The Power of Choice in Scheduling Parallel Tasks*

Walter Ludwig
Prasoon Tiwari

Technical Report #1190

November 1993

The Power of Choice in Scheduling Parallel Tasks*

Walter Ludwig Prasoon Tiwari

Computer Science Department
University of Wisconsin–Madison
Madison WI 53706
{ludwig,tiwari}@cs.wisc.edu

November 22, 1993

Abstract

A *malleable* parallel task is one that can be executed on any number of processors, with its execution time being a function of the number of processors allotted to it. A *nonmalleable* parallel task is one that requires a specific number of processors. Given n independent parallel tasks and m identical processors, we consider the problem of scheduling these tasks to minimize average completion time. Even though this is a classical scheduling problem, the first results in this direction were obtained only recently by Turek, Schwiegelshohn, Wolf, and Yu [Proceedings of the Fifth SIAM Symposium on Discrete Algorithms, 1994]. They presented an algorithm for scheduling nonmalleable parallel tasks with an approximation factor of 32. We give an algorithm for scheduling malleable tasks that has an approximation factor of 2, under the assumption that for each task, work is a nondecreasing function of the number of processors allotted to the task. Our algorithm and analysis incorporate an improved lower bound on average completion time for the nonmalleable problem. We also show that Turek et al.'s algorithm for the nonmalleable problem can be extended to the malleable problem, while preserving the approximation factor. The resulting algorithm does not require any assumptions on the task execution times. The running time of our malleable schedulers is dominated by the cost of determining a minimum weight perfect matching in a bipartite graph.

*Partially supported by National Science Foundation grant CCR-9024516.

1 Introduction

We consider the problem of scheduling parallel tasks to minimize average completion time. Suppose that we are given n independent tasks. Each of the tasks can be executed by any number of processors, and its execution time is a function of the number of processors allotted to it. The number of processors allotted to a particular task remains fixed for the duration of its execution. We are also given m identical processors with which to execute the tasks. The goal is to construct a nonpreemptive schedule that minimizes the average completion time of the tasks. We call this the *malleable average completion time* problem (MACT).

The *nonmalleable* average completion time problem (NACT) is the same, except that the number of processors required for each task is specified as part of the problem instance.

MACT is a generalization of NACT, and NACT is NP-complete in the strong sense [BLK83]. Therefore, we will consider allot approximation algorithms for these problems. We will evaluate these algorithms in terms of approximation factor and running time. Note that we use the term 'running time' to refer to the time required by an algorithm to construct a schedule. On the other hand, we use 'execution time' to refer to the time required to complete a given task, and 'completion time' to refer to the time in a given schedule when a given task is completed.

Scheduling to minimize average completion time is well-studied in the case of tasks that each require a single processor, and many variations have been considered. (See [LLKS92] for a survey.) However, results for the natural extension from single-processor tasks to parallel tasks have been harder to come by. It was only recently that Turek, Schwiegelshohn, Wolf, and Yu [TSWY94] demonstrated a NACT algorithm with an approximation factor of 32, the first proof of a constant approximation factor for NACT. Contrast this to the problem of scheduling parallel tasks to minimize total schedule length (or *makespan*), for which it has long been known that a simple list scheduling approach suffices to guarantee an approximation factor of 2 [GG75]. Furthermore, several authors have investigated scheduling malleable parallel tasks, and all of them have been concerned about makespan, and not about average completion time. (See, for example, [TWY92, LT94, BB90, DL89, FKST93, CYW92].)

Our main result is an algorithm for MACT with an approximation factor of 2, under the assumption that for each task, the work required is nondecreasing with the number of processors allotted to it (Theorem 5). (The *work* required to complete a malleable task is the product of the number of processors allotted to it and its execution time for that number of processors.)

This result is built upon three preliminary results, which are of independent interest. The first of these results is an algorithm ALLOT that finds an allotment of processors to malleable tasks that is optimal in the following sense. When the allotment is applied to the tasks, and then they are scheduled using a NACT algorithm, the resulting two-step MACT algorithm has the same approximation factor as the NACT algorithm (Theorem 3). This MACT algorithm does not require any assumptions on the task execution times. Thus ALLOT can be used to construct a MACT algorithm with approximation factor 32.

The two-step approach to malleable task scheduling was introduced by Turek, Wolf, and Yu [TWY92] for the problem of minimizing makespan. Their algorithm generates a set of at most mn allotments, one of which is the allotment of an optimal schedule (although it is not known which allotment). Then an algorithm for the nonmalleable makespan problem is used in conjunction with each of the generated allotments, and the best of the $O(mn)$ schedules is selected. The present authors [LT94] improved this result, demonstrating an algorithm with running time $O(mn)$ that finds a single allotment that can be used in conjunction with an algorithm for the nonmalleable makespan problem, preserving the approximation factor of that algorithm. Thus, our algorithm *adds* only an $O(mn)$ term to the running time of the nonmalleable makespan algorithm, rather than *multiplying* it by $O(mn)$. Furthermore, if for each task, work is nondecreasing and execution time is nonincreasing with the number of processors allotted to it, the running time of the allotment selection algorithm can be improved to $O(n \log^2 m)$.

The second result gives an upper bound on approximation factor for NACT in terms of the maximum processor requirement over all the tasks: a lower maximum processor requirement results in a better upper bound (Theorem 2). (Feldmann, Kao, Sgall, and Teng [FKST93] applied this idea to the problem of scheduling parallel tasks on-line to minimize makespan.) In particular, if no task requires more than half of the processors, then an approximation factor of 2 can be achieved.

This result can be used as follows to improve the approximation factor for MACT in the case when, for each task, work is nondecreasing with the number of processors. First use ALLOT to find an allotment, and then modify the allotment so that no task uses more than half of the processors.

The third result is a new lower bound for NACT, on which the previous result depends (Theorem 1). Our bound can be viewed as an extension of the bound due to Eastman, Even, and Isaacs [EEI64] for weighted single-processor tasks. It is an improvement of the "squashed area" bound used by Turek et al. [TSWY94] in the analysis of their NACT algorithm.

This work was motivated by scheduling problems arising from the Condor distributed batch system [LLM88, LL90]. Other problems arising in this context are discussed in Section 7.

The rest of the paper is organized as follows. In Section 2, we give an overview of our results, stating the main theorems. In Section 3, we present a new lower bound for NACT. In Section 4, we consider a simple NACT algorithm, and prove an approximation factor in terms of m and the maximum number of processors required by any task. In Section 5, we present a method of choosing an allotment of processors to tasks for MACT. In Section 6, we demonstrate a MACT algorithm with approximation factor 2 for the case when work is nondecreasing. Finally, Section 7 contains conclusions and open problems.

2 Statement of Results

Suppose we are given a set $\mathcal{N}$ of n independent nonmalleable tasks, and m identical processors. Associated with each task $T_i \in \mathcal{N}$ is an execution time requirement t_i and a processor requirement p_i. The task T_i requires p_i processors for the full duration of its execution. Let c_i denote the completion time of task T_i in a given schedule. Let $C = \sum_{i=1}^{n} c_i$ denote the sum of the task completion times for a given schedule. The objective is to construct a schedule that minimizes C. This amounts to minimizing the average completion time. Let C_{opt} denote the optimal value of C. Let $C_{\mathcal{A}}$ denote the value of C for the schedule that results from applying algorithm $\mathcal{A}$.

To prove an approximation factor, some lower bound on C_{opt} is needed. A simple lower bound for C_{opt} is given by

$$H = \sum_{i=1}^{n} t_i, \tag{1}$$

the sum of the task execution times. Another lower bound is based on the amount of work required by each task. Suppose that the tasks are ordered such that $p_i t_i \geq p_{i+1} t_{i+1}$ for all $i \in [n-1]$, where $[n-1]$ denotes the set of integers $\{1, 2, \ldots, n-1\}$. Let

$$A = \frac{1}{m} \sum_{i=1}^{n} i p_i t_i. \tag{2}$$

Turek et al. [TSWY94] call this the "squashed area" bound, and they proved that it is a lower bound on C_{opt}. This bound can be viewed in the following way. Replace each task T_i with a task that has the same work requirement, but that has processor requirement m. (Then its execution time requirement is $p_i t_i / m$.) If these new tasks are scheduled in order of nondecreasing work, then the sum of their completion times is precisely A.

These two lower bounds are used by Turek et al. [TSWY94] to prove an approximation factor of 32 for their NACT algorithm. We present a new lower bound for NACT that dominates A, as stated in the following theorem.

Theorem 1 *For any instance of NACT, suppose that the tasks are ordered such that $p_i t_i \geq p_{i+1} t_{i+1}$ for all $i \in [n-1]$. Then*

$$C_{opt} \geq \frac{1}{m} \sum_{i=1}^{n} (i - \frac{1}{2}) p_i t_i + \frac{1}{2} \sum_{i=1}^{n} t_i. \tag{3}$$

Our bound can be expressed as $A + \frac{1}{2}H - \frac{1}{2}W$, where

$$W = \frac{1}{m} \sum_{i=1}^{n} p_i t_i \tag{4}$$

is the total work requirement divided by the number of processors. Observe that $H \geq W$, and therefore our bound dominates A.

Consider the following algorithm for NACT. Sort the tasks according to nondecreasing work requirement. Then schedule them in order, beginning the execution of each task as soon as enough processors become available. This is the *least work first* (LWF) algorithm.

The improved lower bound of Theorem 1 is essential to the following result for LWF.

Theorem 2 *For any instance of NACT, let $r = \max_i\{p_i\}$ be the maximum number of processors required by any task. Let $\rho = \max\{\frac{m}{m-r+1}, 2\}$. Then*

$$C_{\text{LWF}} \leq \rho C_{opt}. \tag{5}$$

The running time of LWF is $O(n \log n)$.

Unlike NACT, in MACT we choose the allotment of processors to tasks, and therefore we determine r. We will make use of this to show a 2-approximate algorithm for MACT under the assumption that for each task, work is nondecreasing with the number of processors.

Let $\mathcal{M}$ be a set of n independent malleable tasks. In MACT, the execution time of task T_i is specified by a function $p_i : [m] \rightarrow I\!R$. Let C_{opt}^M denote the value of C for the optimal schedule of the given problem instance[1]. Let $C_{\mathcal{A}}^M$ denote the value of C for the schedule that results from applying MACT algorithm $\mathcal{A}$ to the given problem instance.

[1]We use the superscript M to distinguish the malleable case from the nonmalleable case.

Given an instance of MACT, an instance of NACT can be derived by choosing an allotment $\bar{p} = (p_1, p_2, \ldots, p_n)$ of processors to the tasks of the MACT instance. We call this the $\bar{p}$-*derived* instance. This instance can then be scheduled using an algorithm for NACT. This two-step process constitutes an algorithm for MACT. We would like to be able to choose $\bar{p}$ so that the two-step MACT algorithm has the same approximation factor as the NACT algorithm. Our next result shows that this can be done if a lower bound on C_{opt} of a certain form is used to prove the NACT algorithm's approximation factor.

Theorem 3 *Suppose that $L = \lambda_1 A + \lambda_2 H + \lambda_3 W$ is a lower bound on C_{opt} for NACT, where λ_1, λ_2, and λ_3 are constants, and $\lambda_1 > 0$. Then if we are given a NACT algorithm $\mathcal{A}$ with running time $R(m,n)$ such that $C_{\mathcal{A}} \leq \rho L$, then there is a MACT algorithm $\mathcal{A}'$ with running time $O(n^3 + mn^2) + R(m,n)$ such that $C^M_{\mathcal{A}'} \leq \rho C^M_{opt}$.*

Observe that since A and H are both lower bounds on C_{opt}, for any $\lambda_1, \lambda_2 \geq 0$ such that $\lambda_1 + \lambda_2 = 1$, we have $\lambda_1 A + \lambda_2 H \leq C_{opt}$. ($W$ is also a lower bound on C_{opt}, but $W \leq A$ and $W \leq H$.) Turek et al. [TSWY94] use the lower bound $\frac{1}{2}A + \frac{1}{2}H$ to prove an approximation factor of 32 for their NACT algorithm, which has running time $O(n \log n)$. Therefore, we get the following result.

Corollary 4 *There is a MACT algorithm with running time $O(n^3 + mn^2)$ and approximation factor 32.*

The first step of the algorithm $\mathcal{A}'$ of Theorem 3 is to choose an allotment $\bar{p}$ using an algorithm that we call ALLOT. Out of all possible allotments, ALLOT chooses one that minimizes the lower bound L for the derived NACT instance. The value of L for that allotment is thus a lower bound for the MACT instance. Then applying the NACT algorithm $\mathcal{A}$ gives the desired approximation factor.

ALLOT consists primarily of constructing a weighted complete bipartite graph with $2n$ vertices, and then finding a minimum-weight perfect matching in G using the Hungarian method [Kuh55]. Under the nondecreasing work assumption, computing the edge weights involves finding the row maxima of certain totally monotone matrices. This is accomplished using the fast matrix-searching algorithm due to Aggarwal et al. [AKM+87]. The running time of ALLOT in this case is $O(n^3 + mn)$. Otherwise, its running time is $O(n^3 + mn^2)$.

Let $w_i(p_i) = p_i * t_i(p_i)$ denote the amount of work required by task T_i when p_i processors are allotted to it. Suppose that for each task, work is nondecreasing with the number of processors, i.e., $w_i(j) \leq w_i(j+1)$ for all $j \in [m-1]$. Then a much better approximation

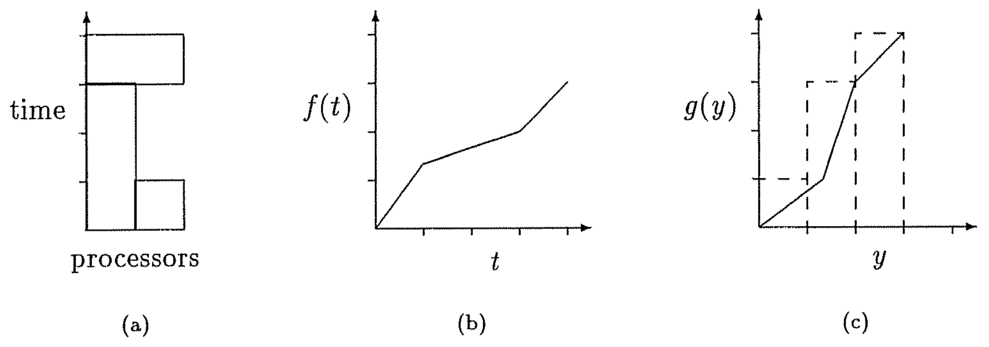

Figure 1: A schedule and its corresponding functions $f(t)$ and $g(y)$

factor can be achieved using the following algorithm. Use ALLOT to find an allotment $\bar{p}$ that minimizes the lower bound of Theorem 1. Then construct an allotment $\bar{q}$ by setting $q_i = \min\{p_i, \lceil \frac{m}{2} \rceil\}$ for all $i \in [n]$. Finally, schedule the $\bar{q}$-derived NACT instance using LWF. Call this algorithm MLWF. We will prove the following theorem.

Theorem 5 *Suppose that $w_i(j) \leq w_i(j+1)$ for all $j \in [m-1]$. Then*

$$C^M_{\text{MLWF}} \leq 2C^M_{opt}. \tag{6}$$

The running time of MLWF is $O(n^3 + mn)$.

3 The NACT Lower Bound

Given a schedule for a set of nonmalleable tasks $\mathcal{N}$, let s_i denote the starting time of task T_i. (Thus, $c_i = s_i + t_i$.) Let $c_{\max} = \max_{i \in [n]}\{c_i\}$ be the *makespan* of the schedule.

Let $E(t) = \{T_i \in \mathcal{N} : s_i \leq t < s_i + t_i\}$ be the set of tasks that are being executed at time t. Let $F(t) = \{T_i \in \mathcal{N} : s_i + t_i \leq t\}$ be the set of tasks that have been completed at time t. Let $f(t) = |F(t)| + \sum_{T_i \in E(t)} \frac{t-s_i}{t_i}$ be the number of tasks that have been completed at time t, including fractional tasks. Let $g(y) = \min\{t : f(t) = y\}$ be the time required to complete y tasks. Observe that if $f(t)$ is strictly increasing, that is, if at all times for the duration of the schedule at least one task is being executed, then $g = f^{-1}$.

See Figure 1 for an example schedule and a graph of the corresponding functions $f(t)$ and $g(y)$. The dashed boxes in Figure 1(c) correspond to the completion times of the tasks. The sum of the areas of the dashed boxes is the sum of the task completion times. It seems natural that the function $g(y)$, which gives the time required to complete y tasks, should be closely related to $\sum c_i$. The following lemma states the precise relationship.

Lemma 6 *Any schedule of $\mathcal{N}$ satisfies*

$$\sum_{i=1}^{n} c_i = \int_0^n g(y)dy + \frac{1}{2}\sum_{i=1}^{n} t_i. \tag{7}$$

Interpreting this lemma geometrically, it says that the area inside the dashed boxes that is above the plot of $g(y)$ is equal to half the sum of the task execution times. In order to prove the lemma, we make use of the following fact.

Lemma 7 *Suppose $a, b \in \mathbb{R}$ are such that $f(a) = b$. then*

$$\int_0^a f(t)dt + \int_0^b g(y)dy = ab. \tag{8}$$

The geometric interpretation of this lemma is as follows. The rectangle with vertices $(0,0)$, $(a,0)$, (a,b), and $(b,0)$ has area ab and is divided into two parts by the plot of $f(t)$. The area on one side of the plot is $\int_0^a f(t)dt$, and the area on the other side is $\int_0^b g(y)dy$.

Proof of Lemma 6: We proceed by induction on n. For the basis, suppose that $n = 1$. Then $g(0) = 0$, $g(1) = c_1$, and $g(y) = s_1 + yt_1$ for $y \in (0, 1]$. Then

$$\int_0^1 g(y)dy = s_1 + \frac{1}{2}t_1. \tag{9}$$

Therefore,

$$\int_0^1 g(y)dy + \frac{1}{2}t_1 = s_1 + t_1 = c_1. \tag{10}$$

For the induction step, suppose that $n > 1$. Let $\mathcal{N}_1 = \{T_1, T_2, \ldots T_{n-1}\}$. Let $\mathcal{N}_2 = \{T_n\}$. Viewing the schedules of $\mathcal{N}_1$ and $\mathcal{N}_2$ as being independent, let $f_1(t)$ and $g_1(y)$ be defined appropriately for the schedule of $\mathcal{N}_1$, and let $f_2(t)$ and $g_2(y)$ be defined appropriately for the schedule of $\mathcal{N}_2$. Observe that $f(t) = f_1(t) + f_2(t)$.

By the induction hypothesis, we have

$$\sum_{i=1}^{n-1} c_i = \int_0^{n-1} g_1(y)dy + \frac{1}{2}\sum_{i=1}^{n-1} t_i \tag{11}$$

and

$$c_n = \int_0^1 g_2(y)dy + \frac{1}{2}t_n \tag{12}$$

Therefore it is sufficient to show that

$$\int_0^n g(y)dy = \int_0^{n-1} g_1(y)dy + \int_0^1 g_2(y)dy,$$

and then we get the desired result by adding equations (11) and (12) together.

Now $f(c_{\max}) = n$, $f_1(c_{\max}) = n-1$, and $f_2(c_{\max}) = 1$, so by Lemma 7, we have

$$\begin{aligned}\int_0^n g(y)dy &= c_{\max}n - \int_0^{c_{\max}} f(t)dt \\ &= c_{\max}n - \int_0^{c_{\max}} (f_1(t) + f_2(t))dt \\ &= c_{\max}(n-1) - \int_0^{c_{\max}} f_1(t)dt + c_{\max} - \int_0^{c_{\max}} f_2(t)dt \\ &= \int_0^{n-1} g_1(y)dy + \int_0^1 g_2(y)dy.\end{aligned}$$

□

We are now ready to proceed with the proof of Theorem 1. Recall that we are assuming that $p_i t_i \geq p_{i+1}t_{i+1}$ for all $i \in [n-1]$.

Proof of Theorem 1: By Lemma 6, we know that

$$\sum_{i=1}^{n} c_i = \int_0^n g(y)dy + \frac{1}{2}\sum_{i=1}^{n} t_i. \tag{13}$$

So we want a lower bound on $g(y)$. That is, for any given $y \in [0, n]$, we want a lower bound on the time required to complete y tasks. For each $i \in [n]$, let $x_i \in [0,1]$ denote the fraction of task T_i that is completed. The value of the following optimization problem gives us the bound we seek.

$$\begin{array}{ll}\min_{x_i} & \frac{1}{m}\sum_{i=1}^{n} p_i t_i x_i \\ \text{subject to} & \sum_{i=1}^{n} x_i = y\end{array} \tag{14}$$

Let $q = n - \lfloor y \rfloor$. Then it can be shown that the optimal solution of (14) is

$$\begin{array}{ll} x_i^\star = 1 & \text{for } i \in \{q+1, q+2, \ldots, n\} \\ x_i^\star = 0 & \text{for } i \in \{1, 2, \ldots, q-1\} \\ x_q^\star = y - \lfloor y \rfloor. & \end{array}$$

Let $g^\star(y)$ denote the value of (14) as a function of y. Then

$$g^\star(y) = \frac{1}{m}\sum_{i=1}^{n} p_i t_i x_i^\star.$$

The next step is to determine the value of $\int_0^n g^\star(y)dy$. One way to do this is by observing that $g^\star(y)$ corresponds to the optimal m-processor schedule of a task set $\mathcal{N}'$ in which each task T_i' requires $p_i' = m$ processors and has execution time $t_i' = p_i t_i / m$. In this schedule, the completion time of task T_i' is given by $c_i' = \sum_{l=i}^n t_l'$. Then applying Lemma 6, we have

$$\begin{aligned}\int_0^n g^\star(y)dy &= \sum_{i=1}^n c_i' - \frac{1}{2}\sum_{i=1}^n t_i' \\ &= \sum_{i=1}^n \sum_{l=i}^n t_l' - \frac{1}{2}\sum_{i=1}^n t_i' \\ &= \sum_{i=1}^n i t_i' - \frac{1}{2}\sum_{i=1}^n t_i' \\ &= \frac{1}{m}\sum_{i=1}^n (i - \frac{1}{2}) p_i t_i.\end{aligned}$$

Then the result follows from (13). □

4 Analysis of LWF for Scheduling Nonmalleable Tasks

Recall that the LWF algorithm for NACT is simply to schedule the tasks in order of non-decreasing work. In this section, we will prove Theorem 2, establishing an upper bound on the approximation factor of LWF. This can be achieved using a slightly weaker lower bound on C_{opt} than the bound we proved in Theorem 1. Let

$$\alpha = \frac{1}{m}\sum_{i=1}^n (i - \frac{1}{2}) p_i t_i + \frac{1}{2}\sum_{i=1}^n t_i \tag{15}$$

be the lower bound of Theorem 1. Let

$$\begin{aligned}\hat{\alpha} &= \frac{1}{m}\sum_{i=2}^n (i-1) p_i t_i + \frac{1}{2}\sum_{i=1}^n t_i \\ &= \sum_{i=1}^n \frac{1}{m}\sum_{l=i+1}^n p_l t_l + \frac{1}{2}\sum_{i=1}^n t_i.\end{aligned} \tag{16}$$

Then $\hat{\alpha} < \alpha$. We will use $\hat{\alpha}$ to prove the approximation factor for LWF. The contribution of task T_i to (16) is $\frac{1}{m}\sum_{l=i+1}^n p_l t_l + \frac{1}{2} t_i$. Let

$$a_i = \frac{1}{m}\sum_{l=i+1}^n p_l t_l. \tag{17}$$

Observe that

$$\hat{\alpha} = \sum_{i=1}^n (a_i + \frac{1}{2} t_i). \tag{18}$$

Then a_i can be viewed as a goal for the starting time of task T_i. In particular, if task T_i begins execution no later then δa_i for some δ, then its completion time is $\delta a_i + t_i$, which does not exceed the contribution of T_i to $\hat{\alpha}$ by more than a factor of $\max\{\delta, 2\}$. Then to prove an approximation factor for LWF, we only need to find δ such that every task satisfies $s_i \leq \delta a_i$.

Lemma 8 *Let $r = \max_{i\in[n]}\{p_i\}$ be the largest number of processors required by any of the tasks in $\mathcal{N}$. Then using LWF guarantees*

$$s_i \leq \frac{m}{m-r+1} a_i \tag{19}$$

for all $T_i \in \mathcal{N}$.

Proof: Consider a task $T_i \in \mathcal{N}$. Execution of task T_i is scheduled after tasks $T_{i+1}, T_{i+2}, \ldots, T_n$ are scheduled, but before any of the other tasks are scheduled. Then not more than $\sum_{k=i+1}^{n} p_k t_k$ work is done before T_i begins execution. Observe that there are not more than $r-1$ idle processors at any time when there is at least one task that has not been started yet. Then there are at least $m-r+1$ processors busy at all times until s_i. We conclude that

$$\begin{aligned} s_i &\leq \frac{1}{m-r+1} \sum_{k=i+1}^{n} p_k t_k \\ &= \frac{m}{m-r+1} a_i. \end{aligned}$$

□

Lemma 9 *Let $\rho = \max\{\frac{m}{m-r+1}, 2\}$. Then LWF satisfies*

$$C_{\text{LWF}} \leq \rho\hat{\alpha}. \tag{20}$$

Proof: By Lemma 8,

$$\begin{aligned} C_{\text{LWF}} &= \sum_{i=1}^{n} c_i \\ &= \sum_{i=1}^{n} (s_i + t_i) \\ &\leq \sum_{i=1}^{n} (\frac{m}{m-r+1} a_i + t_i) \\ &\leq \sum_{i=1}^{n} (\rho a_i + \rho \frac{1}{2} t_i) \\ &= \rho\hat{\alpha}. \end{aligned}$$

□

Proof of Theorem 2: By Lemma 9 and Theorem 1,

$$\begin{aligned} C_{\text{LWF}} &\leq \rho\hat{\alpha} \\ &< \rho\alpha \\ &\leq \rho C_{opt}. \end{aligned}$$

□

5 Choosing an Allotment of Processors to Malleable Tasks

In this section, we describe our algorithm ALLOT for finding an allotment of processors to a set $\mathcal{M}$ of malleable tasks. The allotment will be chosen to minimize a lower bound on C_{opt} for NACT. Given a MACT instance, the quantities A, H, and W introduced in Section 2 can be extended in a natural way to the $\bar{p}$-derived NACT instance. Let

$$H(\bar{p}) = \sum_{i=1}^{n} t_i(p_i). \tag{21}$$

Recall that $w_i(p_i) = p_i * t_i(p_i)$. Let

$$W(\bar{p}) = \frac{1}{m}\sum_{k=1}^{n} w_i(p_i). \tag{22}$$

Recall that computing the bound A requires ordering the tasks according to nonincreasing work. For malleable tasks, this order may vary for different allotments. For any permutation σ on $[n]$, let

$$\begin{aligned} A(\bar{p},\sigma) &= \frac{1}{m}\sum_{k=1}^{n} k w_{\sigma(k)}(p_{\sigma(k)}) \\ &= \frac{1}{m}\sum_{i=1}^{n} \sigma^{-1}(i) w_i(p_i). \end{aligned} \tag{23}$$

The lower bound that we are seeking is then

$$A(\bar{p}) = \min_{\sigma}\{A(\bar{p},\sigma)\}. \tag{24}$$

A permutation σ that minimizes $A(\bar{p},\sigma)$ is characterized by the following definition.

Definition 10 *An allotment* $\bar{p}$ induces a permutation σ *if* $w_{\sigma(k)}(p_{\sigma(k)}) \geq w_{\sigma(k+1)}(p_{\sigma(k+1)})$ *for all* $k \in [n-1]$.

The following lemma is an immediate consequence of (23) and Definition 10.

Lemma 11 *For any allotment $\bar{p}$, let σ be a permutation that is induced by $\bar{p}$. Then for any permutation τ, we have $A(\bar{p},\tau) \geq A(\bar{p},\sigma)$.*

Thus if $\bar{p}$ induces σ, then $A(\bar{p},\sigma) = A(\bar{p})$. Note that if $\bar{p}$ induces σ, then $\sigma(k) = i$ means that T_i is the kth largest task in terms of work when the allotment $\bar{p}$ is used.

Suppose that $L = \lambda_1 A + \lambda_2 H + \lambda_3 W$ is a lower bound on C_{opt} for NACT, where λ_1, λ_2, and λ_3 are constants, and $\lambda_1 > 0$. Let

$$L(\bar{p},\sigma) = \lambda_1 A(\bar{p},\sigma) + \lambda_2 H(\bar{p}) + \lambda_3 W(\bar{p}). \tag{25}$$

Then

$$\begin{aligned} L(\bar{p}) &= \min_{\sigma}\{L(\bar{p},\sigma)\} \\ &= \lambda_1 A(\bar{p}) + \lambda_2 H(\bar{p}) + \lambda_3 W(\bar{p}). \end{aligned}$$

is a lower bound on C_{opt} for the $\bar{p}$-derived instance. Furthermore,

$$\begin{aligned} L^M &= \min_{\bar{p},\sigma}\{L(\bar{p},\sigma)\} \\ &= \min_{\bar{p}}\{L(\bar{p})\} \end{aligned}$$

is a lower bound on C^M_{opt} for MACT. Our goal, then, is to find an allotment $\bar{p}$ that satisfies $L^M = L(\bar{p})$. But rather than finding an allotment directly, we will find it by way of a permutation that it induces.

Observe that

$$\begin{aligned} L^M &= \min_{\bar{p},\sigma}\{L(\bar{p},\sigma)\} \\ &= \min_{\bar{p},\sigma}\{\lambda_1 A(\bar{p},\sigma) + \lambda_2 H(\bar{p}) + \lambda_3 W(\bar{p})\} \\ &= \min_{\bar{p},\sigma}\{\lambda_1 \frac{1}{m}\sum_{i=1}^{n}\sigma^{-1}(i) w_i(p_i) + \lambda_2 \sum_{i=1}^{n} t_i(p_i) + \lambda_3 \frac{1}{m}\sum_{i=1}^{n} w_i(p_i)\} \\ &= \min_{\bar{p},\sigma}\{\sum_{i=1}^{n}[\frac{\lambda_1}{m}\sigma^{-1}(i) w_i(p_i) + \lambda_2 t_i(p_i) + \frac{\lambda_3}{m} w_i(p_i)]\}. \end{aligned} \tag{26}$$

If we know a permutation σ that is induced by an optimal allotment, then the contribution of each task to $L(\bar{p},\sigma)$ can be minimized individually as follows. Let

$$\ell_i(j,k) = \frac{\lambda_1}{m} k w_i(j) + \lambda_2 t_i(j) + \frac{\lambda_3}{m} w_i(j) \tag{27}$$

be the contribution of task T_i to (26) when it is allotted j processors and $\sigma(k) = i$. That is,

$$L(\bar{p}, \sigma) = \sum_{i=1}^{n} \ell_i(p_i, \sigma^{-1}(i)). \tag{28}$$

Then

$$\ell_i(k) = \min_{j \in [m]} \{\ell_i(j, k)\} \tag{29}$$

is the minimum possible contribution of task T_i to $L(\bar{p}, \sigma)$ if $\sigma(k) = i$. So for a given task T_i, if we know $\sigma^{-1}(i)$, then we can determine the allotment of processors p_i to T_i by choosing p_i such that

$$\ell_i(p_i, \sigma^{-1}(i)) = \ell_i(\sigma^{-1}(i)). \tag{30}$$

(There may be multiple choices of p_i that satisfy (30).) The allotment $\bar{p}$ that minimizes $L(\bar{p}, \sigma)$ for a given permutation σ is characterized by the following definition.

Definition 12 *A permutation σ* induces an allotment *$\bar{p}$ if $\ell_i(p_i, \sigma^{-1}(i)) = \ell_i(\sigma^{-1}(i))$ for all $T_i \in \mathcal{M}$.*

We can determine a permutation σ that is induced by an optimal allotment in the following way. We construct a full bipartite graph $G = (U, V, E)$. The set U contains one vertex for each task in $\mathcal{M}$. The set V contains one vertex for each integer in $[n]$. The vertices in V should be interpreted as possible values of $\sigma^{-1}(i)$ for each task T_i. Assign to edge (T_i, k) the weight $\ell_i(k)$. Find a minimum-weight perfect matching M in G. Let σ be the permutation that corresponds to the matching M. That is, if the edge (T_i, k) is in the matching M, then $\sigma(k) = i$. Then select an allotment $\bar{p}$ that is induced by σ. Call this algorithm ALLOT.

Observe that given an allotment, it is easy to find a permutation that it induces. But ALLOT finds an optimal allotment by first finding a permutation that it induces, and then finding an allotment that is induced by that permutation.

Lemma 13 *ALLOT finds an allotment $\bar{p}$ that satisfies $L(\bar{p}) = L^M$. The running time of ALLOT is $O(n^3 + mn)$ if $w_i(j) \leq w_i(j+1)$ for all $T_i \in \mathcal{M}$ and all $j \in [m-1]$. Otherwise, its running time is $O(n^3 + mn^2)$.*

For the proof, we will make use of some additional lemmas. Let

$$b(\sigma) = \sum_{k=1}^{n} \ell_{\sigma(k)}(k) \tag{31}$$

be the weight of permutation σ, which is the weight of the corresponding matching M in G.

Lemma 14 *For any allotment $\bar{p}$, let σ be a permutation that is induced by $\bar{p}$. Then $b(\sigma) \leq L(\bar{p})$.*

Proof: The weight of σ is given by

$$b(\sigma) = \sum_{i=1}^{n} \ell_i(\sigma^{-1}(i)). \tag{32}$$

By the definition of $\ell_i(j,k)$, we have

$$L(\bar{p}) = \sum_{i=1}^{n} \ell_i(p_i, \sigma^{-1}(i)). \tag{33}$$

By (29), $\ell_i(\sigma^{-1}(i)) \leq \ell_i(j, \sigma^{-1}(i))$ for any j. Therefore,

$$\begin{aligned} b(\sigma) &= \sum_{i=1}^{n} \ell_i(\sigma^{-1}(i)) \\ &\leq \sum_{i=1}^{n} \ell_i(p_i, \sigma^{-1}(i)) \\ &= L(\bar{p}). \end{aligned}$$

□

Lemma 15 *For any permutation σ, let $\bar{p}$ be an allotment that is induced by σ. Then $L(\bar{p}) \leq b(\sigma)$.*

Proof: The permutation σ induces the allotment $\bar{p}$, and therefore $\ell_i(p_i, \sigma^{-1}(i)) = \ell_i(\sigma^{-1}(i))$. Then the weight of σ is

$$\begin{aligned} b(\sigma) &= \sum_{i=1}^{n} \ell_i(\sigma^{-1}(i)) \\ &= \sum_{i=1}^{n} \ell_i(p_i, \sigma^{-1}(i)) \\ &= L(\bar{p}, \sigma) \\ &\geq L(\bar{p}). \end{aligned}$$

□

Lemma 16 *Let M be a minimum-weight perfect matching in G. Let σ be the permutation corresponding to M. Let $\bar{p}$ be an allotment induced by σ. Then $\bar{p}$ satisfies $L^M = L(\bar{p}) = b(\sigma)$.*

Proof: Let $\bar{q}$ be any allotment, and let τ be a permutation that is induced by $\bar{q}$. Then by Lemma 14,

$$L(\bar{q}) \geq b(\tau). \tag{34}$$

The permutation σ corresponds to a minimum-weight matching, so

$$b(\tau) \geq b(\sigma). \tag{35}$$

By Lemma 15

$$b(\sigma) \geq L(\bar{p}), \tag{36}$$

because σ induces $\bar{p}$.

Therefore, by (34), (35), and (36), we have

$$L(\bar{q}) \geq L(\bar{p}), \tag{37}$$

and thus $L^M = L(\bar{p})$.

Now suppose that $b(\sigma) > L(\bar{p})$. Let σ' be a permutation that is induced by $\bar{p}$. Then by Lemma 14,

$$L(\bar{p}) \geq b(\sigma'). \tag{38}$$

Then it follows that $b(\sigma) > b(\sigma')$, contradicting the fact that σ corresponds to a minimum-weight matching. Therefore, $b(\sigma) = L(\bar{p})$. □

Now that the correctness of ALLOT has been established, it only remains to determine its running time.

The first step in ALLOT is to construct the graph G. The edge weights $\ell_i(k)$ can be determined by computing $\ell_i(j,k)$ for each $j \in [m]$. Then a total of $O(mn^2)$ steps are required to construct G.

If the nondecreasing work assumption holds, then there is a faster way of computing the edge weights. The following lemma demonstrates a property of the values of $\ell_i(j,k)$ for a given task T_i that can be used to determine the edge weights quickly.

Lemma 17 *Suppose that task $T_i \in \mathcal{M}$ satisfies $w_i(j) \leq w_i(j+1)$ for all $j \in [m-1]$. Let X be an $n \times m$ matrix with entries $x_{kj} = -\ell_i(m+1-j,k)$. Then X is* totally monotone. *That is, for all $k_1 < k_2$ and $j_1 < j_2$, we have $x_{k_1 j_1} < x_{k_1 j_2}$ implies $x_{k_2 j_1} < x_{k_2 j_2}$.*

Observe that $-\ell_i(k)$ is the maximum entry in row k of X. Aggarwal et al. [AKM+87] have shown that the row maxima of a totally monotone matrix can be found by querying

$O(n+m)$ entries of the matrix. As each entry of X can be computed in constant time, we therefore can determine $\ell_i(k)$ for a given task T_i and all values of $k \in [n]$ in $O(n+m)$ steps. Thus it takes $O(n^2+mn)$ steps to construct the graph G under the nondecreasing work assumption.

Proof of Lemma 17: Suppose instead that there exist $k_1, k_2, j_1, j_2 \in [n]$ such that $k_1 < k_2$, and $j_1 < j_2$, and $x_{k_1 j_1} < x_{k_1 j_2}$, and $x_{k_2 j_1} \geq x_{k_2 j_2}$. Then we have

$$-\ell_i(m+1-j_1, k_1) < -\ell_i(m+1-j_2, k_1) \tag{39}$$

and

$$-\ell_i(m+1-j_1, k_2) \geq -\ell_i(m+1-j_2, k_2). \tag{40}$$

Observe that for any j, k_1, and k_2 we have

$$\ell_i(j, k_2) - \ell_i(j, k_1) = (k_2 - k_1)\frac{\lambda_1}{m} w_i(j). \tag{41}$$

Then rewriting (40) as

$$\ell_i(m+1-j_1, k_2) \leq \ell_i(m+1-j_2, k_2), \tag{42}$$

and adding (39) and (42), we get

$$(k_2 - k_1)\frac{\lambda_1}{m} w_i(m+1-j_1) < (k_2 - k_1)\frac{\lambda_1}{m} w_i(m+1-j_2). \tag{43}$$

Therefore we have

$$w_i(m+1-j_1) < w_i(m+1-j_2). \tag{44}$$

But observe that $m+1-j_1 > m+1-j_2$, and therefore by the nondecreasing work assumption, $w_i(m+1-j_1) \geq w_i(m+1-j_2)$, contradicting (44). □

A minimum-weight matching in G can be found in $O(n^3)$ steps using the Hungarian method [Kuh55], which is due to Kuhn. Thus the total running time is $O(n^3+mn)$ under the nondecreasing work assumption, and $O(n^3+mn^2)$ otherwise. This completes the proof of Lemma 13 from which Theorem 3 follows directly.

6 MACT with Nondecreasing Work

We now consider the case of malleable scheduling where each task $T_i \in \mathcal{M}$ satisfies $w_i(j) \leq w_i(j+1)$ for all $j \in [m-1]$.

Given an instance of MACT, let

$$\alpha(\bar{p}) = A(\bar{p}) + \frac{1}{2}H(\bar{p}) - \frac{1}{2}W(\bar{p}). \tag{45}$$

By Theorem 1, $\alpha(\bar{p})$ is a lower bound on C_{opt} for the $\bar{p}$-derived instance of NACT. Let

$$\alpha^M = \min_{\bar{p}}\{\alpha(\bar{p})\}. \tag{46}$$

Then α^M is a lower bound on C^M_{opt}. Let

$$\hat{\alpha}(\bar{p}) = A(\bar{p}) + \frac{1}{2}H(\bar{p}) - W(\bar{p}). \tag{47}$$

As in (16), $\hat{\alpha}(\bar{p})$ is a lower bound on C_{opt} for the $\bar{p}$-derived NACT instance.

Consider the following MACT algorithm. Use ALLOT to find an allotment $\bar{p}$ that satisfies $\alpha(\bar{p}) = \alpha^M$. Define an allotment $\bar{p}^{(r)}$ by $p_i^{(r)} = \min\{p_i, r\}$. Let $r = \lceil \frac{m}{2} \rceil$, and schedule the $\bar{p}^{(r)}$-derived NACT instance using LWF. We call this MACT algorithm MLWF.

Lemma 18 *For any allotment $\bar{p}$, and any $r \geq \lceil \frac{m}{2} \rceil$,*

$$\hat{\alpha}(\bar{p}^{(r)}) \leq \alpha(\bar{p}). \tag{48}$$

Proof: Our goal is to show that $H(\bar{p}^{(r)}) \leq H(\bar{p}) + W(\bar{p})$, and that $A(\bar{p}^{(r)}) - W(\bar{p}^{(r)}) \leq A(\bar{p}) - W(\bar{p})$. Then the lemma follows by adding half the first inequality to the second inequality.

First, we will show that $A(\bar{p}^{(r)}) - W(\bar{p}^{(r)}) \leq A(\bar{p}) - W(\bar{p})$. Let σ be a permutation that is induced by $\bar{p}$. Then $A(\bar{p}^{(r)}) \leq A(\bar{p}^{(r)}, \sigma)$ by Lemma 11. Observe that $p_i^{(r)} \leq p_i$ for all $i \in [n]$, and therefore by the nondecreasing work assumption, $w_i(p_i^{(r)}) \leq w_i(p_i)$. Then

$$\begin{aligned}
A(\bar{p}^{(r)}) - W(\bar{p}^{(r)}) &\leq A(\bar{p}^{(r)}, \sigma) - W(\bar{p}^{(r)}) \\
&= \frac{1}{m}\sum_{i=1}^{n}(\sigma^{-1}(i) - 1)w_i(p_i^{(r)}) \\
&\leq \frac{1}{m}\sum_{i=1}^{n}(\sigma^{-1}(i) - 1)w_i(p_i) \\
&= A(\bar{p}) - W(\bar{p}).
\end{aligned}$$

Next, we show that $H(\bar{p}^{(r)}) \leq H(\bar{p}) + W(\bar{p})$. It is sufficient to show that for each task T_i, we have $t_i(p_i^{(r)}) \leq t_i(p_i) + p_i * t_i(p_i)/m$. If $p_i \leq r$, then $p_i^{(r)} = p_i$, and it is clear that $t_i(p_i^{(r)}) \leq t_i(p_i) + p_i * t_i(p_i)/m$. Suppose instead that $p_i > r$. Then $p_i^{(r)} = r$. Now it suffices to show that $t_i(r) \leq t_i(p_i) + p_i * t_i(p_i)/m$. By the nondecreasing work assumption, we have

$$r * t_i(r) \leq p_i * t_i(p_i). \tag{49}$$

Since $p_i \leq m$, we also have

$$r * t_i(r) \leq m * t_i(p_i). \tag{50}$$

Then adding together (49) and (50), we get

$$2r * t_i(r) \leq m * t_i(p_i) + p_i * t_i(p_i). \tag{51}$$

Dividing both sides by m yields

$$\frac{2r}{m} t_i(r) \leq t_i(p_i) + p_i * t_i(p_i)/m. \tag{52}$$

Then since $\frac{2r}{m} \geq 1$, we have

$$t_i(r) \leq t_i(p_i) + p_i * t_i(p_i)/m. \tag{53}$$

Therefore,

$$H(\bar{p}^{(r)}) \leq H(\bar{p}) + W(\bar{p}). \tag{54}$$

$\square$

Proof of Theorem 5: By Lemma 13, we can find an allotment $\bar{p}$ such that $\alpha(\bar{p}) = \alpha^M$ in $O(n^3 + mn)$ steps. Let $r = \lceil \frac{m}{2} \rceil$. Let $\rho = \max\{\frac{m}{m-r+1}, 2\} = 2$. Recall that C^M_{MLWF} is the value of C that results from scheduling the $\bar{p}^{(r)}$-derived instance using LWF. Then by Lemmas 9 and 18, we have

$$\begin{aligned} C^M_{\text{MLWF}} &\leq 2\hat{\alpha}(\bar{p}^{(r)}) \\ &\leq 2\alpha(\bar{p}) \\ &= 2\alpha^M \\ &\leq 2C^M_{opt}. \end{aligned}$$

$\square$

Consider the special case of MACT in which for each task T_i, a processor requirement p_i and an execution time t_i are given. But we have the option to use fewer than p_i processors, with a corresponding *linear* slowdown in execution time. That is, the execution time of task T_i is given by $t_i(j) = p_i t_i / j$ for any $j \in [p_i]$, and $t_i(j) = t_i$ for $j > p_i$. This is known as *virtualization* [FKST93]. We call the problem of scheduling virtualizable tasks to minimize average completion time VACT. VACT is a special case of MACT with nondecreasing work. In VACT, allotting p_i processors to each task minimizes the bound $\alpha(\bar{p})$. Then a schedule can be constructed by using LWF on the $\bar{p}^{(r)}$-derived NACT instance, as in MLWF. Call this algorithm VLWF.

Theorem 19 *For any instance of VACT,*

$$C^M_{\mathrm{VLWF}} \leq 2C^M_{opt}. \tag{55}$$

The running time of VLWF is $O(n \log n)$.

For our MACT algorithm under the nondecreasing work assumption, we have assumed that each task could be allotted any number of processors up to m. But it may be that the set S_i of possible allotments to task T_i is a proper subset of $[m]$. Then instead of choosing a maximum allowable number of processors to allot to each task $r = \lceil \frac{m}{2} \rceil$ that applies to all the tasks, we can choose an individual upper limit r_i for each task. We can select an allotment $\bar{p}$ to minimize the NACT lower bound as before, and then we need only determine r_i for tasks that have $p_i > \lceil \frac{m}{2} \rceil$. For these tasks, let $r_i = \arg\min_{j \in S_i} \max\{\frac{m}{m-j+1}, \frac{m}{j}\}$. (For tasks that have $p_i \leq \lceil \frac{m}{2} \rceil$, we can use $r_i = \lceil \frac{m}{2} \rceil$ as before.) Then we use the allotment $\bar{p}^{(r)}$, where $p_i^{(r)} = \min\{p_i, r_i\}$. Let $\rho = \max_{i \in [n]} \max\{\frac{m}{m-r_i+1}, \frac{m}{r_i}\}$. Then the resulting MACT algorithm is ρ-approximate.

Note that ρ depends on the sets S_i. For example, if $S_i = \{1, m\}$ for all $i \in [n]$, then it may not be possible to guarantee $\rho < m$ using this method.

7 Conclusions and Open Problems

A natural extension of the problems we have considered is scheduling to minimize *weighted* average completion time. That is, along with each task T_i is associated a weight u_i. The new objective is to minimize $\sum_{i=1}^{n} u_i c_i$. We will use NWACT and MWACT to refer to the weighted versions of NACT and MACT. The results of Theorems 1, 2, 3, and 5 can all be extended to weighted tasks. For the NWACT lower bound and scheduling algorithm, the weighted tasks should be ordered by the ratio of work to weight $p_i t_i / u_i$. Note that Corollary 4 does not extend to weighted tasks, because no algorithm for NWACT is known to have a constant approximation factor. The current state of affairs is summarized in Table 1.

Any nonmalleable problem can be formulated as a malleable problem, and therefore there cannot be an improvement in the best known approximation factor for a malleable problem (with no assumptions on task execution times) without a corresponding improvement in the nonmalleable problem. In the other direction, improved approximation factors for nonmalleable problems carry over to their malleable counterparts provided that the lower bound used to prove the approximation factor of the nonmalleable algorithm is of a

	makespan		ACT		WACT
nonmalleable	2	[GG75]	32	[TSWY94]	?
malleable	2	[TWY92, LT94]	32*		?
malleable, nondec. work	2	[BB90]	2*		2*

*this paper

Table 1: The best known approximation factors for scheduling parallel tasks

certain form. This follows from Theorem 3 in the case of minimizing average completion time, and from previous results [TWY92, LT94] in the case of minimizing makespan. Thus NACT and NWACT appear to be the best places to look for improved results.

As mentioned in Section 1, this work was motivated by the Condor distributed batch system. Condor is designed to make the best use of the computing power available in a workstation environment. However, under Condor [LLM88, LL90], machines must be relinquished when their owners return to use them. Therefore, schedulers must be able to handle unexpected variations in the number of available processors.

In the problems that we have considered, it is assumed that we have complete information on all the tasks. But in a scheduling environment such as Condor, not all of this information is available. Therefore, it would be useful to investigate what results are possible with less information. For example, suppose that for each task and for each possible number of processors, we do not know the execution time, but we know the *speedup* relative to the single-processor execution time of that task. Or suppose that we have an *estimate* of the execution time, and its accuracy is governed by some known distribution. These variations can be applied to the problem of minimizing makespan as well as the problem of minimizing average completion time.

Faster algorithms for MACT are desirable. The $O(n^3)$ term in the runtimes of our schedulers represents the cost of finding a minimum weight perfect bipartite matching. A faster algorithm for finding a perfect matching whose weight is within a constant factor of the optimal can be used to make the scheduler faster at the expense of increasing its approximation factor.

References

[AKM$^+$87] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:209–233, 1987.

[BB90] K. Belkhale and P. Banerjee. Approximate algorithms for the partitionable independent task scheduling problem. In *Proceedings of the 1990 International Conference on Parallel Processing, vol. I*, pages 72–75, 1990.

[BLK83] J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

[CYW92] M. Chen, P. Yu, and K. Wu. Scheduling and processor allocation for the execution of multi-join queries in a multiprocessor system. In *Proceedings of the 8th International Conference on Data Engineering*, pages 58–67, 1992.

[DL89] J. Du and J. Leung. Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*, 2(4):473–487, 1989.

[EEI64] W. L. Eastman, S. Even, and I. M. Isaacs. Bounds for the optimal scheduling of n jobs on m processors. *Management Science*, 11(2):268–279, November 1964.

[FKST93] A. Feldmann, M.-Y. Kao, J. Sgall, and S.-H. Teng. Optimal online scheduling of parallel jobs with dependencies. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 642–651, 1993.

[GG75] M. Garey and R. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2):187–200, June 1975.

[Kuh55] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[LL90] M. J. Litzkow and M. Livny. Experience with the Condor distributed batch system. In *Proceedings of the IEEE Workshop on Experimental Distributed Systems*, 1990.

[LLKS92] E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys. Sequencing and scheduling: Algorithms and complexity. In *Handbook of Operations Research and Management Science, Volume IV: Production Planning and Inventory.* North-Holland, 1992.

[LLM88] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor—a hunter of idle workstations. In *8th International Conference on Distributed Computing Systems*, pages 104–111, 1988.

[LT94] W. Ludwig and P. Tiwari. Scheduling malleable and nonmalleable parallel tasks. To appear in *Proceedings of the 5th SIAM Symposium on Discrete Algorithms*, 1994.

[TSWY94] J. Turek, U. Schwiegelshohn, J. L. Wolf, and P. S. Yu. Scheduling parallel tasks to minimize average response time. To appear in *Proceedings of the 5th SIAM Symposium on Discrete Algorithms*, 1994.

[TWY92] J. Turek, J. Wolf, and P. Yu. Approximate algorithms for scheduling parallelizable tasks. In *4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 323–332, 1992.