $G[\mathcal{I}]$. Let $\chi(G)$ be the chromatic number of G, that is, the fewest number of colors that is sufficient to color each vertex G in such a way that no pair of adjacent vertices are similarly colored.

LEMMA 2.6. *In a canonical schedule, no job J_i can be swapped to a higher $\propto$-rank more than $m - 1$ times.*

Proof. Let $\mathcal{I}$ be the collection of jobs J_j that were involved in a swap with J_i, with J_i being swapped to a higher $\propto$-rank. Assume to reach a contradiction that $|\mathcal{I}| \geq m$. Each $J_j \in \mathcal{I}$ has $C_j(S) \geq C_i(S)$ by the definition of legal swap. By lemma 2.4 it must then be the case that each $J_j \in \mathcal{I}$ is being run at time C_i. This is a contradiction since only m jobs may be run at any particular time. ∎

LEMMA 2.7. *The conflict graph G of every canonical schedule S is $3(m-1)$ density bounded.*

Proof. We actually bound the density of a super-graph H of G. Two jobs J_i and J_j are adjacent in H if there exists a t such that J_i and J_j are $\propto_t$-prime; the time t is a *certificate* for J_i and J_j. We think of H as being constructed sequentially over time. The structure of H at time t contains those edges with at least one certificate that is at most t.

We now argue that H is a super-graph of G. Consider an arbitrary edge, say between J_i and J_j in G. According to the schedule S, it must be the case that both J_i and J_j are run concurrently at some time t'. Therefore, both J_i and J_j must be $\propto_{t'}$-prime. Hence, we conclude that there must be an edge between J_i and J_j in H.

We give an amortization scheme that establishes $3(m-1)$ density bound. Let $\mathcal{I}$ be an arbitrary subset of $\mathcal{J}$.

Each $J_i \in \mathcal{I}$ is given $2(m-1)$ credits at the time F_i that J_i is inserted into a $\propto$ order. Furthermore, each job $J_j \in \mathcal{I}$ with $\propto_{F_i}$-rank between $m+1$ and $2m-1$, inclusive, is given one credit at time F_i. Hence, at time F_i the jobs in $\mathcal{I}$ accrue an aggregate total of at most $3(m-1)$ credits.

From here on, let J_i be an arbitrary job in $\mathcal{I}$. At time F_i, at most $m-1$ edges are incident on vertex J_i. These edges are paid for by $m-1$ of J_i's credits. If J_i's $\propto$-rank changes from $m+1$ to m at some time (note that this can happen either as the result of a $\propto$-prime swap or a job being completed), then any edges created at this time are charged to J_i. We now have to argue that this accounting is fair, that is, no job's account ever becomes negative.

Consider a time t such that the $\propto_{t-1}$-rank of J_i is $m+1$ and the $\propto_t$-rank of J_i is m. Let s be the latest time, strictly before time t, that J_i was

$\propto_s$-prime. If a new edge is added between J_i and J_j at time t, then either (i) J_j and J_i were involved in a swap (with J_i being swapped to a higher rank and J_j being swapped to a lower rank) since time s, or (ii) $s < F_j < t$.

In case (i), J_i's original credits are used to pay for the edge between J_i and J_j. By lemma 2.6, J_i can pick up at most $m-1$ charges from this case.

We consider two subcases of case (ii). In the first subcase assume that during the time interval (s,t) it is the case that J_i was always $\propto$-ranked $2m-1$ or less. In this case J_i must have received a credit at time F_j. This credit is used to pay for the edge between J_i and J_j.

In the second subcase of case (ii) assume that J_i was $\propto$-ranked higher than $2m-1$ at some time u, $s < u < t$. In this case, let Δ the collection of jobs involved in a swap with J_i, with J_i being swapped to a higher $\propto$-rank, during the time interval (s,u). Observe that $\Delta \leq m-1$ by lemma 2.6. There must have been $m-1-|\Delta|$ first run times during the time interval (s,u) since this is the only way, other than a swap that J_i's $\propto$-rank can increase. Hence, J_i picked up at least $m-1-|\Delta|$ credits during the time interval (s,u). The jobs in Δ must be $\propto_t$-prime at time t since they have completion times no earlier than J_i (and hence the jobs in Δ are still in the system at that time t) and by the definition of a legal swap. Hence, the number of J_j that are $\propto$-prime at time t and have $s < F_j < t$ is at most $m-1-|\Delta|$. Hence, the $m-1-|\Delta|$ credits earned by J_i during the time interval (s,u) are sufficient to pay for these edges. ▮

We now relate the density property proved in lemma 2.7 to the minimum degree in the induced subgraphs of G, and to the chromatic number of G.

Lemma 2.8. *If a graph G is γ density bounded, then for every subset S of the vertices in G, $\delta(G[S]) \leq 2\gamma$.*

Proof. Since G is γ density bounded, the number of vertices in $G[S]$ is at most $\gamma|S|$. Since each edge in $G[S]$ is incident on two vertices in $G[S]$, the average degree of a vertex in $G[S]$ is at most 2γ. Therefore, there is at least one vertex in $G[S]$ with degree at most 2γ. ▮

The following lemma can be found in [3]. For completeness we give a slightly modified version of the proof from [3]. Furthermore, note that this proof immediately yields an $O(n^2)$ time algorithm to construct the coloring.

Lemma 2.9. *For every graph $G = (V,E)$*

$$\chi(G) \leq 1 + \max_{U \subseteq V}(\delta(G[U]))$$

Proof. Let $k = \max_{U \subseteq V}(\delta(G[U]))$. Define a sequence $H_n \supset H_{n-1} \supset \ldots$ of spanned subgraphs of G as follows. Put $H_n = G$. For $\ell = n$ through

$\ell = 1$ do the following. Pick a vertex x_ℓ from H_ℓ with degree at most k in H_ℓ. Let $H_{\ell-1} = H_\ell - x_\ell$.

Observe that the set of vertices in the subgraph H_ℓ is $\{x_1, x_2, \ldots, x_\ell\}$. For $1 \leq \ell \leq n$, notice that x_ℓ is only adjacent to at most k vertices in $\{x_1, \ldots, x_{\ell-1}\}$. Hence, the greedy algorithm will use only $k+1$ colors if the vertices are considered in the order $x_1, x_2, \ldots, x_\ell$. ▮

LEMMA 2.10. *The conflict graph G of a canonical schedule S is $6m-5$ colorable.*

Proof. By lemma 2.7, G is $3(m-1)$ density bounded. By lemma 2.8, the minimum degree of every induced subgraph of G is at most $6(m-1)$. By lemma 2.9, G is $6m-5$ colorable. ▮

We are now ready to restate and prove theorem 1.1.

THEOREM 2.1. *$\alpha_m \leq 6m-5$, and hence $\omega_m \leq \frac{6m-5}{m}$.*

Proof. By lemma 2.1 and lemma 2.2 every schedule O on m machines can be transformed into a canonical schedule S on m machines. By lemma 2.10 the conflict graph G of S is $6m-5$ colorable. We can then form a feasible schedule on $6m-5$ machines by running all the jobs J_i, that are in a particular color class in G, on a single processor at the same times that they are run in S. Recall that one can then see that $\omega_m \leq \frac{6m-5}{m}$ by picking the jobs in the m color classes with maximum aggregate benefit. ▮

2.5. Polynomial Time Construction

In this subsection we show how to construct a pre-canonical schedule on $2m$ processors in polynomial time. We first construct an instance of the min-cost max-flow problem, which can be solved in polynomial time if all inputs are integer [7, 18].

The Min-cost Max-Flow Instance: Let O be the given original schedule where the jobs $J_1, \ldots, J_n$ are numbered by increasing order of their completion times, breaking ties arbitrarily. Let $\mathcal{I} = \{I_1, \ldots, I_u\}$ be a minimal cardinality partition of time into *intervals*, such that for each interval $I_i = [a_i, b_i]$, the jobs that are legal in the original schedule O are time invariant throughout I_i, that is, $\mathcal{L}_s(O) = \mathcal{L}_t(O)$ for all $s, t \in I_i$. Note that $u \leq 2n-1$. Also, assume that we number the intervals in such a way that for all $1 \leq i \leq u-1$, we have $a_i \leq b_i < a_{i+1} \leq b_{i+1}$. We introduce a vertex for each interval I_i with demand $m(b_i - a_i + 1)$. We introduce a vertex for each job J_j with supply x_j. For each interval I_i and each job J_j, for which J_j is legal throughout I_i, we introduce a directed edge from J_j to I_i with a capacity of $(b_i - a_i + 1)$, and a cost per unit flow of $(2n-i) \cdot j$. Further we add a dummy job J_0 (which corresponds to the idle times) with supply $mL - \sum_{j=1}^{n} x_j$, and add a directed edge from J_0 to every interval I_i with a capacity of $m(b_i - a_i + 1)$ and a cost per unit flow of 0 (the costs on these edges is actually irrelevant).

A feasible solution of this min-cost max-flow instance is an integer flow $f_{i,j}$, for each edge (J_j, I_i), such that (i) for every job J_j it is the case that $\sum_{i=1}^{u} f_{i,j} = x_j$, (ii) for every interval I_i it is the case that $\sum_{j=0}^{n} f_{i,j} = m(b_i - a_i + 1)$, and (iii) for all jobs J_j and all intervals I_i, $f_{i,j}$ is at most the capacity of that edge. An optimal solution is one that minimizes the sum over of all edges of the product of the flow on that edge times the cost per unit flow for that edge.

Note that a solution to this instance of min-cost max-flow does not uniquely specify a schedule, but each feasible schedule uniquely specifies a feasible solution to this instance of min-cost max-flow, and thus we may treat a feasible schedule as if it is a feasible solution to this min-cost max-flow instance. Since the original schedule O is a feasible solution to this instance of min-cost max flow, this instance of min-cost max-flow has an optimal solution $\mathcal{B}$. We now show how to transform $\mathcal{B}$ into a feasible schedule T on $2m$ machines.

Constructing a schedule T: Consider the optimal solution $\mathcal{B}$ to the above min-cost max-flow instance. We schedule each interval I_i separately. We now fix an interval I_i. Let $\mathcal{J}$ be the collection of jobs scheduled during I_i. We say that a job $J_j \in \mathcal{J}$ is *long* if $f_{i,j} \geq \frac{1}{2}(b_i - a_i + 1)$, and otherwise we say that $J_j \in \mathcal{J}$ is *short*. Let $\mathcal{S}$ be the collection of *short* jobs. We partition the $2m$ processors into $|\mathcal{J}| - |\mathcal{S}|$ *long* processors and $2m - (|\mathcal{J}| - |\mathcal{S}|)$ *short* processors. We schedule the jobs in I_i using a variant of list scheduling. That is, assume that we have scheduled the jobs $J_1, \ldots, J_{k-1}$, and that we now want to schedule J_k. Note that by the numbering of the jobs, $C_k(O) \geq C_j(O)$ for $1 \leq j \leq k-1$. Let s be the earliest time in I_i for which there is currently a short machine P_ℓ idle during $[s, b_i]$. If $J_k \in \mathcal{S}$ then J_k is scheduled from time s to time $s + f_{i,k}$ on P_ℓ (we postpone for the moment an explanation why it must be the case that $s + f_{i,k} \leq b_i$). Suppose $J_k \in \mathcal{J} - \mathcal{S}$. Select a long processor, say P_ℓ, that has not been used thus far during the interval I_i. If $s + f_{i,k} - 1 \leq b_i$ then J_k is scheduled from time s to time $s + f_{i,k}$ on P_ℓ Otherwise, J_k is scheduled from time $b_i - f_{i,k} + 1$ to time b_i on P_ℓ; in this case J_k is called a *border* job. Note that each long processor only schedules one job during I_i. The numbering of the jobs in the schedule T is exactly the same as the numbering of jobs in the schedule O.

LEMMA 2.11. *The schedule T is feasible.*

Proof. The only possible difficulty is if this procedure is not able to schedule some short job J_k on the short processors. However, then all of the short processors would be at least half full before J_k was assigned. Also, every long processor is also at least half full. Therefore, $\sum_{j=0}^{n} f_{i,j} >$

$2m\frac{(b_i - a_i + 1)}{2}$, a contradiction to one of the constraints specified in the min-cost max-flow instance. ∎

Observe that the graph under consideration for the min-cost max-flow problem has an edge between vertices J_i and I_j only when the job J_i is legal throughout the interval I_j. In addition, observe that the schedule T is constructed in such a way that a job J_i is run during I_j only when there is a positive flow from J_i to I_j. As a result, we observe below that the completion time of any job cannot be later in T than in O.

OBSERVATION 2.2. *For any job J_i, $C_i(T) \leq C_i(O)$.*

We now show that T is pseudo-canonical under the linear order $\lhd^O$.

LEMMA 2.12. *There is no $\lhd^O$-applicable switch (J_j, a, J_i, b) for T where a and b are in different intervals I_s and I_t, respectively.*

Proof. Assume to reach a contradiction that such a $\lhd^O$-applicable switch exists. By the applicability of the switch and the numbering of the intervals, it must be the case that $i < j$ and $s < t$. Let U be the schedule that results from applying this switch. The cost for the flow associated with U minus the costs of the flow associated with T is $\Delta = i(2n-s) - j(2n-s) + j(2n-t) - i(2n-t) = s(j-i) - t(j-i)$. If we can show that Δ is negative then this contradicts the assumption that T was constructed from an optimal solution to the min-cost max-flow problem. By algebraic simplification $\Delta < 0$ is equivalent to $t(j-i) > s(j-i)$, which follows since $i < j$ by the definition of $\lhd^O$-applicability and the numbering of the jobs, and $s < t$ by the definition of switch. ∎

LEMMA 2.13. *There is no $\lhd^O$-applicable switch (J_j, a, J_i, b) for T where a and b are in a common interval I_s.*

Proof. Assume for the moment that J_j is not a border job. Then by the property of list scheduling J_i is first run during I_s no later than J_j is first run during I_s. Hence, a must be after J_i last runs during I_s. This precludes the existence a time $b > a$ in I_s when J_i ejects J_j.

Now consider the cases that J_j is a border job. Then J_j is run continuously from when it is first run at time $b_s - f_{s,j} + 1$ to time b_s. This precludes the possibility that J_i ejects J_j during I_s after a time during I_s when J_j ejects J_i. ∎

Lemma 2.14 follows immediately from lemma 2.12 and lemma 2.13.

LEMMA 2.14. *The schedule T is a pseudo-canonical under the $\lhd^O$ relation.*

We now show that T is pre-canonical with respect to some ordering of the jobs.

LEMMA 2.15. *There exists a schedule S that is identical to T, except that the jobs are renumbered, such that S is pre-canonical under $\lhd^S$.*

Proof. We first give the numbering/ordering of the jobs in S. Suppose J_i and J_j are two arbitrary jobs. If $C_i(T) < C_j(T)$ then $i < j$. On the other hand, if $C_i(T) = C_j(T)$ then $i < j$ if and only if the numbering of J_i in O is smaller than the numbering of J_j in O. It is easy to see that one can number jobs in this fashion. Throughout the rest of this proof we retain the naming/numbering/ordering of the jobs in T, so J_1 refers to the job numbered 1 in T.

Observe that the completion time preserving order $\lhd^S$ has the following properties. If $C_i(T) < C_j(T)$ then $J_i \lhd^S J_j$. If $C_i(T) = C_j(T)$ then $J_i \lhd^S J_j$ if and only if $J_i \lhd^O J_j$.

Now assume to reach a contradiction that there exists a $\lhd^S$-applicable switch (J_j, a, J_i, b) in S. By the definition of S it must also be the case that (J_j, a, J_i, b) is a $\lhd^S$-applicable switch (J_j, a, J_i, b) in T. We break the proof into two cases.

In the first case, suppose that $J_i \lhd^O J_j$. We get a contradiction since (J_j, a, J_i, b) would be a $\lhd^O$-applicable switch in T, which is not possible by lemma 2.14.

In the second case suppose that $J_j \lhd^O J_i$. According to the ordering of the jobs in S, we can conclude that $C_i(T) < C_j(T)$. But, by the numbering of jobs in O, and the fact that $J_j \lhd^O J_i$, we can conclude that $C_j(O) \leq C_i(O)$. Applying observation 2.2, $C_j(T) \leq C_j(O) \leq C_i(O)$. Therefore, since $C_i(T) < C_j(T)$ and $C_j(T) \leq C_i(O)$, there must exist a time $t > b$ when J_j ejects J_i in T. As a result, (J_i, b, J_j, t) is a $\lhd^O$-applicable switch in T. This is a contradiction to lemma 2.14. ∎

Then by lemma 2.2, T is canonical under the linear order $\ll^T$, with this new ordering of the jobs

3. USING MIGRATION ELIMINATION IN A CORRECTNESS PROOF

We now give an example of the usefulness of theorem 2.1 by restating and proving theorem 1.2, that is, there is a polynomial time 6-approximation reduction REPEAT-A from the problem $P \mid r_i, pmtn \mid \sum w_i(1 - U_i)$ to the problem $1 \mid r_i, pmtn \mid \sum w_i(1 - U_i)$.

REPEAT-A: Let A be an algorithm under consideration for the problem $1 \mid r_i, pmtn \mid \sum w_i(1 - U_i)$. We iteratively schedule the processors from P_1 to P_m. Let $\mathcal{J}_{j-1}$ be the jobs scheduled on $P_1, \ldots, P_{j-1}$. Assume that we are now ready to schedule P_j. We use an algorithm A on input $\mathcal{J} - \mathcal{J}_{j-1}$ to produce a schedule S_j. Schedule S_j on P_j.

Note that an algorithm A has performance ratio c, if for all inputs $\mathcal{I}$, we have $c \cdot |A(\mathcal{I})| \geq \textsc{Opt}(\mathcal{I})$. Let $|\mathcal{J}_j|$ be the sum of the weights of the jobs in $\mathcal{J}_j$. Similarly, let $|S_i|$ be the sum of the weights of the jobs completed by the schedule S_i.

THEOREM 3.1. *Suppose A has performance ratio c for the problem $1 \mid r_i, pmtn \mid \sum w_i(1-U_i)$. Then* REPEAT-A *has performance ratio* $6c$ *for the problem $P \mid r_i, pmtn \mid \sum w_i(1-U_i)$.*

Proof. Consider an arbitrary collection $\mathcal{J}$ of jobs. Let $\text{N}\textsc{Opt}$ be the optimal nonmigratory schedule on m processors. Note that $3|\text{N}\textsc{Opt}| \geq |\textsc{Opt}|$ by theorem 1.1. We consider two cases.

In the first case assume that $|\text{N}\textsc{Opt}| - \sum_{i=1}^{m} |S_i| \leq |\text{N}\textsc{Opt}|/2$. Hence, $2\sum_{i=1}^{m} |S_i| \geq |\text{N}\textsc{Opt}|$, and $6\sum_{i=1}^{m} |S_i| \geq |\textsc{Opt}|$.

In the second case assume that $|\text{N}\textsc{Opt}| - \sum_{i=1}^{m} |S_i| > |\text{N}\textsc{Opt}|/2$. Let $\text{N}\textsc{Opt}_j$ denote the schedule $\text{N}\textsc{Opt}$ with the jobs in $\mathcal{J}_{j-1}$ deleted. In this case observe that for all j, $1 \leq j \leq m$, $|\text{N}\textsc{Opt}_j| > |\text{N}\textsc{Opt}|/2$. Now consider the time that the algorithm is trying to schedule a processor P_j, $1 \leq j \leq m$. In the schedule $\text{N}\textsc{Opt}_j$ there must be a processor P_i, such that the benefit of those jobs that are scheduled on P_i is at least $\frac{|\text{N}\textsc{Opt}_j|}{m} > \frac{|\text{N}\textsc{Opt}|}{2m}$. Observe that the previous statement need not be true if $\text{N}\textsc{Opt}$ was migratory. Hence, $|S_i| \geq \frac{|\text{N}\textsc{Opt}|}{2cm}$, and $\sum_{i=1}^{m} |S_i| \geq \frac{|\text{N}\textsc{Opt}|}{2c}$. Hence, $6c\sum_{i=1}^{m} |S_i| \geq |\textsc{Opt}|$. ∎

Note that the proof of correctness critically uses theorem 2.1. Also note that there is a pseudo-polynomial time algorithm (see [14]) that exactly solves $1 \mid r_i, pmtn \mid \sum w_i(1-U_i)$. This problem is one of the few natural scheduling problems for which a pseudo-polynomial time algorithm is known, but the standard methods of turning this pseudo-polynomial time algorithm into a fully polynomial time approximation scheme do not apply [20].

Acknowledgments: We would like to thank Eric Torng, Joel Wein, Giri Narasimhan, and Koren Gilad for helpful discussions. We would also like to thank the referees for many helpful comments.

REFERENCES

1. S. Baruah, J. Haritsa, and N. Sharma, "On-line scheduling to maximize task completions", *IEEE Real-time Systems Symposium*, 228 – 237, 1994.
2. S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang, "On the competitiveness of on-line real-time task scheduling", *Journal of Real-Time Systems*, **4**, 124–144, 1992.

3. B. Bollobas, *Extremal Graph Theory*, Academic Press, 1978.

4. P. Brucker, *Scheduling Algorithms*, Springer-Verlag, 1995.

5. P. Brucker, J. Hurink, S. Knust, "Complexity results of scheduling problems", http://www.mathematik.uni-osnabrueck.de/research/OR/class.

6. M. Dertouzos, and A. Mok, "Multiprocessor online scheduling of hard real-time tasks", *IEEE Transactions on Software Engineering*, **15**, 1497 – 1506, 1989.

7. J. Edmonds, and R. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems", *Journal of the ACM*, **19**, 248 –264, 1972.

8. L. Hall, "Approximation algorithms for scheduling", *Approximation Algorithms for NP-hard Problems*, ed. D. Hochbaum, PWS Publishing, 1997.

9. B. Kalyanasundaram and K. Pruhs "Speed is as powerful as clairvoyance", *IEEE Foundations of Computer Science*, 214–223, 1995.

10. B. Kalyanasundaram and K. Pruhs "Maximizing Job Completions Online", *European Symposium on Algorithms*, 235 – 246, 1998.

11. G. Koren, A. Amir, and E. Dar, "The Power of Migration in Multi-Processor Scheduling of Real-Time Systems", *ACM/SIAM Symposium on Discrete Algorithms*, 226 – 235, 1998.

12. G. Koren and D. Shasha, "MOCA: A multiprocessor on-line competitive algorithm for real-time systems scheduling", *Theoretical Computer Science*, **128**, 75–97, 1994.

13. T. Lam, and K. To, "Trade-offs between speed and processor in hard-deadline scheduling", *ACM/SIAM Symposium on Discrete Algorithms*, 623 – 632, 1999.

14. E. Lawler, "A dynamic programming algorithm for preemptive scheduling on a single machine to minimize the number of late jobs", *Annals of Operations Research*, **26**, 125 – 133, 1990.

15. E. Lawler, J.K. Lenstra, A. Rinnooy Kan, and D. Shmoys, "Sequencing and scheduling: algorithms and complexity", *Logistics of Production and Inventory*, eds. S. Graves, A. Rinnooy Kan, and P. Zipkin, Elsevier Science, 1993.

16. C. Phillips, C. Stein, E. Torng, and J. Wein, "Optimal time-critical scheduling via resource augmentation", *ACM Symposium on Theory of Computing*, 140 – 149, 1997.

17. Jiri Sgall, "On-line scheduling" *On-Line Algorithms: The State of the Art*, eds. A. Fiat and G. Woeginger, Lecture Notes in Computer Science 1442, Springer-Verlag, 1998.

18. R. Tarjan, *Data Structures and Network Algorithms*, SIAM, 1983.

19. G. Woeginger, "On-line scheduling of jobs with fixed start and end time", *Theoretical Computer Science*, **130**, 5–16, 1994..

20. G. Woeginger, "When does a dynamic programming formulation guarantee the existence of an FPTAS?", *ACM/SIAM Symposium on Discrete Algorithms*, 820 – 829, 1999.