Algorithms for Scheduling Malleable and Nonmalleable Parallel Tasks

Walter T. Ludwig

Technical Report #1279

August 1995

ALGORITHMS FOR SCHEDULING MALLEABLE AND NONMALLEABLE PARALLEL TASKS

By

Walter T. Ludwig

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN – MADISON

1995

Abstract

A *malleable* parallel task is one whose execution time is 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 problems of constructing a nonpreemptive schedule of these tasks to minimize one of two objectives: makespan or weighted average completion time.

We present a general two-step method for scheduling malleable tasks that applies to both objectives: first, choose an allotment of processors to tasks; second, using the chosen allotment, schedule the tasks as though nonmalleable. In the case of minimizing makespan, we provide an algorithm for selecting an allotment that, when coupled with existing algorithms for scheduling nonmalleable tasks, yields algorithms for scheduling malleable tasks that have the same approximation factor. As a result, we obtain a 2-approximate algorithm for scheduling malleable tasks to minimize makespan with running time $O(mn)$. Also, in the case when the processors are arranged in a line, and the processors allotted to a task must be contiguous, we obtain a 2-approximate algorithm with running time $O(mn + n \log^2 n / \log \log n)$.

For the weighted average completion time objective, we prove a lower bound on this objective for nonmalleable task sets, and use it to prove approximation factors

for three new algorithms for scheduling nonmalleable tasks: an 8-approximate algorithm for unweighted tasks, a 10.43-approximate algorithm for weighted tasks, and a $\max\{\frac{1}{1-\beta}, \frac{1}{2(1-\beta)} + 1\}$-approximate algorithm for weighted tasks that require no more than $\lceil \beta m \rceil$ processors each. We also provide three allotment selection algorithms for this objective that exhibit a tradeoff between running time and allotment quality.

Putting these results together yields several algorithms for scheduling malleable tasks to minimize average completion time, including a 2-approximate algorithm for unweighted tasks with running time $O(n^3 + mn)$ that requires a weak condition on the task execution times, and a 4-approximate algorithm for weighted tasks with running time $O(n^2 + mn)$ that requires the same condition.

Acknowledgments

I am grateful to Prasoon Tiwari, who guided me through the initial steps toward completing this thesis. He discussed a number of research areas with me and helped me to define my area of interest. He provided the ideas for research problems that led to this work. He encouraged me to complete the work when my enthusiasm had worn thin. He also instructed me in numerous other matters, teaching me everything so that I might be successful.

Thanks to Eric Bach, my "godparent," who has taken over as my advisor following Prasoon's departure. He took care to insure that I continued to make progress until this thesis was completed.

I thank Anne Condon and Scott Webster for carefully reading this thesis and providing feedback. Anne also read the first draft of my prelim paper, and promptly responded with detailed comments. Her support has been extremely valuable to me.

Thanks to Mary Vernon and Debby Joseph for being on my committee. Mary advised me in using simulation to compare processor allocation policies. I also thank Su-Hui Chiang for providing me with simulation code, and for patiently answering all my questions.

It is my privilege to have worked with Joel Wolf, and also Uwe Schwiegelshohn, Philip Yu, and John Turek.

I thank Susan Hert for patiently enduring three years in the same office with me.

Finally, I thank my father and my mother, for being my father and my mother.

Contents

Chapter 1

Introduction

Parallel processing systems of various types are now widely available and are attracting large numbers of users, thus creating a need for some method of distributing the processing power of a parallel system among competing jobs. It is plain that executing a program in parallel introduces overhead that is not present in its sequential execution. Moreover, communication and synchronization overhead tend to increase with the number of processors that are allocated to a program. Thus, allocating all the system resources to one job at a time is not likely to be the most efficient way to use a parallel system. On the other hand, users want their jobs to be allocated as much processing power as possible so that they will be completed as quickly as possible.

We consider the problem of allocating system resources to jobs using a model in which there are a collection of independent jobs or tasks to be scheduled on a set of identical processors, and each task's execution time is a known function of the number of processors that are allocated to it. We call these tasks *malleable*. Although we frame the discussion of resource allocation in terms of allocating processors to parallel tasks, the model that we are using applies whenever there is a single resource that can

be allocated to independent tasks in arbitrary amounts. Another example of such a resource is memory [59]. Suppose we have a parallel system with virtual memory and a common primary memory in which each task is executed on a single processor. Then we can treat the execution time of a task as a function of the number of pages of main memory it is allocated.

Once it has been decided how much of the resource to allocate to each task, we can view the tasks as if they were *nonmalleable*, i.e., as if there were no choice, but rather that each task requires some fixed amount of the resource. Then it remains to construct a schedule that minimizes some objective function, which may be either *makespan* (total schedule length), or average completion time over all the tasks.

Nonmalleable task scheduling problems may also arise independently of malleable task scheduling. That is, in some cases, the tasks to be scheduled may be nonmalleable to begin with. This is the case with parallel systems where the user specifies for each job the number of processors on which it is to be executed. Another example of nonmalleable tasks is found in scheduling burn-in operations in semiconductor manufacturing [58]. In this problem, each task consists of some number of circuit boards that are to be heated in an oven for some length of time. Here the resource is the capacity of the oven in terms of the number of boards it can hold.

1.1 The Scheduling Problems

We are given a set T of n independent tasks. Each task i has an execution time function $t_i : M_i \to \mathbb{R}$, where $M_i \subseteq [m]$ is the set of possible allocations of processors to task i, and we use $[m]$ to denote the set $\{1, 2, \ldots, m\}$. If task i is allocated j processors and scheduled to begin execution at time b_i, then it is completed at time $b_i + t_i(j)$:

preemption is not allowed, neither may the number of processors allocated to a task be changed during its execution.

We are also given m identical processors, and our goal is to construct a schedule of the n tasks on the m processors that minimizes some objective. We consider two objectives: makespan and weighted average completion time. We will refer to these problems as MM (malleable makespan) and MWACT (malleable weighted average completion time). In the case of MWACT, a weight w_i is associated with each task. We refer to the *unweighted* case of MWACT, i.e., the case where $w_i = 1$ for all tasks i, as MACT.

If $|M_i| = 1$ for all tasks i, then the task set is nonmalleable. We refer to the nonmalleable versions of MM, MWACT, and MACT as NMM, NMWACT, and NMACT, respectively.

Regardless of objective, scheduling malleable tasks is a generalization of scheduling nonmalleable tasks, which in turn is a generalization of scheduling sequential tasks (i.e., tasks that require one processor). When the objective is minimizing makespan, even scheduling sequential tasks is NP-hard in the strong sense [24], and therefore NMM and MM are also NP-hard in the strong sense. When the objective is minimizing average completion time, while unweighted sequential tasks can be scheduled optimally in $O(n \log n)$ steps [16], scheduling weighted sequential tasks is NP-hard in the strong sense [24]. Furthermore, NMACT is NP-hard in the strong sense even when the tasks have unit execution times [10]. Therefore NMWACT, MACT, and MWACT are also NP-hard in the strong sense.

For malleable scheduling problems, it is often helpful to assume that the task execution time functions satisfy certain reasonable conditions. One such condition is that

execution time does not increase as more processors are allocated:

$$\text{for all } j,k \in M_i\colon\ j < k \ \Rightarrow\ t_i(j) \geq t_i(k). \tag{1}$$

This is called the *nonincreasing execution time* condition. In practice, the execution time of a task typically decreases with the number of processors allocated to it up to some limit, beyond which the execution time increases. This is a result of an inherent limit on the amount of parallelism in a given task, coupled with communication and synchronization overhead that increases as the number of processors increases. Such behavior does not present a serious problem for algorithms that require the nonincreasing execution time condition, as any algorithm can easily be modified to handle tasks that exhibit this behavior.

Another condition is that *work* — the product of processors and execution time — does not decrease as more processors are allocated:

$$\text{for all } j,k \in M_i\colon\ j < k \ \Rightarrow\ j \cdot t_i(j) \leq k \cdot t_i(k). \tag{2}$$

This is called the *nondecreasing work* condition. One way of viewing this condition is that the tasks do not exhibit superlinear speedup. Another is that task efficiency does not increase with the number of processors. While this condition is quite reasonable, since overhead associated with executing a task in parallel tends to increase with the number of processors allocated, there are some circumstances under which the condition will not hold. For example, it may be that a task divides naturally into four subtasks of equal size. In this case, we may see work decreasing as we increase the allocation from three to four processors. Another situation in which the condition may be violated arises from the impact of memory in distributed parallel processing systems [44]. In such systems, the amount of local memory allocated to a task is limited by the number

of processors allocated to it, and allocating insufficient local memory may result in excessive paging overhead.

Malleable tasks under these two conditions, along with the condition that $M_i = [m]$ for all tasks i, are no longer a generalization of nonmalleable tasks. However, they are still a generalization of sequential tasks, and therefore the NP-hardness results for MM and MWACT apply even under these conditions. As for MACT, Shachnai and Glasgow [50] have shown that a restricted version of MACT that satisfies these conditions is NP-hard in the strong sense.

Therefore, since the problems that we are studying are all NP-hard, we will consider approximation algorithms, and evaluate these algorithms in terms of *approximation factor* and running time. We say that an algorithm for a minimization problem has approximation factor ρ, or is ρ-approximate, if for any given instance it produces a solution with objective function value not more then ρ times that of the optimal solution.

1.2 New Results

Our approach to scheduling malleable tasks is to divide the problem into two parts: first choose an allotment of processors to tasks, and then schedule the tasks as though nonmalleable. To see how this method works, let $\bar{p} = (p_1, p_2, \ldots, p_n)$ denote an allotment of processors to tasks, where p_i is the number of processors allocated to task i. Observe that a set of malleable tasks T together with an allotment $\bar{p}$ yields a set of nonmalleable tasks $T(\bar{p})$, in which task i requires p_i processors and has execution time $t_i(p_i)$.

The goal in the first step is to choose an allotment $\bar{p}$ such that the value of a given

lower bound on the objective function is small for $T(\bar{p})$. Let $\Lambda_T(\bar{p})$ be a lower bound on the objective function for the nonmalleable task set $T(\bar{p})$. Then $\Lambda_T = \min_{\bar{p}}\{\Lambda_T(\bar{p})\}$ is a lower bound on the objective function for the malleable task set T. So the first step in our approach to scheduling malleable tasks is to choose an allotment $\bar{p}$ such that $\Lambda_T(\bar{p}) = \Lambda_T$.

The second step is to apply an algorithm for scheduling nonmalleable tasks to the task set $T(\bar{p})$. If we have an algorithm that is ρ-approximate — that is, for any nonmalleable task set $T(\bar{p})$ it produces a schedule with objective function value not more than $\rho\Lambda_T(\bar{p})$ — then this algorithm coupled with an algorithm for the first step yields a ρ-approximate algorithm for scheduling malleable tasks. For any malleable task set T, it will choose an allotment $\bar{p}$ and then construct a schedule for $T(\bar{p})$ with objective function value not exceeding $\rho\Lambda_T(\bar{p}) = \rho\Lambda_T$.

This two-step method can be applied to both the average completion time and the makespan objectives. In the case of average completion time, we present three algorithms for scheduling nonmalleable tasks, and prove approximation factors for all three using a new lower bound. This lower bound is a generalization of the bound for sequential tasks given by Eastman, Even, and Isaacs [19]. We also present three algorithms for choosing an allotment to minimize the lower bound. In particular, we present an $O(n^3 + mn)$ algorithm that selects an optimal allotment — that is, an allotment $\bar{p}$ such that $\Lambda_T(\bar{p}) = \Lambda_T$ — for a set T of unweighted tasks. We also obtain faster algorithms by settling for approximations. We present a 2-approximate allotment selection algorithm with running time $O(n^2 + mn)$ that applies to weighted tasks, and a 4-approximate algorithm with running time $O(\min\{n + m\log m, n^2\})$ that requires $w_i = 1$ and $M_i = [m]$ for all tasks i, along with the nondecreasing work and

nonincreasing execution time conditions.

As for scheduling nonmalleable tasks, we present an 8-approximate NMACT algorithm, a 10.43-approximate NMWACT algorithm, and an NMWACT algorithm with approximation factor $\max\{\frac{1}{1-\beta}, \frac{1}{2(1-\beta)} + 1\}$ when no task requires more than $\lceil \beta m \rceil$ processors. All three algorithms have running time $O(n \log n)$.

In order make use of the last of the three NMWACT algorithms in an MWACT algorithm, it is useful to insert an additional step into the MWACT algorithm as follows. After choosing an allotment $\bar{p}$, adjust it to an allotment $\bar{q}$ by taking $q_i = \min\{p_i, \lceil \alpha m \rceil\}$ for some $\alpha \in (0, 1]$, thus reducing to $\lceil \alpha m \rceil$ the allocation of any task that exceeds $\lceil \alpha m \rceil$. This adjustment will reduce the "quality" of $\bar{p}$ somewhat, depending on α, but to insure that this reduction in quality will not be too great, we require that the tasks satisfy the *α-weak nondecreasing work* condition:

$$\text{for all } k \in M_i\colon\ \lceil \alpha m \rceil < k \ \Rightarrow\ \lceil \alpha m \rceil \cdot t_i(\lceil \alpha m \rceil) \leq k \cdot t_i(k). \tag{3}$$

For example, for unweighted task sets T that satisfy the $\frac{1}{2}$-weak nondecreasing work condition, we have a 2-approximate algorithm: choose an optimal allotment $\bar{p}$ for T, adjust it by taking $q_i = \min\{p_i, \lceil m/2 \rceil\}$, and then schedule the task set $T(\bar{q})$ using the $\max\{\frac{1}{1-\beta}, \frac{1}{2(1-\beta)} + 1\}$-approximate NMWACT algorithm. Note that the α-weak nondecreasing work condition is substantially less restrictive than the nondecreasing work condition, and in particular $\frac{1}{2}$-weak nondecreasing work is quite realistic: it seems unlikely that any gain in efficiency could be obtained by increasing a task's processor allocation beyond $m/2$.

The MACT and MWACT algorithms that result from coupling our allotment selection algorithms with our NMACT and NMWACT algorithms, inserting an allotment adjustment step in some cases, are listed in Table 1. To our knowledge, these are the

approximation factor	running time	nondecr. work?	requires $w_i = 1$?	reference
$\max\{\frac{1}{1-\alpha}, \frac{1}{\alpha}\}$	$O(n^3 + mn)$	α-weak	yes	Cor. 3.2.2
2		$\frac{1}{2}$-weak		
$2 \cdot \max\{\frac{1}{1-\alpha}, \frac{1}{\alpha}\}$	$O(n^2 + mn)$	α-weak	no	Cor. 3.3.2
4		$\frac{1}{2}$-weak		
7.124	$O(\min\{n \log n + m \log m, n^2\})$	yes*	yes	Cor. 3.4.2
8	$O(n^3 + mn)$	no	yes	Cor. 4.3.1
16	$O(n^2 + mn)$	no	yes	Cor. 4.3.1
20.86	$O(n^2 + mn)$	no	no	Cor. 4.2.1

*also requires nonincreasing execution time and $M_i = [m]$

Table 1: MACT and MWACT algorithms

first polynomial-time algorithms for MACT and MWACT with constant approximation factors.

For the makespan objective, we benefit from the fact that many NMM algorithms already exist, and that there is a single lower bound that is used to prove the approximation factor for most of these algorithms. Much of the work in this area is devoted to the problem of scheduling on a *line* of processors: the m processors are arranged in a line, and the processors allocated to each task must be contiguous. By providing an allotment selection algorithm, we extend existing NMM algorithms to MM algorithms, and NMM algorithms for scheduling on a line to MM algorithms for the same.

We distinguish between *monotonic* MM (MMM), in which the tasks are known to satisfy both the nondecreasing work and nonincreasing execution time conditions, and *general* MM (GMM), in which there are no assumptions on the task execution time functions. We present an allotment selection algorithm for MMM with running time $O(n \log^2 m)$. This algorithm can be extended to GMM at a cost of increasing the running time to $O(mn)$. Combining these algorithms with existing NMM algorithms yields

running time	approx. factor	line?	GMM or MMM?	NMM reference
$O(mn)$	2	no	GMM	[22]
$O(n \log^2 m)$	2	no	MMM	
$O(mn + n \log^2 n / \log \log n)$	2	yes	GMM	[53]
$O(n \log^2 m + n \log^2 n / \log \log n)$	2	yes	MMM	
$O(mn + n \log n)$	2.5	yes	GMM	[51]
$O(n \log^2 m + n \log n)$	2.5	yes	MMM	

Table 2: MM algorithms

the results shown in Table 2. To our knowledge, 2 is the best known approximation factor of a polynomial-time algorithm for GMM, and these are the fastest known running times in which that factor can be achieved. In fact, $O(mn)$ is within a constant factor of the best possible running time of any approximation algorithm for GMM with a constant approximation factor. This is because it is necessary to inspect all m values of all n execution time functions in order to guarantee a constant approximation factor.

We also present methods for both GMM and MMM for computing an optimal allotment efficiently in parallel.

1.3 Related Work

1.3.1 Minimizing Makespan

The predecessor of parallel task scheduling problems is the *resource-constrained* scheduling problem [6], as posed by Garey and Graham [22]. In the resource-constrained scheduling problem, there are one or more resources, and each task requires a certain amount of each resource for the duration of its execution time. NMM can be formulated as a resource-constrained scheduling problem, with available processors as the single

resource. Garey and Graham [22] proposed a simple list scheduling approach to this problem. In particular, let the tasks be arranged in some arbitrary order. Whenever processors are free, schedule the first task in the list whose processor requirement does not exceed the number of available processors. Garey and Graham show that this algorithm has an approximation factor of 2 for NMM.

Another closely related problem is the oriented orthogonal rectangle packing problem, first posed by Baker, Coffman, and Rivest [3], and also studied by many others [14, 51, 2, 4, 53]. This problem is in fact identical to NMM on a line of processors: one of the dimensions in the rectangle-packing problem corresponds to time in NMM, and the other corresponds to processors. Sleator [51] gives a 2.5-approximate rectangle-packing algorithm with running time $O(n \log n)$. This was the best known approximation factor for this problem until recently, when Steinberg [53] gave a 2-approximate algorithm with only a slightly greater running time of $O(n \log^2 n / \log \log n)$.

The extension of the rectangle-packing problem to packing in three dimensions amounts to NMM on a two-dimensional mesh of processors. Li and Cheng [36] give a 46/7-approximate algorithm for this problem with running time $O(n \log n)$.

Krishnamurti and Ma [30] first posed MM (the MMM variant), but in a restricted form. In the problem they consider, every task is required to begin execution at time 0. (This requires $m \geq n$.) This restriction reduces the problem to that of deciding how many processors to allocate to each task, subject to the constraint that the total number of processors allocated to the tasks does not exceed the number of available processors.

Belkhale and Banerjee [5] consider a restriction of MMM in which $M_i = [m]$ for all tasks i. For this special case, they give a 2-approximate algorithm with running time $O(n \log m + m \log m)$ that applies to lines.

Krishnamurti and Narahari [31] consider a less restricted version of MMM in which $M_i \supseteq \{1, 2, 4, \ldots, 2^{r_i}\}$ for all tasks i. They obtain a 2-approximate algorithm with running time $O(n + m \log m)$, but it requires the use of preemption. In a later improvement [42], they give a nonpreemptive algorithm for the same problem with the same approximation factor and running time $O(n \log n + m \log m)$. Both algorithms apply to lines.

The precursor to our two-step method was introduced by Turek, Wolf, and Yu [57], who showed that an algorithm for NMM with approximation factor ρ and running time $Q(m, n)$ can be extended to an algorithm for GMM with the same approximation factor and running time $O(mn \cdot Q(m, n))$. Their method is to generate a set of allotments, one of which is the allotment $\bar{p}$ of an optimal schedule for T (although it is not known which allotment). Then an algorithm for NMM is used in conjunction with each of the generated allotments, and the best of the resulting schedules is selected. Now the approximation factor of this GMM algorithm is the same as that of the NMM algorithm, because an optimal schedule for $T(\bar{p})$ is also an optimal schedule for T, and the NMM algorithm is guaranteed to construct a ρ-approximate schedule for $T(\bar{p})$. The running time follows from the fact that the set of allotments generated in the first step contains no more than mn elements. Applying the NMM results of Garey and Graham [22], this technique yields a 2-approximate algorithm for GMM with running time $O(n^2 m \log m)$. For NMM on a line, the rectangle-packing results of Steinberg [53] and Sleator [51] yield a 2-approximate algorithm with running time $O(mn^2 \log^2 n / \log \log n)$, and a 2.5-approximate algorithm with running time $O(mn^2 \log n)$, respectively. Our observation that it is sufficient to generate a single allotment, rather than a set of up to mn allotments, results in a substantial reduction in running time.

Rather than seeking approximation algorithms, another way to approach NP-hard problems is to try to identify the boundaries between cases that can be solved in polynomial time and cases that are NP-hard. This is the approach to MM that is taken by Du and Leung [18]. The same approach can also be applied to NMM [7, 10, 23].

Note that MM can be viewed as resource-constrained scheduling with one discrete malleable resource. For the variation in which the resource is continuous rather than discrete, see [8, 6].

In another variation, Feldmann, Kao, Sgall, and Teng [20] consider on-line MM with precedence constraints and linear speedup.

1.3.2 Minimizing Weighted Average Completion Time

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 [34] for a survey.) However, approximation algorithms for the natural extension from sequential tasks to parallel tasks are scarce. Results for resource-constrained scheduling with this objective are directed toward identifying special cases that can be solved in polynomial time [6, 9], rather than providing approximation algorithms for the general case with a single resource. It was only recently that Turek, Schwiegelshohn, Wolf, and Yu [56] demonstrated an NMACT algorithm with running time $O(n \log n)$ and approximation factor 32, the first proof of a constant approximation factor for a polynomial-time NMACT algorithm. Our 8-approximate NMACT algorithm and 10.43-approximate NMWACT algorithm improve this result.

While our primary focus is on a scheduling environment in which all tasks are ready at time 0, and complete information about task execution times is available, our fastest

allotment selection algorithm can also be adapted to an environment in which tasks arrive over time and little or no information about task execution times is available. Such an environment may reflect more accurately the conditions that are encountered in practice. Thus it is not surprising that many processor allocation policies for this environment have been studied, typically using simulations or analytical modeling to evaluate their relative performance. (For example, see [12, 41, 40, 48, 25, 35, 60, 54].) We introduce a new processor allocation policy based on our fastest allotment selection algorithm, and present simulation results suggesting that its performance is comparable to that of the best known policies.

1.4 Organization of This Thesis

Chapter 2 contains all the elements of an MWACT algorithm, except for allotment selection. We present a lower bound on weighted average completion time, and use it to prove an approximation factor for a simple NMWACT algorithm. We then show how inserting an allotment adjustment step into an MWACT algorithm affects its approximation factor. Chapter 3 describes our allotment selection algorithms, and also includes simulation results for an adaptation of one of the algorithms. Chapter 4 contains our 10.43-approximate NMWACT algorithm and our 8-approximate NMACT algorithm. In Chapter 5, we present both sequential and parallel allotment selection algorithms for MM. Chapter 6 contains conclusions and directions for future work.

Chapter 2

MWACT Algorithm Foundations

Note: The results reported in this chapter and in Sections 3.1 and 3.2 of the subsequent chapter were obtained jointly with Tiwari, and similar results were obtained independently by Turek, Wolf, Fleischer, Glasgow, Schwiegelshohn, and Yu, and appear in [55].

2.1 NMWACT Lower Bounds

A lower bound on weighted average completion time for nonmalleable task sets provides the link between the two steps of our approach to MWACT. Not only does it serve to show approximation factors for NMWACT algorithms, it also provides an objective function for the allotment selection problem.

We will make use of lower bounds on *total* weighted completion time. Note that minimizing total weighted completion time is equivalent to minimizing weighted average completion time. For a given schedule $\mathcal{X}$ for the nonmalleable task set $T(\bar{p})$, let $b_i^{\mathcal{X}}$ denote the starting time of task i, and let $R^{\mathcal{X}} = \sum_{i=1}^{n} w_i[b_i^{\mathcal{X}} + t_i(p_i)]$ denote the total weighted completion time of $\mathcal{X}$. The objective is to construct a schedule that minimizes

$R^{\mathcal{X}}$. Let $R_{\mathcal{A}}(T(\bar{p}))$ be the value of $R^{\mathcal{X}}$ that results from applying algorithm $\mathcal{A}$ to the nonmalleable task set $T(\bar{p})$. Let $R^*(T(\bar{p}))$ denote the optimal value of $R^{\mathcal{X}}$ over all possible schedules $\mathcal{X}$ for the task set $T(\bar{p})$.

Let

$$H_T(\bar{p}) = \sum_{i=1}^{n} w_i t_i(p_i) \tag{4}$$

be the weighted sum of the task execution times. This is a lower bound on total weighted completion time for $T(\bar{p})$.

The next lower bound is based on an idealized schedule of "squashed" tasks. Each task has a squashed counterpart with the same work requirement and the same weight, but that requires all m processors. (So its execution time is $p_i \cdot t_i(p_i)/m$.) An optimal schedule of the squashed tasks is obtained using Smith's rule [52]: schedule the tasks in order of nondecreasing ratio of work to weight. This gives a lower bound on total weighted completion time for the original task set.

In order to give an expression for this lower bound, we introduce the following notation. Let $\bar{w} = (w_1, w_2, \ldots, w_n)$ be the vector of task weights. Let $(\bar{w})_i^{\sigma} = \sum_{k:\sigma(k)\leq\sigma(i)} w_k = \sum_{s=1}^{\sigma(i)} w_{\sigma^{-1}(s)}$ be the sum of the weights of the tasks that precede task i under the order σ, plus the weight w_i of task i itself. Let $\eta_i = p_i \cdot t_i(p_i)/w_i$ be the ratio of work to weight of task i. Let σ be a permutation that orders the tasks by nonincreasing ratio of work to weight. That is, $\eta_{\sigma^{-1}(s)} \geq \eta_{\sigma^{-1}(s+1)}$ for all $s \in [n-1]$. (Note that this is the reverse of the order in which the squashed tasks are scheduled. That is, task $\sigma^{-1}(n)$ comes first in the schedule.)

Observe that the completion time of task i in the schedule of the squashed tasks is $\sum_{s=\sigma(i)}^{n} p_{\sigma^{-1}(s)} t_{\sigma^{-1}(s)}(p_{\sigma^{-1}(s)})/m$. Then the total weighted completion time of this

schedule is

$$A_T(\bar{p}) = \frac{1}{m}\sum_{i=1}^{n} w_i \sum_{s=\sigma(i)}^{n} p_{\sigma^{-1}(s)} t_{\sigma^{-1}(s)}(p_{\sigma^{-1}(s)}). \tag{5}$$

Now if we let $k = \sigma^{-1}(s)$ and $r = \sigma(i)$ and change the order of summation, then we get

$$\begin{aligned} A_T(\bar{p}) &= \frac{1}{m}\sum_{k=1}^{n} p_k t_k(p_k) \sum_{r=1}^{\sigma(k)} w_{\sigma^{-1}(r)} &(6)\\ &= \frac{1}{m}\sum_{k=1}^{n} (\bar{w})_k^{\sigma} p_k t_k(p_k). &(7)\end{aligned}$$

This is the "squashed area" lower bound of [56], extended to weighted tasks.

Let

$$U_T(\bar{p}) = \frac{1}{m}\sum_{i=1}^{n} w_i p_i t_i(p_i) \tag{8}$$

be the ratio of the weighted sum of the task work requirements to the total number of processors. Note that $U_T(\bar{p}) \leq A_T(\bar{p})$ and $U_T(\bar{p}) \leq H_T(\bar{p})$.

The main result of this section is the following theorem, which gives a lower bound on total weighted completion time for NMWACT. This in turn yields an approximation factor for any NMWACT algorithm for which total weighted completion time can be bounded in terms of $A_T(\bar{p})$, $H_T(\bar{p})$, and $U_T(\bar{p})$. It also leads to a lower bound on total weighted completion time for MWACT.

Theorem 2.1.1 *For any nonmalleable task set $T(\bar{p})$, the total weighted completion time $R^{\mathcal{X}}$ of any schedule $\mathcal{X}$ for $T(\bar{p})$ is at least*

$$R^{\mathcal{X}} \geq A_T(\bar{p}) + \frac{1}{2}H_T(\bar{p}) - \frac{1}{2}U_T(\bar{p}). \tag{9}$$

Proof: Given a schedule $\mathcal{X}$ for the nonmalleable task set $T(\bar{p})$, let $r_i^{\mathcal{X}}(t)$ denote the fraction of task i that is not yet completed at time t. That is, let

$$r_i^{\mathcal{X}}(t) = \begin{cases} 1 & \text{if } t \leq b_i^{\mathcal{X}} \\ 1 - \frac{t - b_i^{\mathcal{X}}}{t_i(p_i)} & \text{if } b_i^{\mathcal{X}} < t \leq b_i^{\mathcal{X}} + t_i(p_i) \\ 0 & \text{if } b_i^{\mathcal{X}} + t_i(p_i) < t. \end{cases} \tag{10}$$

Now let

$$r^{\mathcal{X}}(t) = \sum_{i=1}^{n} w_i r_i^{\mathcal{X}}(t) \tag{11}$$

be the weighted number of tasks remaining at time t, including fractional tasks. Then

$$\begin{aligned}
\int_0^\infty r^{\mathcal{X}}(t)dt &= \sum_{i=1}^{n} w_i \int_0^\infty r_i^{\mathcal{X}}(t)dt \\
&= \sum_{i=1}^{n} w_i \left[\int_0^{b_i^{\mathcal{X}}+t_i(p_i)} dt - \int_{b_i^{\mathcal{X}}}^{b_i^{\mathcal{X}}+t_i(p_i)} \frac{t - b_i^{\mathcal{X}}}{t_i(p_i)} dt \right] \\
&= \sum_{i=1}^{n} w_i \left[(b_i^{\mathcal{X}} + t_i(p_i)) - \frac{t_i(p_i)}{2} \right] \\
&= R^{\mathcal{X}} - \frac{1}{2} H_T(\bar{p}).
\end{aligned} \tag{12}$$

Now we seek a lower bound on $r^{\mathcal{X}}(t)$. Define a squashed task set $T(\bar{q})$ as before. That is, let $\bar{q} = (m, m, \ldots, m)$, and for each task i, let $t_i(q_i) = t_i(m) = p_i \cdot t_i(p_i)/m$. Then each squashed task has the same work requirement (and weight) as its counterpart in the original task set. Consider the schedule $\mathcal{Y}$ in which the tasks in $T(\bar{q})$ are ordered by nondecreasing ratio of work to weight. Note that this is the schedule that we used in the definition of $A_T(\bar{p})$. This schedule provides the desired lower bound, according to the following lemma.

Lemma 2.1.1 *For any schedule $\mathcal{X}$, we have $r^{\mathcal{X}}(t) \geq r^{\mathcal{Y}}(t)$ for all t.*

Let us delay the proof of Lemma 2.1.1 until the completion of the proof of Theorem 2.1.1.

Observe that the weighted sum of the task completion times for the schedule $\mathcal{Y}$ is given by

$$R^{\mathcal{Y}} = A_T(\bar{p}), \tag{13}$$

and that

$$H_T(\bar{q}) = U_T(\bar{p}). \tag{14}$$

Also, by applying (12) to the schedule $\mathcal{Y}$, we get

$$\int_0^\infty r^{\mathcal{Y}}(t)dt = R^{\mathcal{Y}} - \frac{1}{2}H_T(\bar{q}). \tag{15}$$

Therefore, from (12), (15), (13), and (14), and from Lemma 2.1.1, it follows that

$$\begin{aligned} R^{\mathcal{X}} - \frac{1}{2}H_T(\bar{p}) &= \int_0^\infty r^{\mathcal{X}}(t)dt \\ &\geq \int_0^\infty r^{\mathcal{Y}}(t)dt \\ &= R^{\mathcal{Y}} - \frac{1}{2}H_T(\bar{q}) \\ &= A_T(\bar{p}) - \frac{1}{2}U_T(\bar{p}). \end{aligned} \tag{16}$$

We conclude that $R^{\mathcal{X}} \geq A_T(\bar{p}) + \frac{1}{2}H_T(\bar{p}) - \frac{1}{2}U_T(\bar{p})$. □

Proof of Lemma 2.1.1: To prove the lemma, we will use linear programming to derive a lower bound on $r^{\mathcal{X}}(t)$ over all possible schedules $\mathcal{X}$. This bound will be none other than $r^{\mathcal{Y}}(t)$.

Fix some t. We wish to minimize $r^{\mathcal{X}}(t)$, which is the weighted number of tasks remaining, but we are constrained by the fact that only so much work can be done in time t with m processors. That is, we have the following problem:

$$\begin{array}{lrcll} \text{minimize} & r^{\mathcal{X}}(t) & & & \\ \text{subject to} & \sum_{i=1}^n (1 - r_i^{\mathcal{X}}(t))p_i \cdot t_i(p_i) & \leq & mt & \\ & r_i^{\mathcal{X}}(t) & \leq & 1 & \text{for all } i \in [n] \\ & r_i^{\mathcal{X}}(t) & \geq & 0 & \text{for all } i \in [n]. \end{array} \tag{17}$$

Rather than minimizing the weighted number of tasks remaining, let us instead maximize the weighted number of tasks completed. If we let $x_i = 1 - r_i^{\mathcal{X}}(t)$ be the fraction of task i that is already completed at time t, and let $a_i = p_i \cdot t_i(p_i)$ be the work required

by task i, then instead of (17), we can solve the following:

$$\begin{array}{lrcll} \text{maximize} & \sum_{i=1}^{n} w_i x_i & & & \\ \text{subject to} & \sum_{i=1}^{n} a_i x_i & \leq & mt & \\ & x_i & \leq & 1 & \text{for all } i \in [n] \\ & x_i & \geq & 0 & \text{for all } i \in [n]. \end{array} \tag{18}$$

For the sake of simplicity, let us suppose that the tasks are ordered by nondecreasing ratio of work to weight. That is, $a_i/w_i \leq a_{i+1}/w_{i+1}$ for all $i \in [n-1]$. If $\sum_{i=1}^{n} a_i \leq mt$, then the optimal solution of (18) is $x_1^* = x_2^* = \cdots = x_n^* = 1$. If instead $\sum_{i=1}^{n} a_i > mt$, then let k be the smallest integer such that $\sum_{i=1}^{k} a_i > mt$. Then the optimal solution is

$$x_i^* = \begin{cases} 1 & \text{if } i \in \{1, 2, \ldots, k-1\} \\ 0 & \text{if } i \in \{k+1, k+2, \ldots, n\} \\ \frac{mt - \sum_{j=1}^{k-1} a_j}{a_k} & \text{if } i = k. \end{cases} \tag{19}$$

To verify that this is optimal, we can construct the dual problem and check that its optimal solution has the same objective function value. (See [13] for a discussion of linear programming and duality.) The dual of (18) is

$$\begin{array}{lrcll} \text{minimize} & mtz + \sum_{i=1}^{n} y_i & & & \\ \text{subject to} & a_i z + y_i & \geq & w_i & \text{for all } i \in [n] \\ & y_i & \geq & 0 & \text{for all } i \in [n] \\ & z & \geq & 0. & \end{array} \tag{20}$$

It has an optimal solution at $z^* = w_k/a_k$ and

$$y_i^* = \begin{cases} w_i - a_i \frac{w_k}{a_k} & \text{if } i \in \{1, 2, \ldots, k-1\} \\ 0 & \text{if } i \in \{k, k+1, \ldots, n\}. \end{cases} \tag{21}$$

Then we have

$$\begin{aligned}\sum_{i=1}^{n} w_i x_i^* &= \sum_{i=1}^{k-1} w_i + \left(\frac{mt - \sum_{j=1}^{k-1} a_j}{a_k}\right) w_k \\ &= mt\frac{w_k}{a_k} + \sum_{i=1}^{k-1}\left(w_i - a_i\frac{w_k}{a_k}\right) \\ &= mtz^* + \sum_{i=1}^{n} y_i^*, \end{aligned} \tag{22}$$

and since the two solutions are feasible, we conclude that they are also optimal.

Now to complete the proof, observe that the solution given in (19) corresponds to the schedule in which the squashed task set $T(\bar{q})$ is ordered by nondecreasing ratio of work to weight. That is, $r_i^{\mathcal{Y}}(t) = 1 - x_i^*$. Since $\sum w_i x_i^*$ is an upper bound on the weighted number of tasks completed at time t, we conclude that $r^{\mathcal{Y}}(t) = \sum w_i r_i^{\mathcal{Y}}(t)$ is a lower bound on the weighted number of tasks remaining at time t. □

Theorem 2.1.1 leads to the following family of lower bounds.

Corollary 2.1.1 *For any nonmalleable task set $T(\bar{p})$ and any $\theta \in [0,1]$, the total weighted completion time $R^{\mathcal{X}}$ of any schedule $\mathcal{X}$ for $T(\bar{p})$ is at least*

$$R^{\mathcal{X}} \geq \theta A_T(\bar{p}) + (1 - \frac{1}{2}\theta) H_T(\bar{p}) - \frac{1}{2}\theta U_T(\bar{p}). \tag{23}$$

Proof:

$$\begin{aligned} R^{\mathcal{X}} &= \theta R^{\mathcal{X}} + (1-\theta) R^{\mathcal{X}} \\ &\leq \theta(A_T(\bar{p}) + \frac{1}{2}H_T(\bar{p}) - \frac{1}{2}U_T(\bar{p})) + (1-\theta)H_T(\bar{p}) \\ &= \theta A_T(\bar{p}) + (1 - \frac{1}{2}\theta)H_T(\bar{p}) - \frac{1}{2}\theta U_T(\bar{p}). \end{aligned} \tag{24}$$

□

Let

$$L_T^{\theta}(\bar{p}) = \theta A_T(\bar{p}) + (1 - \frac{1}{2}\theta) H_T(\bar{p}) - \frac{1}{2}\theta U_T(\bar{p}). \tag{25}$$

Then by Corollary 2.1.1, for any $\theta \in [0,1]$, we have

$$L_T^{\theta}(\bar{p}) \leq R^*(T(\bar{p})). \tag{26}$$

So $L_T^{\theta}(\bar{p})$ is a lower bound on total weighted completion time for $T(\bar{p})$. The following corollary shows how to convert an upper bound on total weighted completion time relative to λ_A,λ_H, and λ_U into an upper bound relative to $L_T^{\theta}(\bar{p})$ for an appropriate θ.

Corollary 2.1.2 *If a schedule $\mathcal{X}$ satisfies $R^{\mathcal{X}} \leq \lambda_A A_T(\bar{p}) + \lambda_H H_T(\bar{p}) + \lambda_U U_T(\bar{p})$, where $\lambda_A > 0$, $\lambda_H > 0$, and $\lambda_A + \lambda_H + \lambda_U > 0$, then*

$$R^{\mathcal{X}} \leq \max\{\lambda_A, \frac{1}{2}\lambda_A + \lambda_H, \lambda_A + \lambda_H + \lambda_U\} \cdot L_T^{\theta}(\bar{p}), \tag{27}$$

where $\theta = \min\{1, \lambda_A/(\frac{1}{2}\lambda_A + \lambda_H), \lambda_A/(\lambda_A + \lambda_H + \lambda_U)\}$.

Proof: Let $\rho = \max\{\lambda_A, \frac{1}{2}\lambda_A + \lambda_H, \lambda_A + \lambda_H + \lambda_U\}$. We will show that $R^{\mathcal{X}} \leq \rho L_T^{\theta}(\bar{p})$ for

$$\theta = \frac{\lambda_A}{\rho} = \min\left\{1, \frac{\lambda_A}{\frac{1}{2}\lambda_A + \lambda_H}, \frac{\lambda_A}{\lambda_A + \lambda_H + \lambda_U}\right\}. \tag{28}$$

We will consider three cases corresponding to the possible values of ρ.

First suppose that $\rho = \lambda_A$. Then $\theta = 1$. Observe that $\rho = \lambda_A$ implies that $\frac{1}{2}\lambda_A \geq \lambda_H$ and $-\lambda_H \geq \lambda_U$. Then we have the following lower bound on total weighted completion time.

$$\begin{aligned}
L_T^{\theta}(\bar{p}) &= A_T(\bar{p}) + \frac{1}{2}H_T(\bar{p}) - \frac{1}{2}U_T(\bar{p}) \\
&= A_T(\bar{p}) + \frac{\lambda_H}{\lambda_A}H_T(\bar{p}) + \frac{\frac{1}{2}\lambda_A - \lambda_H}{\lambda_A}H_T(\bar{p}) - \frac{1}{2}U_T(\bar{p}) \\
&\geq A_T(\bar{p}) + \frac{\lambda_H}{\lambda_A}H_T(\bar{p}) + \frac{\frac{1}{2}\lambda_A - \lambda_H}{\lambda_A}U_T(\bar{p}) - \frac{\frac{1}{2}\lambda_A}{\lambda_A}U_T(\bar{p}) \\
&= A_T(\bar{p}) + \frac{\lambda_H}{\lambda_A}H_T(\bar{p}) - \frac{\lambda_H}{\lambda_A}U_T(\bar{p}) \\
&\geq A_T(\bar{p}) + \frac{\lambda_H}{\lambda_A}H_T(\bar{p}) + \frac{\lambda_U}{\lambda_A}U_T(\bar{p}).
\end{aligned} \tag{29}$$

Now we have

$$\begin{aligned} R^{\mathcal{X}} &\leq \lambda_A A_T(\bar{p}) + \lambda_H H_T(\bar{p}) + \lambda_U U_T(\bar{p}) \\ &\leq \lambda_A L_T^{\theta}(\bar{p}) \\ &= \rho L_T^{\theta}(\bar{p}). \end{aligned} \tag{30}$$

Next suppose that $\rho = \frac{1}{2}\lambda_A + \lambda_H$. Then we have $-\frac{1}{2}\lambda_A \geq \lambda_U$ and $\theta = \lambda_A/(\frac{1}{2}\lambda_A + \lambda_H)$, resulting in the following lower bound.

$$\begin{aligned} L_T^{\theta}(\bar{p}) &= \theta A_T(\bar{p}) + (1 - \frac{1}{2}\theta) H_T(\bar{p}) - \frac{1}{2}\theta U_T(\bar{p}) \\ &= \frac{\lambda_A}{\frac{1}{2}\lambda_A + \lambda_H} A_T(\bar{p}) + \frac{\lambda_H}{\frac{1}{2}\lambda_A + \lambda_H} H_T(\bar{p}) - \frac{\frac{1}{2}\lambda_A}{\frac{1}{2}\lambda_A + \lambda_H} U_T(\bar{p}) \\ &\geq \frac{\lambda_A}{\frac{1}{2}\lambda_A + \lambda_H} A_T(\bar{p}) + \frac{\lambda_H}{\frac{1}{2}\lambda_A + \lambda_H} H_T(\bar{p}) + \frac{\lambda_U}{\frac{1}{2}\lambda_A + \lambda_H} U_T(\bar{p}). \end{aligned} \tag{31}$$

We conclude that

$$\begin{aligned} R^{\mathcal{X}} &\leq \lambda_A A_T(\bar{p}) + \lambda_H H_T(\bar{p}) + \lambda_U U_T(\bar{p}) \\ &\leq (\frac{1}{2}\lambda_A + \lambda_H) L_T^{\theta}(\bar{p}) \\ &= \rho L_T^{\theta}(\bar{p}). \end{aligned} \tag{32}$$

Finally, suppose that $\rho = \lambda_A + \lambda_H + \lambda_U$. Then $\frac{1}{2}\lambda_A + \lambda_U \geq 0$ and $\theta = \lambda_A/(\lambda_A + \lambda_H + \lambda_U)$. Now we have

$$\begin{aligned} L_T^{\theta}(\bar{p}) &= \theta A_T(\bar{p}) + (1 - \frac{1}{2}\theta) H_T(\bar{p}) - \frac{1}{2}\theta U_T(\bar{p}) \\ &= \frac{\lambda_A}{\lambda_A + \lambda_H + \lambda_U} A_T(\bar{p}) + \frac{\frac{1}{2}\lambda_A + \lambda_H + \lambda_U}{\lambda_A + \lambda_H + \lambda_U} H_T(\bar{p}) - \frac{\frac{1}{2}\lambda_A}{\lambda_A + \lambda_H + \lambda_U} U_T(\bar{p}) \\ &= \frac{\lambda_A}{\lambda_A + \lambda_H + \lambda_U} A_T(\bar{p}) + \frac{\lambda_H}{\lambda_A + \lambda_H + \lambda_U} H_T(\bar{p}) + \frac{\frac{1}{2}\lambda_A + \lambda_U}{\lambda_A + \lambda_H + \lambda_U} H_T(\bar{p}) \\ &\quad - \frac{\frac{1}{2}\lambda_A}{\lambda_A + \lambda_H + \lambda_U} U_T(\bar{p}) \end{aligned}$$

$$\begin{aligned} &\geq \frac{\lambda_A}{\lambda_A+\lambda_H+\lambda_U}A_T(\bar{p}) + \frac{\lambda_H}{\lambda_A+\lambda_H+\lambda_U}H_T(\bar{p}) + \frac{\frac{1}{2}\lambda_A+\lambda_U}{\lambda_A+\lambda_H+\lambda_U}U_T(\bar{p}) \\ &\quad - \frac{\frac{1}{2}\lambda_A}{\lambda_A+\lambda_H+\lambda_U}U_T(\bar{p}) \\ &= \frac{\lambda_A}{\lambda_A+\lambda_H+\lambda_U}A_T(\bar{p}) + \frac{\lambda_H}{\lambda_A+\lambda_H+\lambda_U}H_T(\bar{p}) + \frac{\lambda_U}{\lambda_A+\lambda_H+\lambda_U}U_T(\bar{p}). \end{aligned} \tag{33}$$

We conclude that

$$\begin{aligned} R^{\mathcal{X}} &\leq \lambda_A A_T(\bar{p}) + \lambda_H H_T(\bar{p}) + \lambda_U U_T(\bar{p}) \\ &\leq (\lambda_A+\lambda_H+\lambda_U)L_T^{\theta}(\bar{p}) \\ &= \rho L_T^{\theta}(\bar{p}). \end{aligned} \tag{34}$$

□

An additional consequence of Theorem 2.1.1 is the following lower bound on total weighted completion time for malleable task sets, which follows directly from Corollary 2.1.1.

Corollary 2.1.3 *For any malleable task set T and any $\theta \in [0,1]$, the total weighted completion time $R^{\mathcal{X}}$ of any schedule $\mathcal{X}$ for T is at least*

$$R^{\mathcal{X}} \geq \min_{\bar{p}}\{\theta A_T(\bar{p}) + (1-\frac{1}{2}\theta)H_T(\bar{p}) - \frac{1}{2}\theta U_T(\bar{p})\}. \tag{35}$$

2.2 A List Scheduling Algorithm for NMWACT

We can use the results of the previous section to prove an approximation factor for a simple NMWACT algorithm.

The algorithm is as follows. Sort the tasks by nondecreasing ratio of work to weight. Schedule the first task to begin execution at time 0. Then schedule each subsequent

task i to begin execution at the earliest time when at least p_i processors are available, but no sooner than the previous task begins execution. This is the *least ratio first* (LRF) algorithm.

The main result of this section is the following theorem, which gives an upper bound on the total weighted completion time of a schedule produced by the LRF algorithm.

Theorem 2.2.1 *Let $T(\bar{p})$ be a nonmalleable task set, and let $\|\bar{p}\| = \max_i\{p_i\}$ be the maximum number of processors required by any task. Then*

$$R_{LRF}(T(\bar{p})) \leq \frac{m}{m - \|\bar{p}\| + 1}[A_T(\bar{p}) - U_T(\bar{p})] + H_T(\bar{p}). \tag{36}$$

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

Before proceeding with the proof of Theorem 2.2.1, let us recall the following notation. Let $\eta_i = p_i \cdot t_i(p_i)/w_i$ be the ratio of work to weight of task i, and let σ be a permutation that orders the tasks by nonincreasing ratio of work to weight. That is, $\eta_{\sigma^{-1}(s)} \geq \eta_{\sigma^{-1}(s+1)}$ for all $s \in [n-1]$. Then task $\sigma^{-1}(n)$ is scheduled first.

For the proof of Theorem 2.2.1, we make use of a lemma that gives an upper bound on the starting time of each task.

Lemma 2.2.1 *Let $\mathcal{X}$ denote the schedule produced by LRF for a given task set $T(\bar{p})$. Then the starting time $b_i^{\mathcal{X}}$ of task i in the schedule $\mathcal{X}$ satisfies*

$$b_i^{\mathcal{X}} \leq \frac{1}{m - \|\bar{p}\| + 1} \sum_{s=\sigma(i)+1}^{n} p_{\sigma^{-1}(s)} t_{\sigma^{-1}(s)}(p_{\sigma^{-1}(s)}). \tag{37}$$

Proof: Consider a task i. Execution of task i is scheduled to begin after tasks $\sigma^{-1}(\sigma(i)+1), \sigma^{-1}(\sigma(i)+2), \ldots, \sigma^{-1}(n)$ begin, but before any other tasks begin. Then not more than $\sum_{s=\sigma(i)+1}^{n} p_{\sigma^{-1}(s)} t_{\sigma^{-1}(s)}(p_{\sigma^{-1}(s)})$ work is done before task i begins execution. Observe that there are not more than $\|\bar{p}\| - 1$ idle processors at any time

when there is at least one task that has not been started yet. Then there are at least $m - \|\bar{p}\| + 1$ processors busy at all times until $b_i^{\mathcal{X}}$. Now the result follows directly. □

Proof of Theorem 2.2.1: Note that we can rewrite (37) as

$$b_i^{\mathcal{X}} \leq \frac{1}{m - \|\bar{p}\| + 1} \left[\sum_{s=\sigma(i)}^{n} p_{\sigma^{-1}(s)} t_{\sigma^{-1}(s)}(p_{\sigma^{-1}(s)}) - p_i t_i(p_i) \right]. \tag{38}$$

Then the total weighted completion time of the schedule produced by LRF for the task set $T(\bar{p})$ is

$$\begin{aligned} R_{\mathrm{LRF}}(T(\bar{p})) &= \sum_{i=1}^{n} w_i [b_i^{\mathcal{X}} + t_i(p_i)] \\ &= \sum_{i=1}^{n} w_i b_i^{\mathcal{X}} + \sum_{i=1}^{n} w_i t_i(p_i) \\ &\leq \frac{1}{m - \|\bar{p}\| + 1} \sum_{i=1}^{n} w_i \left[\sum_{s=\sigma(i)}^{n} p_{\sigma^{-1}(s)} t_{\sigma^{-1}(s)}(p_{\sigma^{-1}(s)}) - p_i t_i(p_i) \right] + H_T(\bar{p}) \\ &= \frac{m}{m - \|\bar{p}\| + 1} [A_T(\bar{p}) - U_T(\bar{p})] + H_T(\bar{p}). \end{aligned} \tag{39}$$

□

Corollary 2.2.1 *Let $T(\bar{p})$ be a nonmalleable task set such that $\|\bar{p}\| \leq \lceil \beta m \rceil$ for some $\beta \in (0,1)$. Then*

$$R_{LRF}(T(\bar{p})) \leq \frac{1}{1-\beta} A_T(\bar{p}) + H_T(\bar{p}) - \frac{1}{1-\beta} U_T(\bar{p}) \tag{40}$$

and

$$R_{LRF}(T(\bar{p})) \leq \max\left\{ \frac{1}{1-\beta}, \frac{1}{2(1-\beta)} + 1 \right\} \cdot L_T^{\theta}(\bar{p}), \tag{41}$$

where $\theta = \min\{1, 1/(3/2 - \beta)\}$.

Proof: Since $\|\bar{p}\| \leq \lceil \beta m \rceil \leq \beta m + 1$, applying Theorem 2.2.1 yields

$$\begin{aligned} R_{\mathrm{LRF}}(T(\bar{p})) &\leq \frac{m}{m - \|\bar{p}\| + 1} [A_T(\bar{p}) - U_T(\bar{p})] + H_T(\bar{p}) \\ &\leq \frac{m}{m - \beta m} [A_T(\bar{p}) - U_T(\bar{p})] + H_T(\bar{p}) \\ &= \frac{1}{1-\beta} A_T(\bar{p}) + H_T(\bar{p}) - \frac{1}{1-\beta} U_T(\bar{p}). \end{aligned} \tag{42}$$

Now applying Corollary 2.1.2 yields

$$\begin{aligned} R_{\mathrm{LRF}}(T(\bar{p})) &\leq \max\left\{\frac{1}{1-\beta}, \frac{1}{2(1-\beta)}+1, 1\right\} \cdot L_T^{\theta}(\bar{p}) \\ &= \max\left\{\frac{1}{1-\beta}, \frac{1}{2(1-\beta)}+1\right\} \cdot L_T^{\theta}(\bar{p}). \end{aligned} \tag{43}$$

□

2.3 An MWACT Algorithm Framework

We now turn our attention to malleable tasks. Observe that Corollary 2.1.3 defines an *allotment selection problem.* That is, for a given $\theta \in [0,1]$, let

$$L_T^{\theta} = \min_{\bar{p}}\{L_T^{\theta}(\bar{p})\} = \min_{\bar{p}}\{\theta A_T(\bar{p}) + (1-\frac{1}{2}\theta)H_T(\bar{p}) - \frac{1}{2}\theta U_T(\bar{p})\}. \tag{44}$$

Then the allotment selection problem is to find for a given θ an allotment $\bar{p}$ such that $L_T^{\theta}(\bar{p}) = L_T^{\theta}$. Let $R^*(T)$ denote the optimal value of $R^{\mathcal{X}}$ over all possible schedules $\mathcal{X}$ for the malleable task set T. Then for all $\theta \in [0,1]$, we have $L_T^{\theta} \leq R^*(T)$.

The results of the previous two sections suggest the following method of scheduling a set of malleable tasks T. First, find a solution $\bar{p}$ to the allotment selection problem. Then adjust the allotment $\bar{p}$ by taking $q_i = \min\{p_i, \lceil \beta m \rceil\}$ for a carefully chosen $\beta \in (0,1]$. Finally, schedule the nonmalleable task set $T(\bar{q})$ using an NMWACT algorithm. The following theorem gives an approximation factor for this method.

Theorem 2.3.1 *Suppose we are given an NMWACT algorithm $\mathcal{A}$ satisfying*

$$R_{\mathcal{A}}(T(\bar{q})) \leq \lambda_A A_T(\bar{q}) + \lambda_H H_T(\bar{q}) + \lambda_U U_T(\bar{q}) \tag{45}$$

for any nonmalleable task set $T(\bar{q})$ with $\|\bar{q}\| \leq \lceil \alpha m \rceil$. Let T be a malleable task set that satisfies the α-weak nondecreasing work condition. Let $\bar{p}$ be an allotment for T that

satisfies $L_T^\theta(\bar{p}) \leq \rho L_T^\theta$ *and* $\|\bar{p}\| \leq \lceil \beta m \rceil$, *where*

$$\theta = \min\left\{1, \frac{\lambda_A}{\frac{1}{2}\lambda_A + \lambda_H}, \frac{\lambda_A}{\lambda_A + \left[m\left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil}\right) + 1\right]\lambda_H + \lambda_U}\right\}, \tag{46}$$

and β *is such that* $\lceil \alpha m \rceil \leq \lceil \beta m \rceil$. *Let* $\bar{q}$ *be an allotment obtained from* $\bar{p}$ *by taking* $q_i = \min\{p_i, \lceil \alpha m \rceil\}$. *Then*

$$R_{\mathcal{A}}(T(\bar{q})) \leq \rho \cdot \max\left\{\lambda_A, \frac{1}{2}\lambda_A + \lambda_H, \lambda_A + \left[m\left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil}\right) + 1\right]\lambda_H + \lambda_U\right\} \cdot R^*(T). \tag{47}$$

Note that if $\lceil \alpha m \rceil = \lceil \beta m \rceil$, then $\bar{q} = \bar{p}$, so the allotment adjustment step is effectively omitted.

Applying this theorem to LRF using Corollary 2.2.1 yields the following corollary.

Corollary 2.3.1 *Suppose that the task set* T *satisfies the* α*-weak nondecreasing work condition. Let* $\bar{p}$ *be an allotment for* T *that satisfies* $L_T^\theta(\bar{p}) \leq \rho L_T^\theta$ *and* $\|\bar{p}\| \leq \lceil \beta m \rceil$, *where*

$$\theta = \min\left\{1, \frac{1}{\frac{3}{2} - \alpha}, \frac{1}{(1-\alpha)\left[m\left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil}\right) + 1\right]}\right\}, \tag{48}$$

and β *is such that* $\lceil \alpha m \rceil \leq \lceil \beta m \rceil$. *Let* $\bar{q}$ *be an allotment obtained from* $\bar{p}$ *by taking* $q_i = \min\{p_i, \lceil \alpha m \rceil\}$. *Then*

$$R_{LRF}(T(\bar{q})) \leq \rho \cdot \max\left\{\frac{1}{1-\alpha}, \frac{1}{2(1-\alpha)} + 1, m\left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil}\right) + 1\right\} \cdot R^*(T). \tag{49}$$

With these results in hand, it only remains to provide algorithms for finding exact or approximate solutions to the allotment selection problem. Note that it follows from Corollary 2.3.1 with $\beta = 1$ that an algorithm that solves the allotment selection problem exactly can be coupled with LRF to produce a 2-approximate algorithm for malleable task sets that satisfy the $\frac{1}{2}$-weak nondecreasing work condition.

For the proof of Theorem 2.3.1, we make use of the following lemma.

Lemma 2.3.1 *Let $\bar{p}$ be any allotment for a malleable task set T, and let β be such that $\|\bar{p}\| \leq \lceil \beta m \rceil$. Define an allotment $\bar{q}$ by $q_i = \min\{p_i, \lceil \alpha m \rceil\}$, where α is such that $\lceil \alpha m \rceil \leq \lceil \beta m \rceil$. Let $\lambda_A > 0$, $\lambda_H > 0$, and λ_U be such that $\lambda_A + \lambda_H + \lambda_U > 0$. Then if the task set T satisfies the α-weak nondecreasing work condition, then*

$$\begin{aligned} &\lambda_A A_T(\bar{q}) + \lambda_H H_T(\bar{q}) + \lambda_U U_T(\bar{q}) \\ &\quad \leq \lambda_A A_T(\bar{p}) + \lambda_H H_T(\bar{p}) + \left[m \left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil} \right) \lambda_H + \lambda_U \right] U_T(\bar{p}). \end{aligned} \tag{50}$$

Before we proceed with the proof of Lemma 2.3.1, it will be useful to extend the definition of $A_T(\bar{p})$. Recall that the definition of $A_T(\bar{p})$ given in (7) incorporates a permutation σ that orders the tasks by nonincreasing ratio of work to weight. Let us now extend this definition to incorporate an arbitrary permutation π. Let

$$A_T(\bar{p}, \pi) = \frac{1}{m} \sum_{i=1}^{n} (\bar{w})_i^{\pi} p_i t_i(p_i). \tag{51}$$

Let the ratio of work to weight for task i when it is assigned j processors be given by

$$\eta_i(j) = j t_i(j) / w_i. \tag{52}$$

We will say that an allotment $\bar{p}$ *induces a permutation* σ if

$$\eta_{\sigma^{-1}(s)}(p_{\sigma^{-1}(s)}) \geq \eta_{\sigma^{-1}(s+1)}(p_{\sigma^{-1}(s+1)}) \text{ for all } s \in [n-1]. \tag{53}$$

Observe that in the definition of $A_T(\bar{p})$ in (7), we made use of a permutation induced by $\bar{p}$. So if an allotment $\bar{p}$ induces a permutation σ, then we have $A_T(\bar{p}) = A_T(\bar{p}, \sigma)$. The following lemma gives another characterization of a permutation σ that is induced by an allotment $\bar{p}$.

Lemma 2.3.2 *A permutation σ is induced by an allotment $\bar{p}$ if and only if $A_T(\bar{p}, \sigma) = \min_{\pi}\{A_T(\bar{p}, \pi)\}$.*

Proof: For the "if" part, we will show that if a permutation ϕ is not induced by $\bar{p}$, then it satisfies $A_T(\bar{p},\phi) > \min_\pi\{A_T(\bar{p},\pi)\}$. Since ϕ is not induced by $\bar{p}$, there is a pair of consecutive tasks in the order ϕ such that the first has a smaller ratio of work to weight than the second. That is, there exists $r \in [n-1]$ such that $\eta_{\phi^{-1}(r)}(p_{\phi^{-1}(r)}) < \eta_{\phi^{-1}(r+1)}(p_{\phi^{-1}(r+1)})$. Let $k = \phi^{-1}(r)$ and let $l = \phi^{-1}(r+1)$. Then we have

$$\frac{p_k t_k(p_k)}{w_k} < \frac{p_l t_l(p_l)}{w_l}. \tag{54}$$

Now define a permutation ψ by interchanging the pair of tasks that is out of order. That is, let

$$\psi(i) = \begin{cases} r+1 & \text{if } i = k \\ r & \text{if } i = l \\ \phi(i) & \text{otherwise.} \end{cases} \tag{55}$$

Now we have

$$\begin{aligned} A_T(\bar{p},\phi) - A_T(\bar{p},\psi) &= \frac{1}{m}\sum_{i=1}^{n}(\bar{w})_i^{\phi} p_i t_i(p_i) - \frac{1}{m}\sum_{i=1}^{n}(\bar{w})_i^{\psi} p_i t_i(p_i) \\ &= \frac{1}{m}\sum_{i=1}^{n}\left[(\bar{w})_i^{\phi} - (\bar{w})_i^{\psi}\right] p_i t_i(p_i). \end{aligned} \tag{56}$$

Observe that

$$(\bar{w})_i^{\phi} - (\bar{w})_i^{\psi} = \begin{cases} -w_l & \text{if } i = k \\ w_k & \text{if } i = l \\ 0 & \text{otherwise.} \end{cases} \tag{57}$$

Then we have

$$\begin{aligned} A_T(\bar{p},\phi) - A_T(\bar{p},\psi) &= -w_l p_k t_k(p_k) + w_k p_l t_l(p_l) \\ &= w_k w_l \left[-\frac{p_k t_k(p_k)}{w_k} + \frac{p_l t_l(p_l)}{w_l}\right] \\ &> 0. \end{aligned} \tag{58}$$

We conclude that $A_T(\bar{p}, \phi) > \min_\pi\{A_T(\bar{p}, \pi)\}$.

Before proceeding with the "only if" part, consider a permutation ϕ that satisfies $A_T(\bar{p}, \phi) = \min_\pi\{A_T(\bar{p}, \pi)\}$. We have shown that this implies that ϕ is induced by $\bar{p}$. Now by definition we have $A_T(\bar{p}) = A_T(\bar{p}, \phi)$, and therefore

$$A_T(\bar{p}) = \min_\pi\{A_T(\bar{p}, \pi)\}. \tag{59}$$

Now for the "only if" part, let σ be a permutation that is induced by $\bar{p}$. Then by definition we have $A_T(\bar{p}) = A_T(\bar{p}, \sigma)$, and so it follows from (59) that $A_T(\bar{p}, \sigma) = \min_\pi\{A_T(\bar{p}, \pi)\}$, as required. □

Now the following characterization of $A_T(\bar{p})$ follows directly from Lemma 2.3.2.

Corollary 2.3.2

$$A_T(\bar{p}) = \min_\pi\{A_T(\bar{p}, \pi)\}. \tag{60}$$

Proof of Lemma 2.3.1: We will first prove the following three inequalities:

$$A_T(\bar{q}) - U_T(\bar{q}) \leq A_T(\bar{p}) - U_T(\bar{p}); \tag{61}$$

$$H_T(\bar{q}) - U_T(\bar{q}) \leq H_T(\bar{p}) + \left[m\left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil}\right) - 1\right] U_T(\bar{p}); \tag{62}$$

$$U_T(\bar{q}) \leq U_T(\bar{p}). \tag{63}$$

Then we will conclude that

$$\lambda_A(A_T(\bar{q}) - U_T(\bar{q})) \leq \lambda_A(A_T(\bar{p}) - U_T(\bar{p})), \tag{64}$$

$$\lambda_H(H_T(\bar{q}) - U_T(\bar{q})) \leq \lambda_H\left(H_T(\bar{p}) + \left[m\left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil}\right) - 1\right] U_T(\bar{p})\right), \tag{65}$$

$$(\lambda_A + \lambda_H + \lambda_U)U_T(\bar{q}) \leq (\lambda_A + \lambda_H + \lambda_U)U_T(\bar{p}), \tag{66}$$

and the lemma follows from adding (64), (65), and (66) together.

Observe that since $q_i = \min\{p_i, \lceil \alpha m \rceil\}$, the α-weak nondecreasing work condition guarantees that

$$q_i t_i(q_i) \leq p_i t_i(p_i) \text{ for all } i \in [n]. \tag{67}$$

Then (63) follows directly.

Next, we will show (61). Let σ be a permutation that is induced by $\bar{p}$. Then by Corollary 2.3.2, we have

$$A_T(\bar{q}) \leq A_T(\bar{q}, \sigma). \tag{68}$$

This fact together with (67) yields

$$\begin{aligned} A_T(\bar{q}) - U_T(\bar{q}) &\leq A_T(\bar{q}, \sigma) - U_T(\bar{q}) \\ &= \frac{1}{m} \sum_{i=1}^{n} (\bar{w})_i^{\sigma} q_i t_i(q_i) - \frac{1}{m} \sum_{i=1}^{n} w_i q_i t_i(q_i) \\ &= \frac{1}{m} \sum_{i=1}^{n} \left[(\bar{w})_i^{\sigma} - w_i \right] q_i t_i(q_i) \\ &\leq \frac{1}{m} \sum_{i=1}^{n} \left[(\bar{w})_i^{\sigma} - w_i \right] p_i t_i(p_i) \\ &= A_T(\bar{p}) - U_T(\bar{p}). \end{aligned} \tag{69}$$

Finally, we will show (62). For this purpose, it is sufficient to show that for each task i, we have

$$t_i(q_i) - \frac{1}{m} q_i t_i(q_i) \leq t_i(p_i) + \left[m \left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil} \right) - 1 \right] \frac{1}{m} p_i t_i(p_i). \tag{70}$$

Pick any task i. Note that (70) holds if $p_i \leq \lceil \alpha m \rceil$, because in that case $q_i = p_i$, and by hypothesis, $1/\lceil \alpha m \rceil - 1/\lceil \beta m \rceil \geq 0$.

Suppose instead that $p_i > \lceil \alpha m \rceil$. Then $q_i = \lceil \alpha m \rceil$. Let

$$\epsilon = \frac{1}{m} p_i t_i(p_i) - \frac{1}{m} q_i t_i(q_i), \tag{71}$$

and observe that (67) guarantees that $\epsilon \geq 0$. Now we have

$$\begin{aligned} t_i(q_i) &= \frac{1}{\lceil \alpha m \rceil} q_i t_i(q_i) \\ &= \frac{1}{\lceil \alpha m \rceil} p_i t_i(p_i) - \frac{m}{\lceil \alpha m \rceil} \epsilon \\ &= \frac{p_i}{\lceil \beta m \rceil} t_i(p_i) + \left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil} \right) p_i t_i(p_i) - \frac{m}{\lceil \alpha m \rceil} \epsilon \\ &\leq t_i(p_i) + m \left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil} \right) \frac{1}{m} p_i t_i(p_i) - \epsilon. \end{aligned} \tag{72}$$

□

Then (70) follows directly.

Proof of Theorem 2.3.1: First observe that since (45) holds for all nonmalleable task sets $T(\bar{q})$ with $\|\bar{q}\| \leq \lceil \alpha m \rceil$, we can conclude that $\lambda_A > 0$, $\lambda_H > 0$, and $\lambda_A + \lambda_H + \lambda_U > 0$. Otherwise, we could construct a task set for which there is no schedule that satisfies the stated upper bound on $R_{\mathcal{A}}(T(\bar{q}))$. Therefore, we can apply Lemma 2.3.1, along with Corollaries 2.1.2 and 2.1.3, to get

$$\begin{aligned} R_{\mathcal{A}}(T(\bar{q})) &\leq \lambda_A A_T(\bar{q}) + \lambda_H H_T(\bar{q}) + \lambda_U U_T(\bar{q}) \\ &\leq \lambda_A A_T(\bar{p}) + \lambda_H H_T(\bar{p}) + \left[m \left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil} \right) \lambda_H + \lambda_U \right] U_T(\bar{p}) \\ &\leq \max \left\{ \lambda_A, \frac{1}{2}\lambda_A + \lambda_H, \lambda_A + \left[m \left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil} \right) + 1 \right] \lambda_H + \lambda_U \right\} \cdot L_T^{\theta}(\bar{p}) \\ &\leq \rho \cdot \max \left\{ \lambda_A, \frac{1}{2}\lambda_A + \lambda_H, \lambda_A + \left[m \left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil} \right) + 1 \right] \lambda_H + \lambda_U \right\} \cdot L_T^{\theta} \\ &\leq \rho \cdot \max \left\{ \lambda_A, \frac{1}{2}\lambda_A + \lambda_H, \lambda_A + \left[m \left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil} \right) + 1 \right] \lambda_H + \lambda_U \right\} \\ &\quad \cdot R^*(T). \end{aligned} \tag{73}$$

□

Chapter 3

Allotment Selection Algorithms

We have seen in the previous chapter that an algorithm for the allotment selection problem (44) can serve as the basis for an MWACT algorithm. In this chapter, we present algorithms that select for any $\theta \in [0, 1]$ an allotment of processors $\bar{p}$ to a set T of malleable tasks such that $L_T^\theta(\bar{p}) \leq \rho L_T^\theta$, where ρ is some constant. For the case of equal task weights, we present a polynomial-time algorithm in Section 3.2 that solves the problem exactly ($\rho = 1$). We also present two faster algorithms in Sections 3.3 and 3.4 with $\rho = 2$ and $\rho = 4$. For the general case of task weights, the allotment selection problem is not known to be solvable in polynomial time. (Neither is it known to be NP-complete.) However, the 2-approximate algorithm mentioned above also applies to this case. Finally, in Section 3.5 we will consider a scheduling environment in which tasks arrive over time, and their execution time functions are unknown. We present an allotment selection policy for this environment that is based on the algorithm of Section 3.4, and we use experiments to evaluate its performance relative to other policies.

Before describing the algorithms, we will begin by recasting the allotment selection

problem as a problem of putting the tasks in the right order.

3.1 Choosing a Permutation

Rather than choosing an allotment and letting that determine a permutation of the tasks according to work-to-weight ratio, we will choose a permutation and let that determine an allotment. The first step in this direction is to isolate each individual task's contribution to the objective function $L_T^\theta(\bar{p})$ of the allotment selection problem. Recall that the definition of $A_T(\bar{p})$ incorporates a particular permutation of the tasks, and therefore so does the definition of $L_T^\theta(\bar{p})$. We now extend the definition of $L_T^\theta(\bar{p})$ to an arbitrary permutation π, using the extended definition of $A_T(\bar{p})$ given in (51). Let

$$L_T^\theta(\bar{p},\pi) = \theta A_T(\bar{p},\pi) + (1-\frac{1}{2}\theta)H_T(\bar{p}) - \frac{1}{2}\theta U_T(\bar{p}). \tag{74}$$

Then it follows from Corollary 2.3.2 that

$$L_T^\theta(\bar{p}) = \min_\pi\{L_T^\theta(\bar{p},\pi)\}. \tag{75}$$

Since by definition $L_T^\theta = \min_{\bar{p}}\{L_T^\theta(\bar{p})\}$, we now have

$$L_T^\theta = \min_{\bar{p}}\min_\pi\{L_T^\theta(\bar{p},\pi)\}. \tag{76}$$

Now let us consider each task's contribution to $L_T^\theta(\bar{p},\pi)$. Expanding $L_T^\theta(\bar{p},\pi)$ using the definitions of $A_T(\bar{p},\pi)$, $H_T(\bar{p})$, and $U_T(\bar{p})$, we get

$$\begin{aligned} L_T^\theta(\bar{p},\pi) &= \theta A_T(\bar{p},\pi) + (1-\frac{1}{2}\theta)H_T(\bar{p}) - \frac{1}{2}\theta U_T(\bar{p}) \\ &= \frac{\theta}{m}\sum_{i=1}^n (\bar{w})_i^\pi p_i t_i(p_i) + (1-\frac{1}{2}\theta)\sum_{i=1}^n w_i t_i(p_i) - \frac{\theta}{2m}\sum_{i=1}^n w_i p_i t_i(p_i) \\ &= \sum_{i=1}^n \left[\frac{\theta p_i}{m}\left((\bar{w})_i^\pi - \frac{1}{2}w_i\right) + (1-\frac{1}{2}\theta)w_i\right] t_i(p_i). \end{aligned} \tag{77}$$

Now if we define

$$C_i(\mu, j) = \left\lceil \frac{\theta j}{m} \left(\mu - \frac{1}{2} w_i \right) + (1 - \frac{1}{2}\theta) w_i \right\rceil t_i(j), \tag{78}$$

then from (77) we see that the contribution of task i to $L_T^\theta(\bar{p}, \pi)$ is given by $C_i((\bar{w})_i^\pi, p_i)$. Therefore, we have

$$L_T^\theta(\bar{p}, \pi) = \sum_{i=1}^{n} C_i((\bar{w})_i^\pi, p_i). \tag{79}$$

Now a permutation π determines an allotment of processors to task i in the following way. Let

$$C_i(\mu) = \min_{j \in M_i} \{C_i(\mu, j)\}, \tag{80}$$

and let

$$\kappa_i(\mu) = \min\{j \in M_i : C_i(\mu, j) = C_i(\mu)\} \tag{81}$$

be a number of processors that achieves the minimum. Then for a given permutation π, the contribution of task i to $L_T^\theta(\bar{p}, \pi)$ is minimized by assigning $p_i = \kappa_i((\bar{w})_i^\pi)$ processors to task i. Now we are ready to prove that the problem of choosing an allotment can be viewed instead as a problem of choosing a permutation.

Lemma 3.1.1

$$L_T^\theta = \min_\pi \left\{ \sum_{i=1}^{n} C_i((\bar{w})_i^\pi) \right\}. \tag{82}$$

Proof: Starting from (76) and applying (79), we have

$$\begin{aligned} L_T^\theta &= \min_{\bar{p}} \min_\pi \{L_T^\theta(\bar{p}, \pi)\} \\ &= \min_\pi \min_{\bar{p}} \{L_T^\theta(\bar{p}, \pi)\} \\ &= \min_\pi \min_{\bar{p}} \left\{ \sum_{i=1}^{n} C_i((\bar{w})_i^\pi, p_i) \right\}. \end{aligned} \tag{83}$$

Observe that for a fixed permutation π, the functions $C_i((\bar{w})_i^{\pi}, p_i)$ are independent, and so each function can be minimized separately. Using this fact and then applying (80) yields

$$\begin{aligned} L_T^{\theta} &= \min_{\pi}\left\{\sum_{i=1}^{n} \min_{j\in M_i}\{C_i((\bar{w})_i^{\pi}, j)\}\right\} \\ &= \min_{\pi}\left\{\sum_{i=1}^{n} C_i((\bar{w})_i^{\pi})\right\}. \end{aligned} \tag{84}$$

□

From now on, we will view the allotment selection problem in terms of (82), rather than (44).

3.2 An Exact Algorithm for the Unweighted Case

Let us now restrict our attention to the case where $w_i = 1$ for all $i \in [n]$. Observe that in this case, $(\bar{w})_i^{\pi} = \pi(i)$. Then from Lemma 3.1.1 we have

$$L_T^{\theta} = \min_{\pi}\left\{\sum_{i=1}^{n} C_i(\pi(i))\right\}. \tag{85}$$

Now the contribution $C_i(\pi(i))$ of task i depends only on the position $\pi(i)$ of task i, and not on the positions of any of the other tasks in the order π. Therefore the problem given by (85) is a weighted bipartite matching problem. Construct a $2n$-node bipartite graph $G = (V_1, V_2, E)$. The n vertices in V_1 correspond to tasks, and the n vertices in V_2 correspond to possible positions in an ordering of the tasks. Assign weight $C_i(s)$ to edge (i, s). Find a minimum-weight perfect matching M in G. Let π be the permutation that corresponds to the matching M. That is, if the edge (i, s) is present in the matching M, then $\pi(i) = s$. Then construct an allotment $\bar{p}$ by taking

$p_i = \kappa_i(\pi(i))$. We refer to this as the *exact* algorithm, since it gives an exact solution to the allotment selection problem.

Theorem 3.2.1 *For any set T of malleable tasks with $w_i = 1$ for all i, and any $\theta \in [0,1]$, the exact algorithm finds an allotment $\bar{p}$ that satisfies $L_T^\theta(\bar{p}) = L_T^\theta$. Its running time is $O(n^3 + mn)$.*

Corollary 3.2.1 *Suppose we are given an NMACT algorithm $\mathcal{A}$ with running time $Q(m,n)$ such that $R_{\mathcal{A}}(T(\bar{q})) \leq \lambda_A A_T(\bar{q}) + \lambda_H H_T(\bar{q}) + \lambda_U U_T(\bar{q})$ for all nonmalleable task sets $T(\bar{q})$ satisfying $\|\bar{q}\| \leq \lceil \alpha m \rceil$. Then there is a MACT algorithm with approximation factor $\max\{\lambda_A, \frac{1}{2}\lambda_A + \lambda_H, \lambda_A + \frac{1}{\alpha}\lambda_H + \lambda_U\}$ and running time $O(n^3 + mn) + Q(m,n)$ for task sets that satisfy the α-weak nondecreasing work condition.*

Proof: Use the exact algorithm to select an allotment, and then apply Theorem 2.3.1 with $\rho = 1$ and $\beta = 1$. □

Using the exact algorithm in conjunction with LRF yields the following result.

Corollary 3.2.2 *There is a MACT algorithm with approximation factor $\max\{\frac{1}{1-\alpha}, \frac{1}{\alpha}\}$ and running time $O(n^3 + mn)$ for task sets that satisfy the α-weak nondecreasing work condition.*

Proof: Use the exact algorithm to select an allotment, and then apply Corollary 2.3.1 with $\rho = 1$ and $\beta = 1$. □

We have not yet specified how the edge weights $C_i(s)$ will be computed, and how a minimum weight matching will be found. The latter can be done in $O(n^3)$ steps using the Hungarian method, which is due to Kuhn [32, 43].

As for computing the edge weights, one simple method is to compute $C_i(s,j)$ for all $i, s \in [n]$ and all $j \in M_i$. This requires $O(mn^2)$ steps. However, there is a faster way.

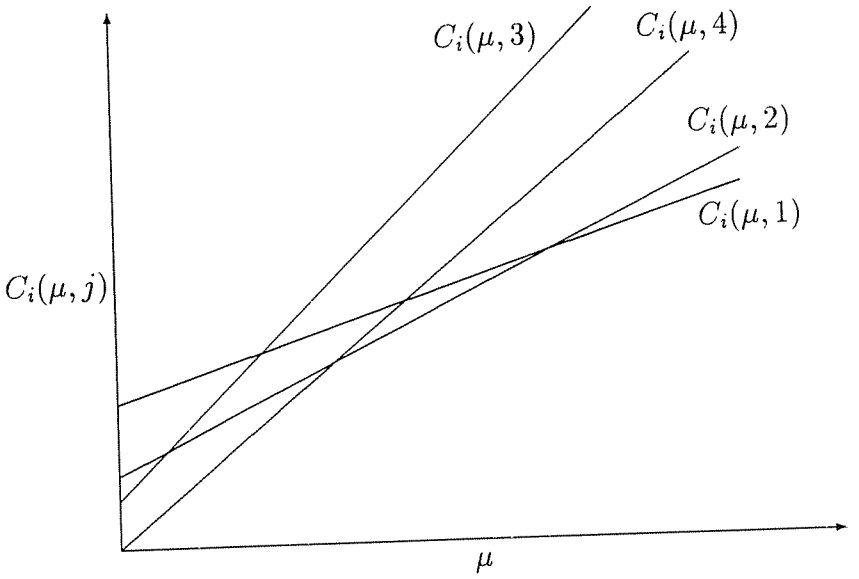


Figure 1: The functions $C_i(\mu, j)$ associated with a task i

Observe that for any fixed task i and number of processors j, the contribution $C_i(\mu, j)$ is a linear function of μ. Then for a given task i we have m linear functions that partition the range $(0, \sum_{l=1}^{n} w_l]$ into m or fewer subintervals. Each subinterval is the range over which the linear function corresponding to a particular j is the minimum of the m functions. So for each subinterval, there is some j such that every μ in that subinterval has $\kappa_i(\mu) = j$. See Figure 3.2. Note that the slope of the line corresponding to j processors is directly proportional to the work required by task i when it is assigned j processors. The following lemma shows that the subintervals can be computed quickly for each task i.

Lemma 3.2.1 *For all $i \in [n]$, there exist v_{ij} such that $0 = v_{i,m+1} \leq v_{im} \leq \cdots \leq v_{i2} \leq v_{i1} = \sum_{l=1}^{n} w_l$, and for all $j \in [m]$, if $\mu \in (v_{i,j+1}, v_{ij}]$, then $C_i(\mu) = C_i(\mu, j)$. Furthermore, $v_{i1}, v_{i2}, \ldots, v_{i,m+1}$ can be computed in $O(m)$ steps for any task i.*

The proof of this lemma is rather tedious, and has been banished to the Appendix.

For a given task i, once the subintervals have been computed, then $C_i(s)$ can be computed for all $s \in [n]$ in a total of $O(n+m)$ steps. Thus the total time to compute all the edge weights $C_i(s)$ is $O(n^2+mn)$, and the total running time of the exact algorithm is $O(n^3 + mn)$. This completes the proof of Theorem 3.2.1.

3.3 An Approximation Algorithm for the Weighted Case

If we wish to have an algorithm that is faster than the exact algorithm, and we are willing to settle for an approximation, then a simple greedy approach is a natural candidate. This approach has the added benefit that it also applies to the general case of weighted tasks.

Note that in the case of weighted tasks, the contribution $C_i((\bar{w})_i^{\pi})$ of task i depends not only on the position $\pi(i)$ of task i, but also on which tasks precede it in the order π — or more precisely, on the total weight of those tasks. Therefore, the problem of choosing a permutation of weighted tasks is not simply a weighted bipartite matching problem.

The greedy algorithm determines an order ϕ by assigning tasks to positions one at a time, in the following manner. Start by setting $s := n$ and $\mu_n := \sum_{i=1}^{n} w_i$. In each iteration, choose from among the tasks that have not yet been assigned to positions the task with the smallest value of $C_i(\mu_s)/w_i$. Let i_s denote this task, and assign it to position s. That is, let $\phi(i_s) := s$. Then let $\mu_{s-1} := \mu_s - w_{i_s}$ be the total weight of the tasks that remain, and continue with $s := s - 1$. Repeat this process until

every task has been assigned to a position. Finally, construct an allotment $\bar{p}$ by taking $p_i = \kappa_i((\bar{w})_i^{\phi})$.

Theorem 3.3.1 *For any set T of malleable tasks and any $\theta \in [0,1]$, the greedy algorithm finds an allotment $\bar{p}$ that satisfies $L_T^{\theta}(\bar{p}) \leq 2L_T^{\theta}$. Its running time is $O(n^2 + mn)$.*

Corollary 3.3.1 *Suppose we are given an NMWACT algorithm $\mathcal{A}$ with running time $Q(m,n)$ such that $R_{\mathcal{A}}(T(\bar{q})) \leq \lambda_A A_T(\bar{q}) + \lambda_H H_T(\bar{q}) + \lambda_U U_T(\bar{q})$ for all nonmalleable task sets $T(\bar{q})$ satisfying $\|\bar{q}\| \leq \lceil \alpha m \rceil$. Then there is an MWACT algorithm with approximation factor $2 \cdot \max\{\lambda_A, \frac{1}{2}\lambda_A + \lambda_H, \lambda_A + \frac{1}{\alpha}\lambda_H + \lambda_U\}$ and running time $O(n^2 + mn) + Q(m,n)$ for task sets that satisfy the α-weak nondecreasing work condition.*

Proof: Use the greedy algorithm to select an allotment, and then apply Theorem 2.3.1 with $\rho = 2$ and $\beta = 1$. □

Using the greedy algorithm in conjunction with LRF yields the following result.

Corollary 3.3.2 *There is an MWACT algorithm with approximation factor $2 \cdot \max\{\frac{1}{1-\alpha}, \frac{1}{\alpha}\}$ and running time $O(n^2 + mn)$ for task sets that satisfy the α-weak nondecreasing work condition.*

Proof: Use the greedy algorithm to select an allotment, and then apply Corollary 2.3.1 with $\rho = 2$ and $\beta = 1$. □

To prove an approximation factor for the greedy algorithm, we need an upper bound on $L_T^{\theta}(\bar{p})/L_T^{\theta}$, where $\bar{p}$ is the greedy allotment. We can obtain a lower bound on L_T^{θ} by considering how much the objective function value $\sum_{i=1}^{n} C_i((\bar{w})_i^{\phi})$ can be reduced by changing the order of the tasks from the greedy order ϕ to some other order ϕ'. For this purpose, given the contribution $C_i((\bar{w})_i^{\phi})$ of task i under the order ϕ, we will prove

a lower bound on the contribution $C_i((\bar{w})_i^{\phi'})$ of that task under the order ϕ'. The next two lemmas provide this bound.

Lemma 3.3.1 *Let $\mu, \mu' \geq w_i$. Then*

1. *if $\mu' \leq \mu$, then $C_i(\mu') \geq \frac{\mu'}{\mu} C_i(\mu)$;*

2. *if $\mu' \geq \mu$, then $C_i(\mu') \geq C_i(\mu)$.*

For any x, let $\langle x \rangle = \min\{x, 1\}$. Then Lemma 3.3.1 can be restated as $C_i(\mu') \geq \left\langle \frac{\mu'}{\mu} \right\rangle C_i(\mu)$.

Proof:

$$\begin{aligned}
C_i(\mu') &= \left[\frac{\theta\kappa_i(\mu')}{m}(\mu' - \frac{1}{2}w_i) + (1 - \frac{1}{2}\theta)w_i\right] t_i(\kappa_i(\mu')) \\
&= \frac{\frac{\theta\kappa_i(\mu')}{m}(\mu' - \frac{1}{2}w_i) + (1 - \frac{1}{2}\theta)w_i}{\frac{\theta\kappa_i(\mu')}{m}(\mu - \frac{1}{2}w_i) + (1 - \frac{1}{2}\theta)w_i} \left[\frac{\theta\kappa_i(\mu')}{m}(\mu - \frac{1}{2}w_i) + (1 - \frac{1}{2}\theta)w_i\right] t_i(\kappa_i(\mu')) \\
&= \frac{\frac{\theta\kappa_i(\mu')}{m}(\mu' - \frac{1}{2}w_i) + (1 - \frac{1}{2}\theta)w_i}{\frac{\theta\kappa_i(\mu')}{m}(\mu - \frac{1}{2}w_i) + (1 - \frac{1}{2}\theta)w_i} C_i(\mu, \kappa_i(\mu')) \\
&\geq \frac{\frac{\theta\kappa_i(\mu')}{m}(\mu' - \frac{1}{2}w_i) + (1 - \frac{1}{2}\theta)w_i}{\frac{\theta\kappa_i(\mu')}{m}(\mu - \frac{1}{2}w_i) + (1 - \frac{1}{2}\theta)w_i} C_i(\mu). \qquad (86)
\end{aligned}$$

If $\mu' \geq \mu$, then the result is immediate. If $\mu' \leq \mu$, then we have

$$\begin{aligned}
C_i(\mu') &\geq \frac{\frac{\theta\kappa_i(\mu')}{m}(\mu' - \frac{1}{2}w_i) + (1 - \frac{1}{2}\theta)w_i}{\frac{\theta\kappa_i(\mu')}{m}(\mu - \frac{1}{2}w_i) + (1 - \frac{1}{2}\theta)w_i} C_i(\mu) \\
&\geq \frac{\theta(\mu' - \frac{1}{2}w_i) + (1 - \frac{1}{2}\theta)w_i}{\theta(\mu - \frac{1}{2}w_i) + (1 - \frac{1}{2}\theta)w_i} C_i(\mu) \\
&= \frac{\theta\mu' + (1 - \theta)w_i}{\theta\mu + (1 - \theta)w_i} C_i(\mu) \\
&\geq \frac{\mu'}{\mu} C_i(\mu). \qquad (87)
\end{aligned}$$

□

Lemma 3.3.2 *Let ϕ be the permutation produced by the greedy algorithm, and let ϕ' be any permutation. Then $C_i((\bar{w})_i^{\phi'}) \geq \frac{w_i}{w_k}\left\langle \frac{(\bar{w})_i^{\phi'}}{(\bar{w})_k^{\phi}} \right\rangle C_k((\bar{w})_k^{\phi})$ for all k such that $\phi(k) \geq \phi(i)$.*

Proof: By Lemma 3.3.1, we have $C_i((\bar{w})_i^{\phi'}) \geq \left\langle \frac{(\bar{w})_i^{\phi'}}{(\bar{w})_k^{\phi}} \right\rangle C_i((\bar{w})_k^{\phi})$. Since $\phi(k) \geq \phi(i)$, and because of the way the greedy algorithm selects a task for each position, we also have $C_i((\bar{w})_k^{\phi})/w_i \geq C_k((\bar{w})_k^{\phi})/w_k$. The result follows directly. □

Now to obtain a lower bound on L_T^{θ}, let a_s denote the contribution and b_s denote the weight of the task in position s in the greedy order ϕ. Then $a_{\phi(i)} = C_i((\bar{w})_i^{\phi})$ and $b_{\phi(i)} = w_i$. Let π be a permutation that, when applied to the greedy-ordered tasks, yields the permutation ϕ'. That is, $\pi(\phi(i)) = \phi'(i)$ for all $i \in [n]$. For any vector $\bar{x}$, define $(\bar{x})_i = \sum_{k=1}^{i} x_k$ to be the sum of the first i elements of $\bar{x}$. Note that

$$(\bar{w})_i^{\phi} = \sum_{s=1}^{\phi(i)} w_{\phi^{-1}(s)} = \sum_{s=1}^{\phi(i)} b_s = (\bar{b})_{\phi(i)} \tag{88}$$

and

$$(\bar{w})_i^{\phi'} = \sum_{s=1}^{\phi'(i)} w_{\phi'^{-1}(s)} = \sum_{s=1}^{\pi(\phi(i))} w_{\phi^{-1}(\pi^{-1}(s))} = \sum_{s=1}^{\pi(\phi(i))} b_{\pi^{-1}(s)} = (\bar{b})_{\phi(i)}^{\pi}. \tag{89}$$

Then it follows from Lemma 3.3.2 that

$$\begin{aligned} C_i((\bar{w})_i^{\phi'}) &\geq \max_{k:\phi(k)\geq\phi(i)} \left\{ \frac{w_i}{w_k} \left\langle \frac{(\bar{w})_i^{\phi'}}{(\bar{w})_k^{\phi}} \right\rangle C_k((\bar{w})_k^{\phi}) \right\} \\ &= \max_{k:\phi(k)\geq\phi(i)} \left\{ \frac{b_{\phi(i)}}{b_{\phi(k)}} \left\langle \frac{(\bar{b})_{\phi(i)}^{\pi}}{(\bar{b})_{\phi(k)}} \right\rangle a_{\phi(k)} \right\} \\ &= \max_{l\geq\phi(i)} \left\{ \frac{b_{\phi(i)}}{b_l} \left\langle \frac{(\bar{b})_{\phi(i)}^{\pi}}{(\bar{b})_l} \right\rangle a_l \right\}. \end{aligned} \tag{90}$$

Now let

$$f_s(\bar{a}, \bar{b}, \pi) = \max_{l\geq s} \left\{ \frac{b_s}{b_l} \left\langle \frac{(\bar{b})_s^{\pi}}{(\bar{b})_l} \right\rangle a_l \right\}, \tag{91}$$

and let

$$F(\bar{a}, \bar{b}, \pi) = \sum_{s=1}^{n} f_s(\bar{a}, \bar{b}, \pi). \tag{92}$$

Then we have

$$C_i((\bar{w})_i^{\phi'}) \geq f_{\phi(i)}(\bar{a}, \bar{b}, \pi). \tag{93}$$

Therefore, any malleable task set T defines a pair of vectors $\bar{a}$ and $\bar{b}$ that satisfy

$$\min_{\pi}\{F(\bar{a}, \bar{b}, \pi)\} \leq \min_{\phi'}\left\{\sum_{i=1}^{n} C_i((\bar{w})_i^{\phi'})\right\} = L_T^{\theta}, \tag{94}$$

and thus we have the desired lower bound on L_T^{θ}.

Finally, observe that from (75) and (79) it follows that

$$L_T^{\theta}(\bar{p}) \leq L_T^{\theta}(\bar{p}, \phi) = \sum_{i=1}^{n} C_i((\bar{w})_i^{\phi}, p_i) = \sum_{s=1}^{n} a_s. \tag{95}$$

Therefore, for a given problem instance, the greedy algorithm produces a schedule with total weighted completion time that does not exceed the optimal by more than a factor of

$$\frac{L_T^{\theta}(\bar{p})}{L_T^{\theta}} \leq \frac{\sum_{i=1}^{n} a_i}{\min_{\pi}\{F(\bar{a}, \bar{b}, \pi)\}}, \tag{96}$$

where $\bar{a}$ and $\bar{b}$ are vectors corresponding to the problem instance as described above. We have proved the following result.

Lemma 3.3.3 *The approximation factor ρ of the greedy algorithm satisfies $\rho \leq \omega_G$, where ω_G is the value of the following optimization problem:*

$$\begin{aligned} \omega_G := \quad & maximize_{\bar{a},\bar{b},\pi} \quad \frac{\sum a_i}{F(\bar{a},\bar{b},\pi)} \\ & subject\ to \qquad b_i > 0 \quad for\ all\ i \in [n] \\ & \qquad\qquad\ \ a_i > 0 \quad for\ all\ i \in [n] \end{aligned} \tag{97}$$

Now our goal is to solve the maximization problem (97). According to the following lemma, the optimal value of (97) is achieved when the largest ratio a_i/b_i is a_n/b_n.

Lemma 3.3.4

$$\begin{array}{lll} \omega_G = & maximize_{\bar{a},\bar{b},\pi} & \frac{\sum a_i}{F(\bar{a},\bar{b},\pi)} \\ & subject\ to & \frac{a_n}{b_n} \geq \frac{a_i}{b_i} \quad for\ all\ i \in [n-1] \\ & & b_i > 0 \quad for\ all\ i \in [n] \\ & & a_i > 0 \quad for\ all\ i \in [n] \end{array} \tag{98}$$

Proof: We will show that given any permutation π and any pair of vectors $\bar{a}$ and $\bar{b}$ of length n satisfying the constraints of (97), they can be transformed so that they satisfy the constraints of (98), without reducing the ratio $(\sum a_i)/F(\bar{a},\bar{b},\pi)$. Let $\|\bar{a},\bar{b}\| = \max_i\{a_i/b_i\}$ be the largest ratio a_i/b_i, and let $v = \max\{i : a_i/b_i = \|\bar{a},\bar{b}\|\}$ be the largest index for which that ratio is achieved. The transformation will proceed in steps, with each step reducing the quantity $n - v$, i.e., moving the largest ratio a_i/b_i closer to the end of the vectors.

In each step there are two cases. If

$$\frac{\sum_{i=1}^{v} a_i}{\sum_{i=1}^{v} f_i(\bar{a},\bar{b},\pi)} \geq \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} f_i(\bar{a},\bar{b},\pi)}, \tag{99}$$

then truncate $\bar{a}$ and $\bar{b}$ to length v. In particular, let $\bar{a}'$ and $\bar{b}'$ be vectors of length v, where $a'_i = a_i$ and $b'_i = b_i$ for all $i \in [v]$. Also, restrict the permutation π on $[n]$ to a permutation π' on $[v]$ by taking

$$\pi'(i) = |\{j \leq v : \pi(j) \leq \pi(i)\}| \tag{100}$$

for all $i \in [v]$. Note that this construction has the property that $\pi(i) < \pi(k) \Rightarrow \pi'(i) < \pi'(k)$. Then for all $i \in [v]$, we have

$$(\bar{b})_i^{\pi} = \sum_{k:\pi(k)\leq\pi(i)} b_k \geq \sum_{k\leq v \wedge \pi(k)\leq\pi(i)} b_k = \sum_{k:\pi'(k)\leq\pi'(i)} b'_k = (\bar{b}')_i^{\pi'}. \tag{101}$$

Observe that for all $l > v$ we have $a_l/b_l < a_v/b_v$, and also $(\bar{b})_l > (\bar{b})_v$, and therefore

$$\begin{aligned} f_i(\bar{a}', \bar{b}', \pi') &= \max_{i \le l \le v} \left\{ \frac{b'_i}{b'_l} \left\langle \frac{(\bar{b}')^{\pi'}_i}{(\bar{b}')_l} \right\rangle a'_l \right\} \\ &\le \max_{i \le l \le v} \left\{ \frac{b_i}{b_l} \left\langle \frac{(\bar{b})^{\pi}_i}{(\bar{b})_l} \right\rangle a_l \right\} \\ &= \max_{i \le l \le n} \left\{ \frac{b_i}{b_l} \left\langle \frac{(\bar{b})^{\pi}_i}{(\bar{b})_l} \right\rangle a_l \right\} \\ &= f_i(\bar{a}, \bar{b}, \pi). \end{aligned} \tag{102}$$

We conclude that

$$\frac{\sum_{i=1}^{v} a'_i}{\sum_{i=1}^{v} f_i(\bar{a}', \bar{b}', \pi')} \ge \frac{\sum_{i=1}^{v} a_i}{\sum_{i=1}^{v} f_i(\bar{a}, \bar{b}, \pi)} \ge \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} f_i(\bar{a}, \bar{b}, \pi)}. \tag{103}$$

Furthermore, $\|\bar{a}', \bar{b}'\| = a_v/b_v$, so the transformation is complete.

Suppose instead that

$$\frac{\sum_{i=1}^{v} a_i}{\sum_{i=1}^{v} f_i(\bar{a}, \bar{b}, \pi)} < \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} f_i(\bar{a}, \bar{b}, \pi)}. \tag{104}$$

Let $y = \max_{i>v}\{a_i/b_i\}$ be the largest ratio that comes after a_v/b_v. Note that $y < \|\bar{a}, \bar{b}\|$. Let $\bar{a}'$ be a vector of length n with $a'_i = \frac{y}{\|\bar{a}, \bar{b}\|} a_i$ for $i \le v$, and $a'_i = a_i$ for $i > v$. Now $\|\bar{a}', \bar{b}\| = y$, and $\max\{i : a'_i/b_i = \|\bar{a}', \bar{b}\|\} > v$, so the largest ratio a'_i/b_i is now closer to the end of the vectors. Since the new solution $\bar{a}', \bar{b}, \pi$ is feasible for (97), it only remains to check that its objective function is no less than that of the original solution.

Observe that for $i > v$, we have $f_i(\bar{a}', \bar{b}, \pi) = f_i(\bar{a}, \bar{b}, \pi)$. For $i \le v$, note that for all $l \in [n]$ we have $a'_l/b_l \le \|\bar{a}', \bar{b}\| = a'_v/b_v$, and if $l > v$, then we also have $(\bar{b})_l > (\bar{b})_v$. Therefore

$$\begin{aligned} f_i(\bar{a}', \bar{b}, \pi) &= \max_{i \le l \le n} \left\{ \frac{b_i}{b_l} \left\langle \frac{(\bar{b})^{\pi}_i}{(\bar{b})_l} \right\rangle a'_l \right\} \\ &= \max_{i \le l \le v} \left\{ \frac{b_i}{b_l} \left\langle \frac{(\bar{b})^{\pi}_i}{(\bar{b})_l} \right\rangle a'_l \right\} \end{aligned}$$

$$
\begin{aligned}
&= \frac{y}{\|\bar{a},\bar{b}\|} \cdot \max_{i \le l \le v} \left\{ \frac{b_i}{b_l} \left\langle \frac{(\bar{b})_i^{\pi}}{(\bar{b})_l} \right\rangle a_l \right\} \\
&= \frac{y}{\|\bar{a},\bar{b}\|} \cdot \max_{i \le l \le n} \left\{ \frac{b_i}{b_l} \left\langle \frac{(\bar{b})_i^{\pi}}{(\bar{b})_l} \right\rangle a_l \right\} \\
&= \frac{y}{\|\bar{a},\bar{b}\|} f_i(\bar{a},\bar{b},\pi). \qquad (105)
\end{aligned}
$$

Now we have

$$
\begin{aligned}
\frac{\sum_{i=1}^{n} a_i'}{\sum_{i=1}^{n} f_i(\bar{a}',\bar{b},\pi)} &= \frac{\sum_{i=1}^{v} a_i' + \sum_{i=v+1}^{n} a_i'}{\sum_{i=1}^{v} f_i(\bar{a}',\bar{b},\pi) + \sum_{i=v+1}^{n} f_i(\bar{a}',\bar{b},\pi)} \\
&= \frac{\frac{y}{\|\bar{a},\bar{b}\|} \sum_{i=1}^{v} a_i + \sum_{i=v+1}^{n} a_i}{\frac{y}{\|\bar{a},\bar{b}\|} \sum_{i=1}^{v} f_i(\bar{a},\bar{b},\pi) + \sum_{i=v+1}^{n} f_i(\bar{a},\bar{b},\pi)} \\
&> \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} f_i(\bar{a},\bar{b},\pi)}, \qquad (106)
\end{aligned}
$$

where the last inequality follows from (104) and the fact that $y < \|\bar{a},\bar{b}\|$.

This process can be repeated until we have a solution that satisfies the constraints of (98). □

We can make use of Lemma 3.3.4 to prove the desired upper bound on ω_G.

Lemma 3.3.5 $\omega_G \le 2$.

Proof: Our goal is to show that $2F(\bar{a},\bar{b},\pi) \ge \sum_{i=1}^{n} a_i$ given the constraints of (98).

$$
\begin{aligned}
2F(\bar{a},\bar{b},\pi) &= 2\sum_{i=1}^{n} f_i(\bar{a},\bar{b},\pi) \\
&= 2\sum_{i=1}^{n} \max_{l \ge i} \left\{ \frac{b_i}{b_l} \left\langle \frac{(\bar{b})_i^{\pi}}{(\bar{b})_l} \right\rangle a_l \right\} \\
&\ge 2\sum_{i=1}^{n} \frac{b_i}{b_n} \frac{(\bar{b})_i^{\pi}}{(\bar{b})_n} a_n \\
&= 2\frac{a_n}{b_n(\bar{b})_n} \sum_{i=1}^{n} b_i (\bar{b})_i^{\pi} \\
&= 2\frac{a_n}{b_n(\bar{b})_n} \sum_{i=1}^{n} b_i \sum_{k:\pi(k) \le \pi(i)} b_k
\end{aligned}
$$

$$
\begin{aligned}
&= 2\frac{a_n}{b_n(\bar{b})_n} \sum_{\pi(k)\leq\pi(i)} b_i b_k \\
&= 2\frac{a_n}{b_n(\bar{b})_n} \sum_{r\leq s} b_r b_s \\
&= \frac{a_n}{b_n \sum_{i=1}^{n} b_i} \left[\left(\sum_{i=1}^{n} b_i \right)^2 + \sum_{i=1}^{n} b_i^2 \right] \\
&> \frac{a_n}{b_n} \sum_{i=1}^{n} b_i \\
&= \sum_{i=1}^{n} b_i \frac{a_n}{b_n} \\
&\geq \sum_{i=1}^{n} b_i \frac{a_i}{b_i} \\
&= \sum_{i=1}^{n} a_i. \qquad (107)
\end{aligned}
$$

□

Now it follows directly from Lemmas 3.3.3 and 3.3.5 that the greedy algorithm is 2-approximate. To show that its running time is $O(n^2 + mn)$, we make use of Lemma 3.2.1. We begin by computing v_{ij} for all i and j in $O(mn)$ steps. Then for any task i, the time required to compute $C_i(\mu_s)$ over all iterations s of the greedy algorithm is $O(n + m)$. This yields a total of $O(n^2 + mn)$ steps to compute the values of $C_i(\mu_s)$. Finally, it takes $O(n)$ steps in each iteration to choose the task with the smallest value of $C_i(\mu_s)/w_i$, for a total of $O(n^2)$ steps. Thus the greedy algorithm has running time $O(n^2 + mn)$. This completes the proof of Theorem 3.3.1.

3.4 A Faster Approximation Algorithm for the Unweighted Case

If $M_i = [m]$ and $w_i = 1$ for all tasks i, and the tasks satisfy the nondecreasing work and nonincreasing execution time conditions, then we can obtain an even faster allotment selection algorithm using the following observation. Suppose that $n > m$. Since the tasks do not exhibit superlinear speedup, intuition suggests that smaller tasks should be executed first and assigned one processor each. This will yield high efficiency, which will in turn keep task completion times down. So if task i is in position s, and s is sufficiently large, then we expect that $p_i = 1$ is a good allotment for task i. According to the following lemma, this intuition can be extended to any position s, so that a good allotment of processors to the task in position s can be determined independently of that task's execution time function.

Lemma 3.4.1 *Suppose that $M_i = [m]$ and $w_i = 1$ for all tasks i and that the tasks satisfy the nondecreasing work and nonincreasing execution time conditions. Let $p(s) = \left\lceil \frac{(1-\frac{1}{2}\theta)m}{\theta s - \frac{3}{2}\theta + 2} \right\rceil$. Then $C_i(s, p(s)) \leq 2C_i(s)$ for all tasks $i \in [n]$ and all positions $s \in [n]$.*

Proof: Consider assigning task i to position s and assigning it j processors. Then its contribution is

$$C_i(s,j) = \left\lceil \frac{\theta j}{m}(s - \frac{1}{2}) + 1 - \frac{1}{2}\theta \right\rceil t_i(j) \tag{108}$$

$$= \left\lceil \frac{\theta}{m}(s - \frac{1}{2}) + \frac{1}{j}(1 - \frac{1}{2}\theta) \right\rceil j t_i(j). \tag{109}$$

Let us consider the two cases $j \leq \kappa_i(s)$ and $j \geq \kappa_i(s)$, and in each case determine by how much $C_i(s,j)$ can exceed $C_i(s) = C_i(s, \kappa_i(s))$.

Suppose that $j \leq \kappa_i(s)$. Then we have $j \cdot t_i(j) \leq \kappa_i(s) \cdot t_i(\kappa_i(s))$ due to the nondecreasing work condition. Therefore

$$\begin{aligned} C_i(s) &= C_i(s, \kappa_i(s)) \\ &= \left[\frac{\theta}{m}(s - \frac{1}{2}) + \frac{1}{\kappa_i(s)}(1 - \frac{1}{2}\theta)\right] \kappa_i(s) t_i(\kappa_i(s)) \\ &\geq \left[\frac{\theta}{m}(s - \frac{1}{2}) + \frac{1}{\kappa_i(s)}(1 - \frac{1}{2}\theta)\right] j t_i(j) \\ &\geq \frac{1}{m}(\theta s - \theta + 1) j t_i(j). \end{aligned} \tag{110}$$

It follows from (109) and (110) that

$$\begin{aligned} C_i(s, j) &\leq \frac{\frac{\theta}{m}(s - \frac{1}{2}) + \frac{1}{j}(1 - \frac{1}{2}\theta)}{\frac{1}{m}(\theta s - \theta + 1)} C_i(s) \\ &= \frac{\theta(s - \frac{1}{2})j + (1 - \frac{1}{2}\theta)m}{(\theta s - \theta + 1)j} C_i(s). \end{aligned} \tag{111}$$

Now suppose that $j \geq \kappa_i(s)$. Then we have $t_i(j) \leq t_i(\kappa_i(s))$ due to the nonincreasing execution time condition. Therefore

$$\begin{aligned} C_i(s) &= C_i(s, \kappa_i(s)) \\ &= \left[\frac{\theta \kappa_i(s)}{m}(s - \frac{1}{2}) + 1 - \frac{1}{2}\theta\right] t_i(\kappa_i(s)) \\ &\geq \left[\frac{\theta \kappa_i(s)}{m}(s - \frac{1}{2}) + 1 - \frac{1}{2}\theta\right] t_i(j) \\ &\geq \left[\frac{\theta}{m}(s - \frac{1}{2}) + 1 - \frac{1}{2}\theta\right] t_i(j). \end{aligned} \tag{112}$$

It follows from (108) and (112) that

$$\begin{aligned} C_i(s, j) &\leq \frac{\frac{\theta j}{m}(s - \frac{1}{2}) + 1 - \frac{1}{2}\theta}{\frac{\theta}{m}(s - \frac{1}{2}) + 1 - \frac{1}{2}\theta} C_i(s) \\ &= \frac{\theta(s - \frac{1}{2})j + (1 - \frac{1}{2}\theta)m}{\theta(s - \frac{1}{2}) + (1 - \frac{1}{2}\theta)m} C_i(s). \end{aligned} \tag{113}$$

Let

$$g_1(x) = \frac{\theta(s - \frac{1}{2})x + (1 - \frac{1}{2}\theta)m}{(\theta s - \theta + 1)x}, \tag{114}$$

and let

$$g_2(x) = \frac{\theta(s - \frac{1}{2})x + (1 - \frac{1}{2}\theta)m}{\theta(s - \frac{1}{2}) + (1 - \frac{1}{2}\theta)m}. \tag{115}$$

Then from (111) and (113) we have

$$C_i(s, j) \leq \max\{g_1(j), g_2(j)\} \cdot C_i(s). \tag{116}$$

Now $p(s) = \lceil x_0 \rceil$, where

$$x_0 = \frac{(1 - \frac{1}{2}\theta)m}{\theta s - \frac{3}{2}\theta + 2}. \tag{117}$$

Since $g_1(x)$ is decreasing, $g_1(\lceil x \rceil) \leq g_1(x)$. And since $g_2(x)$ is increasing, $g_2(\lceil x \rceil) \leq g_2(x + 1)$. Therefore,

$$\begin{aligned} \max\{g_1(p(s)), g_2(p(s))\} &= \max\{g_1(\lceil x_0 \rceil), g_2(\lceil x_0 \rceil)\} \\ &\leq \max\{g_1(x_0), g_2(x_0 + 1)\}. \end{aligned} \tag{118}$$

A straightforward calculation reveals that $\max\{g_1(x_0), g_2(x_0 + 1)\} \leq 2$, and therefore

$$\max\{g_1(p(s)), g_2(p(s))\} \leq 2. \tag{119}$$

Then from (116) and (119), we have

$$C_i(s, p(s)) \leq \max\{g_1(p(s)), g_2(p(s))\} \cdot C_i(s) \leq 2C_i(s). \tag{120}$$

□

The algorithm is as follows. Let $s := n$. Let $s' := \max\{r : p(r) > p(s)\}$. Note that $p(s' + 1) = p(s' + 2) = \cdots = p(s)$. From among those tasks that have not yet been assigned to positions, choose the $s - s'$ tasks with the smallest values of $t_i(p(s))$ to be in positions s down to $s' + 1$, assigning $p(s)$ processors to each one. Then let $s := s'$ and proceed in the same way until every task has been assigned to a position. Note that this algorithm determines an order ψ and an allotment $\bar{p}$ such that $p_i = p(\psi(i))$.

We say that this algorithm is 'semi-oblivious' because it uses the task execution time functions to determine the order of the tasks, but once the order is determined, the allotment of processors to tasks is determined independently of the task execution time functions.

Theorem 3.4.1 *For any set T of malleable tasks satisfying the nondecreasing work and nonincreasing execution time conditions, with $M_i = [m]$ and $w_i = 1$ for all tasks i, and any $\theta \in [0,1]$, the semi-oblivious algorithm finds an allotment $\bar{p}$ that satisfies $L_T^\theta(\bar{p}) \leq 4L_T^\theta$. Its running time is $O(\min\{n + m\log m, n^2\})$.*

Corollary 3.4.1 *Suppose we are given an NMACT algorithm $\mathcal{A}$ with running time $Q(m,n)$ such that $R_{\mathcal{A}}(T(\bar{q})) \leq \lambda_A A_T(\bar{q}) + \lambda_H H_T(\bar{q}) + \lambda_U U_T(\bar{q})$ for all nonmalleable task sets $T(\bar{q})$ satisfying $\|\bar{q}\| \leq \lceil \alpha m \rceil$. Then there is a MACT algorithm with approximation factor $4 \cdot \max\{\lambda_A, \frac{1}{2}\lambda_A + \lambda_H, \lambda_A + (\frac{1}{\alpha} - 1 + \frac{2}{m+1})\lambda_H + \lambda_U, \lambda_A + \lambda_H + \lambda_U\}$ and running time $O(\min\{n + m\log m, n^2\}) + Q(m,n)$ for task sets that satisfy the nondecreasing work and nonincreasing execution time conditions, and with $M_i = [m]$ for all tasks i.*

Proof: First, use the semi-oblivious algorithm to select an allotment $\bar{p}$. In order to apply Theorem 2.3.1, we wish to select β as small as possible subject to $\lceil \beta m \rceil \geq \|\bar{p}\|$ and $\lceil \beta m \rceil \geq \lceil \alpha m \rceil$. Observe that

$$\|\bar{p}\| = \left\lceil \frac{(1 - \frac{1}{2}\theta)m}{2 - \frac{1}{2}\theta} \right\rceil \leq \left\lceil \frac{m}{2} \right\rceil, \tag{121}$$

since $(1 - \frac{1}{2}\theta)/(2 - \frac{1}{2}\theta) \leq \frac{1}{2}$ for all $\theta \in [0,1]$. Now applying Theorem 2.3.1 with $\beta = \max\{\frac{1}{2}, \alpha\}$ and $\rho = 4$ yields the desired approximation factor. □

Using the semi-oblivious algorithm in conjunction with LRF yields the following result.

Corollary 3.4.2 *There is a MACT algorithm with approximation factor* $3 + \sqrt{17} \approx 7.124$ *and running time* $O(\min\{n \log n + m \log m, n^2\})$ *for task sets that satisfy the nondecreasing work and nonincreasing execution time conditions, and with* $M_i = [m]$ *for all tasks* i.

Proof: Corollary 2.3.1 with $\rho = 4$ gives

$$R_{\mathrm{LRF}}(T(\bar{q})) \leq 4 \cdot \max\left\{\frac{1}{1-\alpha}, \frac{1}{2(1-\alpha)} + 1, m\left(\frac{1}{\lceil \alpha m \rceil} - \frac{1}{\lceil \beta m \rceil}\right) + 1\right\} \cdot R^*(T). \quad (122)$$

Let

$$\beta = \frac{1 - \frac{1}{2}\theta}{2 - \frac{1}{2}\theta}, \quad (123)$$

and then $\|\bar{p}\| \leq \lceil \beta m \rceil$, where $\bar{p}$ is an allotment produced by the semi-oblivious algorithm. Now choosing $\alpha = \beta$ yields

$$\begin{aligned} R_{\mathrm{LRF}}(T(\bar{q})) &= 4 \cdot \max\{2 - \frac{1}{2}\theta, 2 - \frac{1}{4}\theta, 1\} \cdot R^*(T) \\ &= (8 - \theta)R^*(T), \end{aligned} \quad (124)$$

provided that we choose θ such that $\theta = 1/(\frac{3}{2} - \alpha) = (2 - \frac{1}{2}\theta)/(2 - \frac{1}{4}\theta)$, as required by Corollary 2.3.1. So choosing $\theta = 5 - \sqrt{17}$ yields

$$R_{\mathrm{LRF}}(T(\bar{p})) \leq (3 + \sqrt{17})R^*(T) \approx 7.124R^*(T). \quad (125)$$

□

The analysis of the semi-oblivious algorithm is similar to that of the greedy algorithm. We need an upper bound on $L_T^\theta(\bar{p})/L_T^\theta$, where $\bar{p}$ is the semi-oblivious allotment. We will obtain a lower bound on L_T^θ by considering how much the objective function value $\sum_{i=1}^n C_i(\psi(i), p(\psi(i)))$ can be reduced by changing the order of the tasks from the semi-oblivious order ψ to some other order ψ' and assigning to each task the optimal number of processors for its position.

The following lemma gives lower bounds on the contribution of the task in position s if it is moved to position r and assigned the optimal number of processors.

Lemma 3.4.2 *Let σ and σ' be any permutations on $[n]$. Let $s = \sigma(i)$, and let $r = \sigma'(i)$. Then*

1. *if $r \leq s$, then $C_i(r) \geq \frac{1}{2}\frac{r}{s}C_i(s, p(s))$;*
2. *if $r \geq s$, then $C_i(r) \geq \frac{1}{2}C_i(s, p(s))$.*

Proof: Immediate from Lemmas 3.3.1 and 3.4.1. □

Note that although the semi-oblivious algorithm does not distinguish between positions that have the same value of $p(s)$, for the purpose of the analysis we will say that among such positions, the tasks are ordered so that the task with the least execution time is assigned to the highest-numbered position.

Lemma 3.4.3 *Let ψ be the permutation produced by the semi-oblivious algorithm, and let ψ' be any permutation. Then $C_i(\psi'(i)) \geq \frac{1}{2}\left\langle \frac{\psi'(i)}{\psi(k)} \right\rangle C_k(\psi(k), p(\psi(k)))$ for all k such that $\psi(k) \geq \psi(i)$.*

Proof: By Lemma 3.4.2, we have $C_i(\psi'(i)) \geq \frac{1}{2}\left\langle \frac{\psi'(i)}{\psi(k)} \right\rangle C_i(\psi(k), p(\psi(k)))$. Since $\psi(k) \geq \psi(i)$, and because of the way the semi-oblivious algorithm assigns tasks to positions, we have $C_i(\psi(k), p(\psi(k))) \geq C_k(\psi(k), p(\psi(k)))$. The result follows directly. □

Now to obtain a lower bound on L_T^θ, let $a_{\psi(i)} = C_i(\psi(i), p(\psi(i)))$ be the contribution of task i in the semi-oblivious order ψ. Let π be a permutation such that $\pi(\psi(i)) = \psi'(i)$ for all $i \in [n]$. Let $\bar{1}$ denote the all-ones vector. Then from (91) we have

$$f_s(\bar{a}, \bar{1}, \pi) = \max_{l \geq s} \left\{ \left\langle \frac{\pi(s)}{l} \right\rangle a_l \right\}. \tag{126}$$

Then it follows from Lemma 3.4.3 that

$$\begin{aligned} C_i(\psi'(i)) &\geq \frac{1}{2} \cdot \max_{k:\psi(k)\geq\psi(i)} \left\{ \left\langle \frac{\psi'(i)}{\psi(k)} \right\rangle C_k(\psi(k), p(\psi(k))) \right\} \\ &= \frac{1}{2} \cdot \max_{k:\psi(k)\geq\psi(i)} \left\{ \left\langle \frac{\pi(\psi(i))}{\psi(k)} \right\rangle a_{\psi(k)} \right\} \\ &= \frac{1}{2} \cdot \max_{l\geq\psi(i)} \left\{ \left\langle \frac{\pi(\psi(i))}{l} \right\rangle a_l \right\} \\ &= \frac{1}{2} f_{\psi(i)}(\bar{a}, \bar{1}, \pi). \end{aligned} \tag{127}$$

Now we have the desired lower bound:

$$\min_{\pi}\{\frac{1}{2}F(\bar{a}, \bar{1}, \pi)\} \leq \min_{\psi'} \left\{ \sum_{i=1}^{n} C_i(\psi'(i)) \right\} = L_T^{\theta}. \tag{128}$$

For the final step, observe that

$$L_T^{\theta}(\bar{p}) \leq L_T^{\theta}(\bar{p}, \psi) = \sum_{i=1}^{n} C_i(\psi(i), p_i) = \sum_{i=1}^{n} C_i(\psi(i), p(\psi(i))) = \sum_{s=1}^{n} a_s. \tag{129}$$

Now we conclude that

$$\frac{L_T^{\theta}(\bar{p})}{L_T^{\theta}} \leq \frac{2\sum_{i=1}^{n} a_i}{\min_{\pi}\{F(\bar{a}, \bar{1}, \pi)\}}. \tag{130}$$

Thus for the semi-oblivious algorithm we have the following version of Lemma 3.3.3.

Lemma 3.4.4 *The approximation factor ρ of the semi-oblivious algorithm satisfies $\rho \leq \omega_S$, where ω_S is the value of the following optimization problem:*

$$\begin{aligned} \omega_S := \quad & \textit{maximize}_{\bar{a},\pi} \quad \frac{2\sum a_i}{F(\bar{a}, \bar{1}, \pi)} \\ & \textit{subject to} \quad a_i > 0 \quad \textit{for all } i \in [n] \end{aligned} \tag{131}$$

Observe that the optimization problem (131) is a restriction of (97) with $\bar{b} = \bar{1}$, and with an extra factor of 2 in the objective function. Therefore, $\omega_S \leq 2\omega_G$. Now it follows directly from Lemmas 3.4.4 and 3.3.5 that $\omega_S \leq 4$, and thus that the semi-oblivious algorithm is 4-approximate.

It only remains to show that its running time is $O(\min\{n + m \log m, n^2\})$. In each iteration, we will use a linear time selection algorithm as in [1] to choose the $s - s'$ tasks with the smallest values of $t_i(p(s))$. Observe that there are $O(m/j)$ positions s for which $p(s) > j$. Thus it takes $O(n)$ steps to choose the tasks that will be assigned a single processor, and $O(m/j)$ steps to choose the tasks that will be assigned $j + 1$ processors, where $j \geq 1$. This gives a total of $O(n + \sum_{j=1}^{m-1} \frac{m}{j}) = O(n + m \log m)$ steps. But also note that at least one task is assigned to a position in each iteration, and therefore the running time is $O(n^2)$. This completes the proof of Theorem 3.4.1.

3.5 An Alternate Scheduling Environment

We will now take another view of allotment selection by considering a different scheduling environment. Suppose that tasks are arriving over time, that the existence of a task is unknown until its arrival, and that little or no information about task execution times is available. However, it is assumed that the *available parallelism* N_i of each task is known, where N_i is the number of processors that the scheduler believes task i can use productively. (This may be the number of processors requested by the user upon submitting the task.) The goal remains the same: schedule the tasks in a way that minimizes (unweighted) mean response time, where a task's response time is the difference between its completion time and its arrival time. We will show how to adapt the semi-oblivious algorithm to a processor allocation policy for this environment, and use simulations to compare its performance to that of policies that have been proposed in the literature.

3.5.1 The Policies

The semi-oblivious algorithm can be adapted to a processor allocation policy for the environment described above in the following way. Tasks are scheduled in first-come-first-served order. Let s denote the number of tasks currently waiting to be executed. Then using $\theta = 1$, allocate $\min\{p(s), N_i\} = \min\{\lceil m/(2s+1) \rceil, N_i\}$ processors to the next task in line. If the number of free processors is less than the number of processors required by the next task, then those processors remain idle until the situation changes.

The resulting policy, which we call SO, belongs to the class of *run-to-completion* (RTC) policies — nonpreemptive policies that do not change the allocation of processors to a task during that task's execution. Chiang, Mansharamani, and Vernon [12] have recently compared RTC policies that have been proposed in the literature, and concluded that the policies they call ASP-*max* and ASP-*max*+ appear to have the best performance. Therefore, we will follow the experimental methods of Chiang et al. and compare the performance of SO with that of ASP-*max* and ASP-*max*+.

ASP-*max* is based on ASP (adaptive static partitioning), a policy introduced by Setia and Tripathi [47]. ASP works in the following way. Whenever processors are free and there are tasks ready to be executed, divide the processors as equally as possible among the tasks, subject to the constraint that no task is assigned more processors than its available parallelism. ASP-*max* is the same as ASP, except that there is an additional parameter *max*, and no task can be assigned more than *max* processors. ASP-*max* is essentially identical to EPM, which was introduced by Rosti, Smirni, Dowdy, Serazzi, and Carlson [45]. ASP-*max*+ is the same as ASP-*max*, except that instead of using *max* as an upper bound on the number of processors allocated to each task, it uses a function ν_i of *max* and the system load ρ that is computed using a

formula analogous to the processor allocation formula given by Sevcik [49]:

$$\nu_i = \begin{cases} \frac{\rho}{\pi} max + \left(1 - \frac{\rho}{\pi}\right) N_i & \text{if } \rho < \pi \\ \frac{1-\rho}{1-\pi} max + \left(1 - \frac{1-\rho}{1-\pi}\right) \cdot 1 & \text{if } \rho \geq \pi, \end{cases} \tag{132}$$

where $\pi = .25$.

3.5.2 The Model

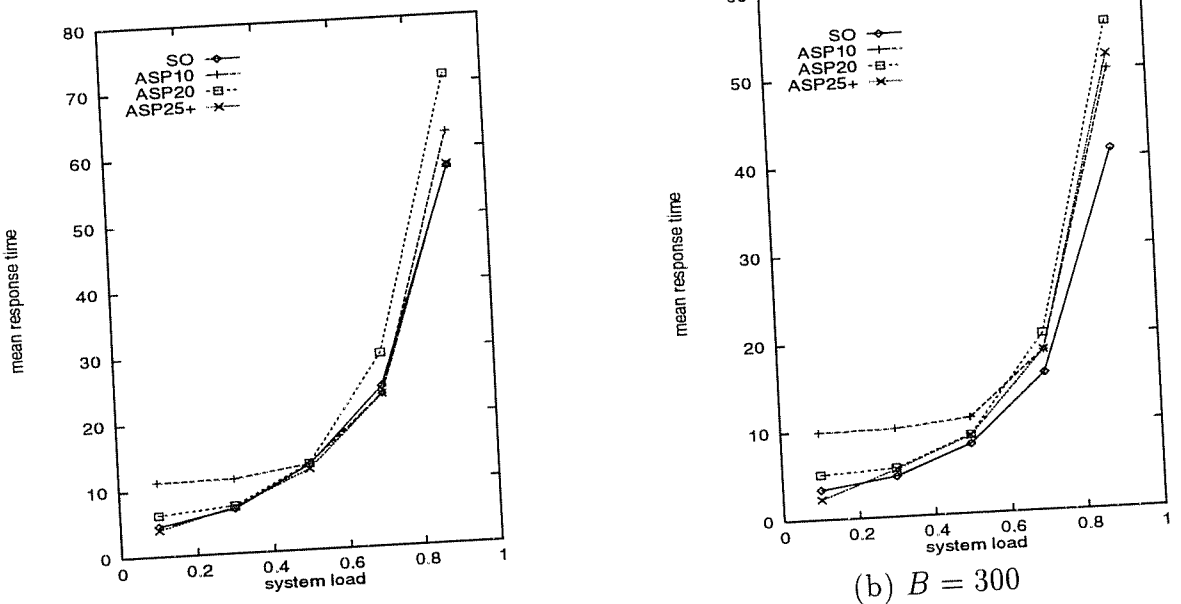
For our simulations, we adopt the system and workload model used by Chiang et al. [12], which is due to Mansharamani and Vernon [39]. Tasks arrive according to a Poisson process with rate λ. We have m identical processors on which to schedule the tasks, and there is no scheduling overhead.

Each task is characterized by its *demand* $D_i = t_i(1)$, its available parallelism N_i, and its *execution rate function* $E_i : [m] \rightarrow \mathbb{R}$. The execution time function of a task in terms of E_i and D_i is given by $t_i(j) = D_i / E_i(j)$. The experiments we present here all use $N_i = m$, but letting N_i be a random variable yields similar results. For task demand, we use a two-stage hyperexponential distribution with coefficient of variation $C_D = 5$, following Majumdar, Eager, and Bunt [38]. We choose a distribution with mean $\bar{D} = m$, so that the system load will be given by $\rho = \lambda \bar{D}/m = \lambda$. We use an execution rate function derived from that of Dowdy [17]. It is the same for every task, and is given by

$$E_i(j) = E(j) = \frac{(B+1)j}{B+j}. \tag{133}$$

We refer to B as the *sublinearity parameter*. Note that the function $E(j)$ more nearly reflects linear speedup as B increases.

In our use of this system and workload model we choose to vary only three parameters: m, λ, and B. We use default values for these parameters of $m = 100$, $\lambda = .7$,



(a) $B = 70$ (b) $B = 300$

Figure 2: Comparison of policies under varying system load
$m = 100$

and $B = 70$ or $B = 300$.

3.5.3 Simulation Results

All the mean response times reported here were obtained using discrete event simulation, with 90% confidence intervals and half-widths usually less than 5%. The confidence intervals were computed using the regenerative method whenever possible. Otherwise, we used the method of batch means.

Figure 2 plots the mean response times of SO and selected ASP-max and ASP-max+ policies versus system load with $m = 100$ and $B = 70$ and 300. We have chosen values of max that yield the best performance for these parameter settings. Observe that SO performs at least as well as the other policies. For $B = 70$, its mean response times are about the same as those of ASP25+, while for $B = 300$, SO has a slight

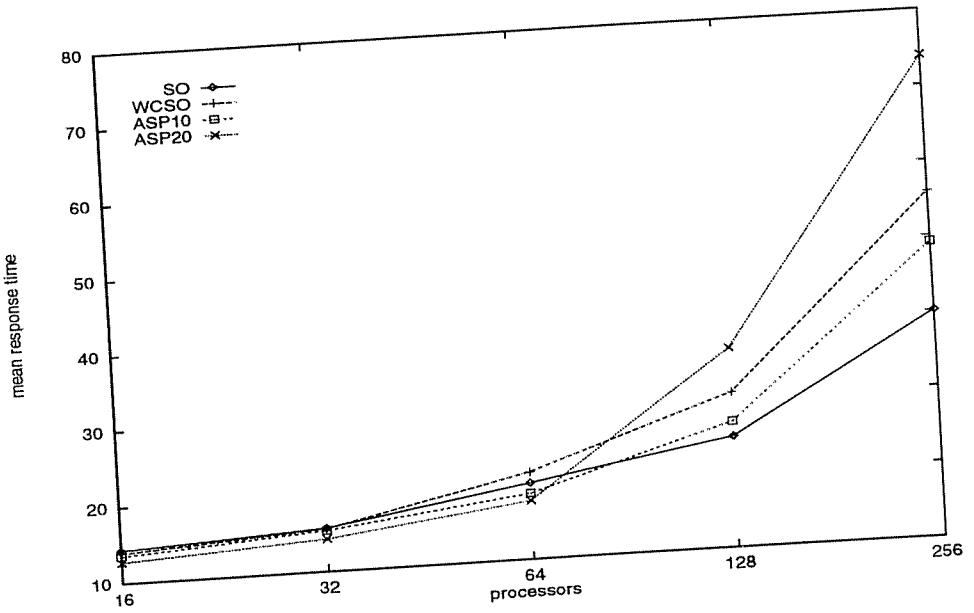


Figure 3: Comparison of policies for varying numbers of processors
$\lambda = .7$; $B = 70$

advantage.

An important difference between SO and the other policies is that SO is not *work-conserving*: it may leave processors idle even though tasks are waiting to be executed. To demonstrate the importance of this difference, we introduce a work-conserving version of SO, called WCSO. WCSO is like SO, except that if a task is ready and fewer than $p(s)$ processors are free, then it allocates to the task all of the free processors. Figure 3 plots mean response time versus number of processors for $\lambda = .7$ and $B = 70$. Note that as m varies, we let the "10" in ASP10 denote 10%, and similarly for ASP20. That is, the upper limit on processor allocation for ASP-max is $(max/100)m$. Also note that when we fix $\lambda = .7$, ASP25+ and ASP10 are identical policies.

Notice how the performance of the work-conserving policies suffers as the number of processors increases. A certain increase in mean response time for all the policies is expected, since task demands are increasing with the number of processors, but B is

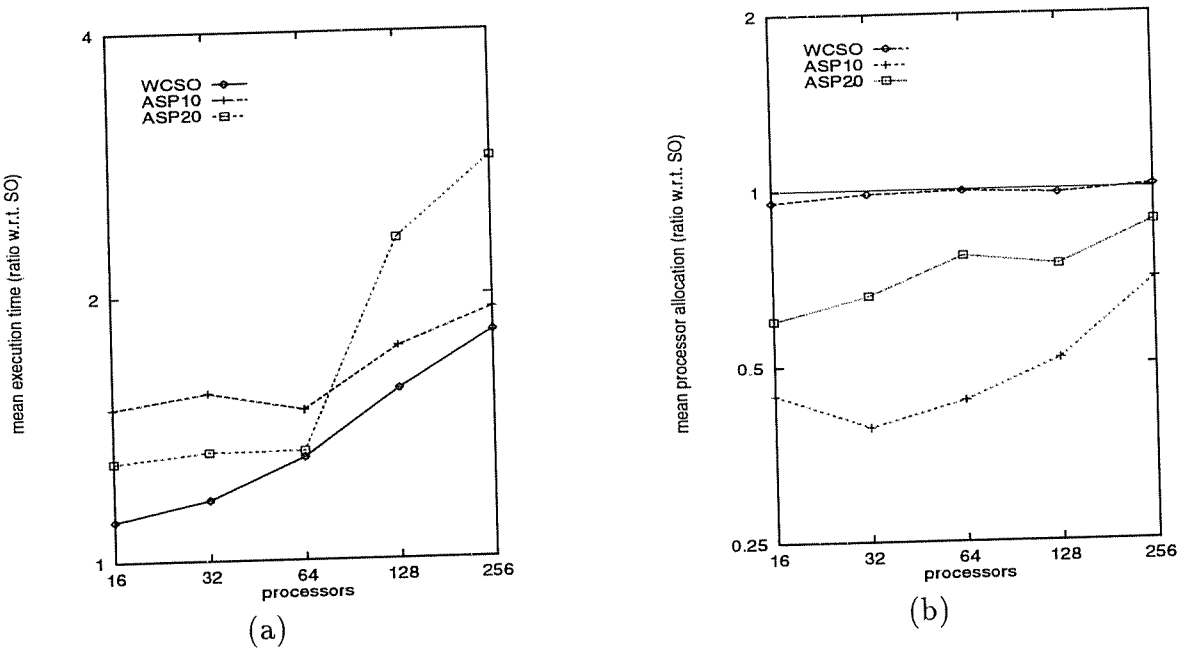


Figure 4: Mean execution time and processor allocation relative to SO
$\lambda = .7$; $B = 70$

not. Therefore, when m is doubled, the tasks are twice as large, but assigning twice as many processors to a task does not double its execution rate. However, observe that the mean response times of the work-conserving policies are increasing with m at a greater rate than those of SO.

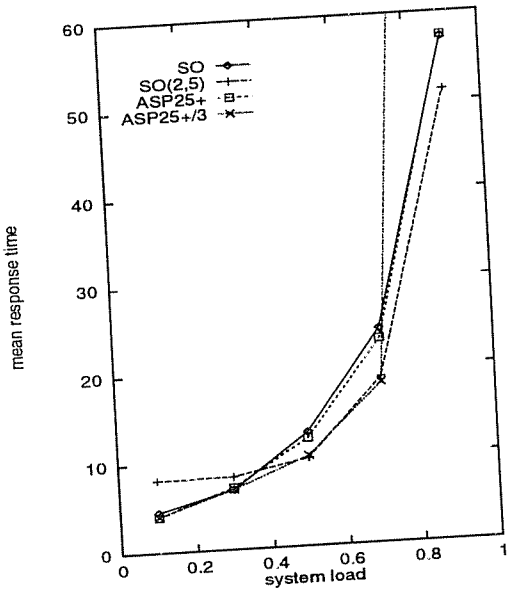
The reason for this is illustrated by Figure 4, which plots the ratio of the mean task execution time of each policy with respect to that of SO, as well as the ratio of the mean processor allocation of each policy with respect to that of SO. For each policy, mean execution time is increasing relative to SO, even though mean allocation is increasing or staying the same relative to SO. The reason for this phenomenon, and the cause of the relatively poor performance of the work-conserving policies, is that they allocate a small number of processors to a relatively large number of tasks. For example, consider the difference between allocating 1 and 2 processors to a task. The execution time of a task is maximized when it is allocated 1 processor, but 2 processors nearly cut

execution time in half. Thus, changing the allocation of a task from 2 processors to 1 has a much more pronounced effect on mean execution time than on mean allocation. Work-conserving policies suffer from this behavior because they are obliged to begin execution of a task even if it can only be allocated a single processor, and they often do so.

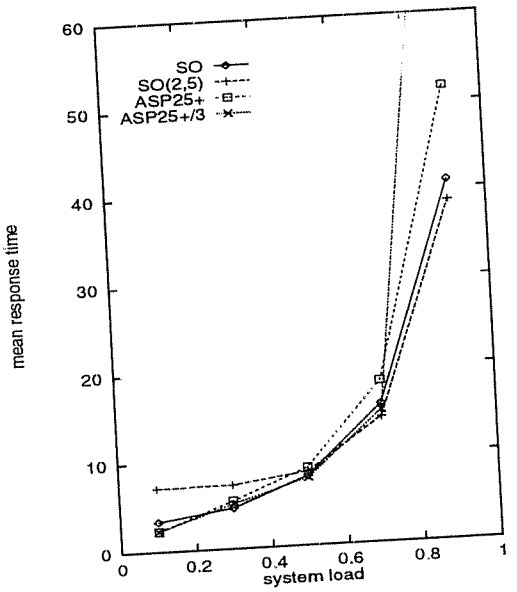
A simple remedy is to incorporate a minimum processor allocation into the ASP-*max* policies. We therefore introduce ASP-*max/min*, which is like ASP-*max* except that it does not allocate fewer than *min* processors to any task. For example, if 10 processors become free and 5 tasks are ready, then ASP10/3 will allocate 4, 3, and 3 processors to the next 3 tasks, while ASP10 will allocate 2 processors to each of the 5 tasks. We write ASP10/3 to denote a minimum allocation of 3 processors for each task, and ASP10//32 to denote a minimum allocation of $m/32$ processors.

We can also improve SO by extending it to a family of policies SO(a_1,a_2) that allocate $\max\{\lfloor m/(a_1 s + a_2)\rfloor, 1\}$ processors to the next task in line when there are s tasks waiting to be executed.

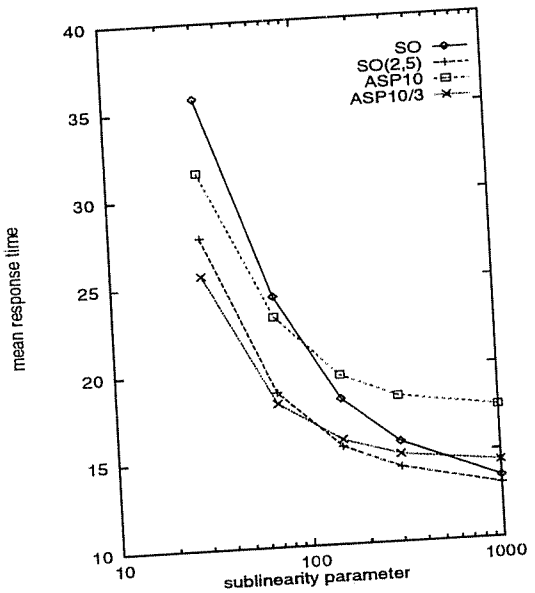
Figure 5 plots mean response times for these new SO(a_1,a_2) and ASP-*max/min* policies as well as for SO and ASP25+, against varying λ, m, and B. In (a) and (b), mean response time is plotted against λ. For light loads, SO(2,5) does not do as well as the other policies, since it tends to allocate fewer processors to each task. For $\lambda = .9$, ASP25+/3 performs poorly because its maximum processor allocation of 4 coupled with a minimum of 3 results in allocating 4 processors to every task, while other policies do better by allocating fewer processors. (This suggests also making the minimum allocation sensitive to system load.) Otherwise, ASP25+/3 and SO(2,5) tend to give the best results.



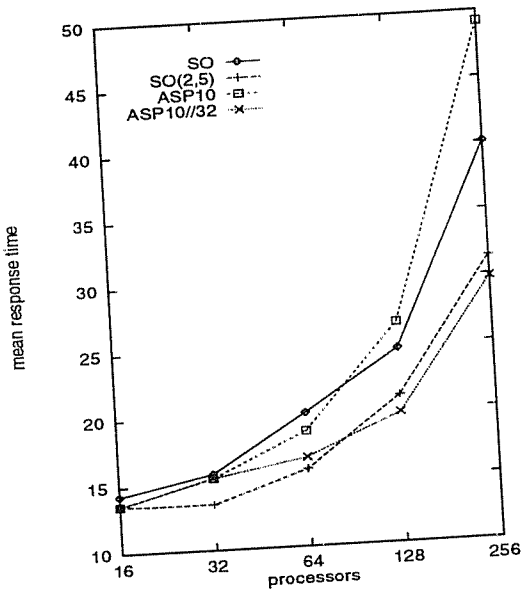
(a) $m = 100$; $B = 70$



(b) $m = 100$; $B = 300$



(c) $\lambda = .7$; $m = 100$



(d) $\lambda = .7$; $B = 70$

Figure 5: Comparison of modified policies

Observe that the mean response times for SO and SO(2,5) are very close for $B = 300$, but for $B = 70$, the mean response times for SO(2,5) are lower. The effect of B on the relative performance of SO and SO(2,5) is seen more clearly in (c), which shows that SO improves relative to SO(2,5) as B increases. This is because SO(2,5) tends to allocate fewer processors per task than SO, which gives it an advantage when B is smaller and the execution rate functions are more sublinear. This also explains why SO and SO(2,5) improve relative to ASP10 and ASP10//3 as B increases, since the former policies tend to allocate more processors per task than the latter policies.

Finally, in (d) we can see that incorporating a minimum processor allocation has corrected the previously identified flaw in ASP10. ASP10//32 and SO(2,5) are the clear winners for large m. The response times for all policies increase as m increases for reasons noted above, but the effects of increasing m are less pronounced for SO(2,5) than for SO, once again due to the fact that the SO(2,5) tends to allocate fewer processors to each task than does SO.

In conclusion, these experiments suggest that SO(a_1,a_2) policies have performance comparable to that of ASP-*max* policies (with or without *min*). Furthermore, SO(a_1,a_2) policies and ASP-*max* policies have three important characteristics in common. First, both are sensitive to system load, allocating fewer processors to each task as system load increases. Various studies show that this is a desirable property [49, 25, 41, 40, 48, 12]. Second, both place an upper limit on the allocation of processors to a task, which has also been shown to be desirable [12]. Third, neither requires any information about task demand or execution rate — information that may not be available. We conclude that SO(a_1,a_2) appears to be a promising family of RTC policies.

Chapter 4

Algorithms for NMACT and NMWACT

Note: The results reported in this chapter were obtained jointly with Schwiegelshohn, Wolf, Turek, and Yu, and are reported in [46].

We have seen that the approximation factor of the LRF algorithm for NMWACT depends on the number of processors required by the tasks. In this chapter, we present algorithms for NMWACT and NMACT and prove an approximation factor for each that is independent of the number of processors required by the tasks. This allows us to couple these algorithms with allotment selection algorithms from Chapter 3 to obtain MACT and MWACT algorithms that place no restrictions on the task execution time functions. In Section 4.2, we present a 10.43-approximate algorithm for NMWACT. Then in Section 4.3, we give a variation of this algorithm for NMACT that is 8-approximate. In Section 4.4 we present worst-case examples for both algorithms. Finally, in Section 4.5 we give a practical tip for obtaining improved schedules. As both of our algorithms produce *shelf*-based schedules, we will begin by considering the

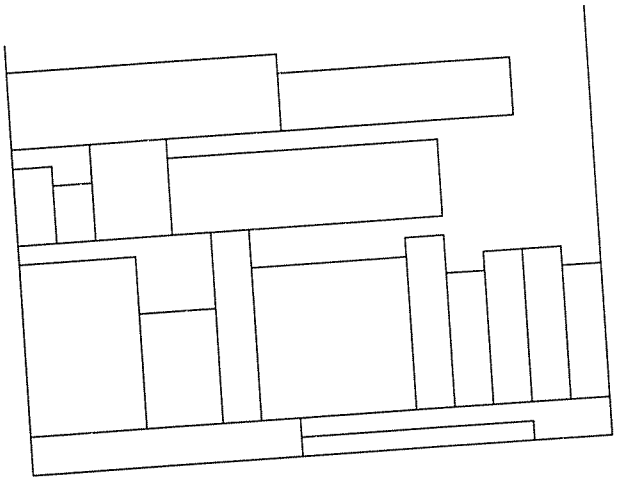


Figure 6: A shelf schedule

properties of such schedules.

4.1 Shelf Schedules

Shelf schedules can be characterized by the following properties. The tasks are assigned to shelves, with all tasks on any given shelf having the same starting time. The sum of the required processors for all tasks on a given shelf must not exceed the total number of processors. The *height* of a shelf is the largest task execution time of any task assigned to that shelf. The first shelf is started at time zero, and the next shelf is started when all tasks on the previous shelf are completed. See Figure 4.1.

If we think of the tasks as rectangles, the physical analogy becomes clear. Rectangle i has height t_i[1], width p_i, and area $a_i = p_i t_i$. Rectangles are packed onto shelves, they must fit on the shelves, the first shelf sits on the floor, and each new shelf rests on the highest rectangle of the shelf before it. So any shelf-based algorithm will apply

[1]Throughout this chapter, we abbreviate $t_i(p_i)$ by writing t_i. We also replace $T(\bar{p})$, $A_T(\bar{p})$, $H_T(\bar{p})$, and $U_T(\bar{p})$ with T, A_T, H_T, and U_T, respectively.

to the problem of scheduling tasks on a line of processors.

If T is a set of nonmalleable tasks, then we can think of a shelf assignment as a surjective function $\mathcal{S} : [n] \to [\ell]$ such that $\sum_{\mathcal{S}(i)=k} p_i \leq m$ for all $k \in [\ell]$. Let $\mathcal{W}_k = \sum_{\mathcal{S}(i)=k} w_i$ be the total weight of the tasks on shelf k, so that $\sum_{k=1}^{\ell} \mathcal{W}_k = (\bar{w})_n$. Let $\mathcal{H}_k = \max_{\mathcal{S}(i)=k} t_i$ be the height of shelf k.

Note that for any shelf assignment $\mathcal{S}$, we can view the pair $(T, \mathcal{S})$ as a set of shelves, and that the indexing of these shelves determines a schedule. The following lemma characterizes an optimal order of a set of shelves.

Lemma 4.1.1 *For any set of shelves $(T, \mathcal{S})$, an ordering of these shelves is optimal if and only if $\frac{\mathcal{H}_k}{\mathcal{W}_k} \leq \frac{\mathcal{H}_{k+1}}{\mathcal{W}_{k+1}}$ for all $k \in [\ell - 1]$.*

Let $\mathcal{Z}(T, \mathcal{S})$ be the schedule that results from ordering the shelves $(T, \mathcal{S})$ optimally as specified by Lemma 4.1.1, and let $R(T, \mathcal{S})$ be its total weighted completion time. This method of ordering the shelves amounts to Smith's ratio rule for scheduling tasks of width 1 on a single processor [52], and has the same proof of optimality.

Proof of Lemma 4.1.1: Suppose that the shelves are not in order of nondecreasing $\mathcal{H}_k/\mathcal{W}_k$. Then there is a shelf k such that $\mathcal{H}_k/\mathcal{W}_k > \mathcal{H}_{k+1}/\mathcal{W}_{k+1}$. If the two shelves are interchanged, then the completion times of the tasks on shelf k will increase by $\mathcal{H}_{k+1}$ and the completion times of the tasks on shelf $k+1$ will decrease by $\mathcal{H}_k$. Therefore, interchanging the shelves decreases the total weighted completion time by $\mathcal{W}_{k+1}\mathcal{H}_k - \mathcal{W}_k\mathcal{H}_{k+1} = \mathcal{W}_k\mathcal{W}_{k+1}(\mathcal{H}_k/\mathcal{W}_k - \mathcal{H}_{k+1}/\mathcal{W}_{k+1}) > 0$. □

4.2 An Algorithm for Weighted Tasks

Let $\gamma > 1$ be a constant that we will specify later. Partition T into *height components* by assigning task i to the height component j that satisfies $\gamma^{j-1} < t_i \leq \gamma^j$. Next assign tasks in each height component j to shelves according to the *NFIW* (for *Next Fit Increasing Width-to-Weight Ratio*) bin packing algorithm: reindex the tasks within height component j in order of nondecreasing width-to-weight ratio and assign them in sequence to shelves, which we regard as bins of size m. Each task is assigned to the current shelf, which is initialized to be shelf number one and incremented by one whenever a task does not fit. Finally, combine all the shelves in the various height components in the order dictated by Lemma 4.1.1. We will call this algorithm γ-SMART$_{\text{NFIW}}$. (Note that SMART stands for Scheduling to Minimize Average Response Time.)

Let $R_{\text{NFIW}}(T)$ be the total weighted completion time of the γ-SMART$_{\text{NFIW}}$ schedule for the task set T. The following theorem is the main result in this section.

Theorem 4.2.1 *For any nonmalleable task set T, the 1.718-SMART$_{NFIW}$ algorithm satisfies*

$$R_{NFIW}(T) \leq 8.751 A_T + 6.051 H_T - 5.093 U_T \leq 10.43 R^*(T). \tag{134}$$

Its running time is $O(n \log n)$.

Corollary 4.2.1 *There is an MWACT algorithm with approximation factor 20.86 and running time $O(n^2 + mn)$.*

Proof: The result is immediate from Corollary 3.3.1. Note that we are using $\alpha = 1$, so there are no conditions on the task set T. □

To prove the approximation factor, we will make use of the lower bound on $R^*(T)$ from Section 2.1, and an upper bound on $R_{\text{NFIW}}(T)$. To obtain this upper bound, we will create a new task set $\hat{T}$ by adjusting the task heights to facilitate the analysis. We will then partition the task set $\hat{T}$ into two subsets, and consider schedules for the two subsets separately, proving upper bounds for each. Finally, we will consider the result of combining the two schedules to obtain a schedule for $\hat{T}$, which will give an upper bound on $R_{\text{NFIW}}(T)$.

We now create a new task set $\hat{T}$ called the *γ-height construction* whose tasks are in natural one-to-one correspondence with the tasks in T. The width of each task in $\hat{T}$ will be identical to that of its counterpart in T, while the task heights will be at least as large but less than γ times larger. In particular, let $\hat{t}_i = \gamma^{\lceil \log_\gamma t_i \rceil}$.

Let $\mathcal{S}_T$ denote the shelf assignment that results from applying γ-SMART$_{\text{NFIW}}$ to the task set T. By construction, for any given shelf k in $(T, \mathcal{S}_T)$, all the tasks on that shelf belong to the same height component. That is, there exists some j such that for all i satisfying $\mathcal{S}_T(i) = k$, we have $\gamma^{j-1} < t_i \leq \gamma^j$. Therefore, every task i on shelf k satisfies $\lceil \log_\gamma t_i \rceil = j$. We conclude that all the tasks on any given shelf in $(\hat{T}, \mathcal{S}_T)$ have identical heights. For this reason, we say that that the set of shelves $(\hat{T}, \mathcal{S}_T)$ satisfies the *uniform height condition.*

Observe that $R_{\text{NFIW}}(T) = R(T, \mathcal{S}_T)$. To get a bound on $R_{\text{NFIW}}(T)$, we will bound $R(\hat{T}, \mathcal{S}_T)$. First we establish a relationship between these two quantities.

Lemma 4.2.1 *Let T be any nonmalleable task set, with the height of task i given by t_i and its width given by p_i, and let $\mathcal{S}$ be any shelf assignment for T. Let V be a nonmalleable task set whose tasks are in one-to-one correspondence with the tasks in T, with the height of task i given by $v_i \geq t_i$ and its width given by p_i. (So each*

task in V is at least as tall and has the same width as its counterpart in T.) Then $R(T, \mathcal{S}) \leq R(V, \mathcal{S}) + H_T - H_V$.

Proof: Observe that the shelf assignment $\mathcal{S}$ applies to V as well as T, and that the sets of shelves $(T, \mathcal{S})$ and $(V, \mathcal{S})$ are identical except for the task heights, with each shelf in $(V, \mathcal{S})$ being at least as tall as its counterpart in $(T, \mathcal{S})$. Let $\mathcal{X}$ denote the schedule that results from starting each shelf of $(T, \mathcal{S})$ at the time the corresponding shelf starts in $\mathcal{Z}(V, \mathcal{S})$, and let $R^{\mathcal{X}}$ denote the total weighted completion time of $\mathcal{X}$. Since we are starting the shelves at the same time in the schedules $\mathcal{X}$ and $\mathcal{Z}(V, \mathcal{S})$, the weighted sum of the task starting times is equivalent for these two schedules. That is,

$$R(V, \mathcal{S}) - H_V = R^{\mathcal{X}} - H_T. \tag{135}$$

Now from Lemma 4.1.1 and (135) we have

$$R(T, \mathcal{S}) \leq R^{\mathcal{X}} = R(V, \mathcal{S}) + H_T - H_V. \tag{136}$$

□

So from Lemma 4.2.1 we conclude that $R(T, \mathcal{S}_T) \leq R(\hat{T}, \mathcal{S}_T) + H_T - H_{\hat{T}}$. The next step is to get an upper bound on $R(\hat{T}, \mathcal{S}_T)$.

To obtain this upper bound, we will partition $\hat{T}$ into two subsets $\hat{T}_1$ and $\hat{T}_2$, and bound the quantities $R(\hat{T}_1, \mathcal{S}_T)$ and $R(\hat{T}_2, \mathcal{S}_T)$ separately. To partition $\hat{T}$, we will partition the task set T into T_1 and T_2, and then the task set $\hat{T}$ will also partition into two subsets $\hat{T}_1$ and $\hat{T}_2$ corresponding task for task to T_1 and T_2. The partition is as follows. The first subset T_1 contains the tasks on the first shelf (if any) of each height component — that is, in each height component, the first shelf created by the NFIW packing. The second subset T_2 contains the tasks on all remaining shelves. Then we also have a partition of the set of shelves $(T, \mathcal{S}_T)$ into $(T_1, \mathcal{S}_T)$ and $(T_2, \mathcal{S}_T)$, and a

partition of the set of shelves $(\hat{T}, \mathcal{S}_T)$ into $(\hat{T}_1, \mathcal{S}_T)$ and $(\hat{T}_2, \mathcal{S}_T)$ corresponding shelf for shelf to $(T_1, \mathcal{S}_T)$ and $(T_2, \mathcal{S}_T)$.

After we have obtained bounds on $R(\hat{T}_1, \mathcal{S}_T)$ and $R(\hat{T}_2, \mathcal{S}_T)$, we can use the following lemma to produce a bound on $R(\hat{T}, \mathcal{S}_T)$.

Lemma 4.2.2 *Let $(V, \mathcal{S})$ be a set of shelves satisfying the uniform height condition. Then for any partition of $(V, \mathcal{S})$ into $(V_1, \mathcal{S})$ and $(V_2, \mathcal{S})$, and any $\delta > 0$, we have $R(V, \mathcal{S}) \leq (\delta + 1) \cdot R(V_1, \mathcal{S}) + (\frac{1}{\delta} + 1) \cdot R(V_2, \mathcal{S})$.*

Proof: Let y_k denote the completion time of shelf k in its respective schedule, either $\mathcal{Z}(V_1, \mathcal{S})$ or $\mathcal{Z}(V_2, \mathcal{S})$. For each shelf k in $(V_1, \mathcal{S})$, let $y'_k = \delta y_k$, and for each shelf k in $(V_2, \mathcal{S})$, let $y'_k = y_k$. Then construct a schedule of all the shelves in $(V, \mathcal{S})$ by arranging them in order of nondecreasing y'_k. Call this schedule $\mathcal{X}$, and let x_k denote the completion time of shelf k in $\mathcal{X}$. Let $R^{\mathcal{X}}$ denote the total weighted completion time of $\mathcal{X}$.

Now consider a shelf k in $(V_1, \mathcal{S})$. Let l be the last shelf in $(V_2, \mathcal{S})$ that is completed before shelf k in $\mathcal{X}$. Since shelf l comes before shelf k, we have $y_l = y'_l \leq y'_k = \delta y_k$. Then $x_k = y_k + y_l \leq (1 + \delta) y_k$. Therefore, the completion time of each task in V_1 in the schedule $\mathcal{X}$ is not more than $\delta + 1$ times its completion time in the schedule $\mathcal{Z}(V_1, \mathcal{S})$.

Next consider a shelf k in $(V_2, \mathcal{S})$. Let l be the last shelf in $(V_1, \mathcal{S})$ that is completed before shelf k in $\mathcal{X}$. Since shelf l comes before shelf k, we have $\delta y_l = y'_l \leq y'_k = y_k$. Then $x_k = y_k + y_l \leq (1 + \frac{1}{\delta}) y_k$. Therefore, the completion time of each task in V_2 in the schedule $\mathcal{X}$ is not more than $\frac{1}{\delta} + 1$ times its completion time in the schedule $\mathcal{Z}(V_2, \mathcal{S})$.

We conclude that $R^{\mathcal{X}} \leq (\delta + 1) \cdot R(V_1, \mathcal{S}) + (\frac{1}{\delta} + 1) \cdot R(V_2, \mathcal{S})$. Now from Lemma 4.1.1 we have $R(V, \mathcal{S}) \leq R^{\mathcal{X}}$, and the result follows. □

It still remains to prove upper bounds on $R(\hat{T}_1, \mathcal{S}_T)$ and $R(\hat{T}_2, \mathcal{S}_T)$. The following

lemma is useful for bounding $R(\hat{T}_1, \mathcal{S}_T)$.

Lemma 4.2.3 *Let $(V, \mathcal{S})$ be a set of shelves satisfying the uniform height condition such that every shelf k has height γ^l for some $\gamma > 1$ and some integer l, and no two shelves have the same height. Then $R(V, \mathcal{S}) \leq \frac{\gamma}{\gamma-1} H_V$.*

Proof: Let $\mathcal{X}$ be a schedule in which the shelves $(V, \mathcal{S})$ are arranged in order of increasing height, and let $R^{\mathcal{X}}$ denote the total weighted completion time of $\mathcal{X}$. If a given shelf has height h, then its completion time in $\mathcal{X}$ is at most $h + \frac{1}{\gamma}h + \frac{1}{\gamma^2}h + \cdots \leq \frac{1}{1-\frac{1}{\gamma}}h = \frac{\gamma}{\gamma-1}h$. Therefore, the completion time of task i in the schedule $\mathcal{X}$ does not exceed $\frac{\gamma}{\gamma-1}v_i$, where v_i is the height of task i, and so we have $R^{\mathcal{X}} \leq \frac{\gamma}{\gamma-1} H_V$. Now by Lemma 4.1.1, we have $R(V, \mathcal{S}) \leq R^{\mathcal{X}}$, and the result follows. □

The set of shelves $(\hat{T}_1, \mathcal{S}_T)$ satisfies the conditions of Lemma 4.2.3, and so we conclude that $R(\hat{T}_1, \mathcal{S}_T) \leq \frac{\gamma}{\gamma-1} H_{\hat{T}_1}$. The following pair of lemmas can be used to bound $R(\hat{T}_2, \mathcal{S}_T)$.

Lemma 4.2.4 *The γ-height construction $\hat{T}$ satisfies $A_{\hat{T}} \leq \gamma A_T$ and $H_{\hat{T}_1} \leq \gamma H_{T_1}$.*

Proof: First observe that for any task i, we have $\hat{t}_i = \gamma^{\lceil \log_\gamma t_i \rceil} < \gamma^{\log_\gamma t_i + 1} = \gamma t_i$. Now it follows directly that $H_{\hat{T}_1} < \gamma H_{T_1}$.

Now let $\hat{a}_i = p_i \hat{t}_i$, and we have $\hat{a}_i < \gamma a_i$. Let ϕ be a permutation that is induced by the task set $\hat{T}$, and let ψ be a permutation that is induced by the task set T. Then using Corollary 2.3.2, we have

$$A_{\hat{T}} = \frac{1}{m} \sum_{i=1}^{n} (\bar{w})_i^{\phi} \hat{a}_i \leq \frac{1}{m} \sum_{i=1}^{n} (\bar{w})_i^{\psi} \hat{a}_i < \frac{\gamma}{m} \sum_{i=1}^{n} (\bar{w})_i^{\psi} a_i = \gamma A_T. \tag{137}$$

□

Lemma 4.2.5 *The partition $\hat{T}_2$ satisfies $R(\hat{T}_2, \mathcal{S}_T) \leq 2A_{\hat{T}} - 2A_{\hat{T}_1}$.*

We delay the proof of this lemma until the end of this section.

Now applying Lemmas 4.2.1, 4.2.2, 4.2.3, 4.2.4, and 4.2.5 yields

$$\begin{aligned}
R_{\text{NFIW}}(T) &\leq R(\hat{T}, \mathcal{S}_T) + H_T - H_{\hat{T}} \\
&\leq (\delta+1)R(\hat{T}_1, \mathcal{S}_T) + (\frac{1}{\delta}+1)R(\hat{T}_2, \mathcal{S}_T) + H_T - H_{\hat{T}} \\
&\leq 2(\frac{1}{\delta}+1)A_{\hat{T}} - 2(\frac{1}{\delta}+1)A_{\hat{T}_1} + \frac{\gamma}{\gamma-1}(\delta+1)H_{\hat{T}_1} - H_{\hat{T}} + H_T \\
&= 2(\frac{1}{\delta}+1)A_{\hat{T}} - 2(\frac{1}{\delta}+1)A_{\hat{T}_1} + \left[\frac{\gamma}{\gamma-1}(\delta+1) - 1\right] H_{\hat{T}_1} - H_{\hat{T}_2} + H_T \\
&\leq 2\gamma(\frac{1}{\delta}+1)A_T - 2(\frac{1}{\delta}+1)A_{T_1} + \left[\frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma\right] H_{T_1} - H_{T_2} + H_T \\
&= 2\gamma(\frac{1}{\delta}+1)A_T - 2(\frac{1}{\delta}+1)A_{T_1} + \left[\frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1\right] (H_T - H_{T_2}) \\
&\leq 2\gamma(\frac{1}{\delta}+1)A_T + \left[\frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1\right] H_T \\
&\quad - 2(\frac{1}{\delta}+1)U_{T_1} - \left[\frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1\right] U_{T_2} \\
&\leq 2\gamma(\frac{1}{\delta}+1)A_T + \left[\frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1\right] H_T \\
&\quad - \min\left\{2(\frac{1}{\delta}+1), \frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1\right\} U_T. \qquad (138)
\end{aligned}$$

Then by Corollary 2.1.2, we have

$$\begin{aligned}
R_{\text{NFIW}}(T) \leq \max\Bigg\{&2\gamma(\frac{1}{\delta}+1), \gamma(\frac{1}{\delta}+1) + \frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1, \\
&2(\gamma-1)(\frac{1}{\delta}+1) + \frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1\Bigg\} \cdot R^*(T). \qquad (139)
\end{aligned}$$

Now choosing

$$\delta = \frac{1}{18}(692 + 108\sqrt{29})^{1/3} + \frac{26}{9}\frac{1}{(692 + 108\sqrt{29})^{1/3}} - \frac{2}{9} \approx .64648835 \qquad (140)$$

and

$$\gamma = \frac{1}{1-\delta^2} \approx 1.7180572 \qquad (141)$$

yields

$$R_{\text{NFIW}}(T) \leq 8.7511588 A_T + 6.0501691 H_T - 5.0936365 U_T \tag{142}$$

and

$$\begin{aligned} R_{\text{NFIW}}(T) &\leq \left[\gamma(\frac{1}{\delta}+1) + \frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1\right] R^*(T) \\ &\leq 10.425749 R^*(T). \end{aligned} \tag{143}$$

This completes the proof of Theorem 4.2.1. We conclude this section with a proof of Lemma 4.2.5.

Proof of Lemma 4.2.5: Arrange the shelves in $(\hat{T}_2, \mathcal{S}_T)$ according to the area-to-weight ratio of the task on each shelf with the least such ratio, so that the shelves are in nondecreasing order according to that ratio. Call this schedule $\mathcal{Y}$. Let g_i denote the completion time of task $i \in \hat{T}_2$ in the schedule $\mathcal{Y}$, and let $R^{\mathcal{Y}} = \sum_{i \in \hat{T}_2} w_i g_i$ be the total weighted completion time of $\mathcal{Y}$.

Our goal is to relate each task's completion time g_i to its contribution to $A_{\hat{T}}$. For this purpose, we will carefully reindex the tasks. For each task i in $\hat{T}$, let $\hat{a}_i = p_i \hat{t}_i$ denote its area. Now reindex the tasks in $\hat{T}$ to satisfy the following three conditions. First, the tasks are ordered by nondecreasing ratio of area to weight, i.e., $\hat{a}_i / w_i \leq \hat{a}_{i+1} / w_{i+1}$ for all $i \in [n-1]$. Second, tasks with equal area-to-weight ratio are ordered by nondecreasing start time in $\mathcal{Y}$. Third, within each height component, the indexing reflects the order in which the tasks are packed onto shelves. Note that the third condition does not conflict with the first two, since within a height component, all tasks in $\hat{T}$ have the same height, and so ordering the tasks by nondecreasing width-to-weight ratio also orders them by nondecreasing area-to-weight ratio. Let

$$\alpha_i = \sum_{j=1}^{i} \hat{a}_j. \tag{144}$$

Then

$$A_{\hat{T}} = \frac{1}{m} \sum_{i=1}^{n} w_i \alpha_i. \tag{145}$$

Also note that

$$A_{\hat{T}_1} \leq \frac{1}{m} \sum_{i \in \hat{T}_1} w_i \alpha_i, \tag{146}$$

and these quantities are not equal in general because α_i may include tasks that are not in $\hat{T}_1$. Our goal is to show that $g_i \leq \frac{2}{m}\alpha_i$ for all $i \in \hat{T}_2$. Using Lemma 4.1.1 along with (145) and (146), we will then conclude that

$$R(\hat{T}_2, \mathcal{S}_T) \leq \sum_{i \in \hat{T}_2} w_i g_i \leq \frac{2}{m} \sum_{i \in \hat{T}_2} w_i \alpha_i = 2 \left(\frac{1}{m} \sum_{i \in \hat{T}} w_i \alpha_i - \frac{1}{m} \sum_{i \in \hat{T}_1} w_i \alpha_i \right) \leq 2(A_{\hat{T}} - A_{\hat{T}_1}). \tag{147}$$

To prove that $g_i \leq \frac{2}{m}\alpha_i$, we will begin by proving a similar result for each individual height component. To this end we introduce the following notation. For any two tasks i and j, we write $i \sim j$ if they are in the same height component, i.e., $\lceil \log_\gamma t_i \rceil = \lceil \log_\gamma t_j \rceil$. For a given task i, consider the schedule consisting only of the shelves in $(\hat{T}_2, \mathcal{S}_T)$ that contain tasks that are in the same height component as i, with the shelves ordered as they are in $\mathcal{Y}$. Note that this is the order in which the shelves are created when the tasks are packed onto shelves using NFIW. Call this schedule $\mathcal{Y}(i)$. For $i \in \hat{T}_2$, let g_i' denote the completion time of task i in $\mathcal{Y}(i)$. Also let

$$\alpha_i' = \sum_{j \leq i \wedge j \sim i} \hat{a}_j. \tag{148}$$

Then we have the following result pertaining to individual height components.

Lemma 4.2.6 *$g_i' \leq \frac{2}{m}\alpha_i'$ for all $i \in \hat{T}_2$.*

Let us delay the proof of Lemma 4.2.6 until the completion of the proof of Lemma 4.2.5.

Now we are ready to show that $g_i \leq \frac{2}{m}\alpha_i$ for all $i \in \hat{T}_2$. Pick any task $j_0 \in \hat{T}_2$. Suppose that there are r height components, other than the component containing the

task j_0 itself, that contain a shelf that comes before j_0 in the schedule $\mathcal{Y}$. For each such component $l \in [r]$, let i_l be a task with least area-to-weight ratio on the last shelf of the component l that comes before the shelf containing j_0. Also let i_0 be a task with least area-to-weight ratio on the shelf containing j_0. Then $i_0 \leq j_0$. Note that the ordering of the shelves guarantees that $\hat{a}_{i_l}/w_{i_l} \leq \hat{a}_{i_0}/w_{i_0} \leq \hat{a}_{j_0}/w_{j_0}$ for all $l \in [r]$, and so the indexing of the tasks guarantees that $i_l < j_0$ for all $l \in [r]$. Now applying Lemma 4.2.6, we get

$$g_{j_0} = \sum_{l=0}^{r} g'_{i_l} \leq \frac{2}{m} \sum_{l=0}^{r} \alpha'_{i_l} = \frac{2}{m} \sum_{l=0}^{r} \sum_{j \leq i_l \wedge j \sim i_l} \hat{a}_j \leq \frac{2}{m} \sum_{j=1}^{j_0} \hat{a}_j = \frac{2}{m} \alpha_{j_0}. \tag{149}$$

□

Proof of Lemma 4.2.6: Let j_0 be any task in $\hat{T}_2$, and let $T^0 = \{i \in \hat{T} : i \sim j_0\}$ be the height component containing j_0. Index the shelves in $(T^0, \mathcal{S}_T)$ according to the order in which they were created. Then shelf 1 is in $(\hat{T}_1, \mathcal{S}_T)$ and shelf 2 is the first shelf in $\mathcal{Y}(j_0)$. Note that tasks in $\hat{T}_1$ contribute to α'_{j_0}, but not to g'_{j_0}.

Let shelf k_0 be the shelf containing the task j_0, and let i_0 be the first task assigned to shelf k_0. Let z denote the total area of the tasks on the $k_0 - 1$ preceding shelves. Observe that the first task assigned to any shelf $k > 1$ is too wide to fit on shelf $k - 1$, and therefore the total width of any shelf plus the first task on the next shelf is greater than m. Also, the total width of any two consecutive shelves is greater than m. Since every task in T^0 has height $h = \hat{t}_{j_0} = \gamma^{\lceil \log_\gamma t_{j_0} \rceil}$, it follows that

$$z + \hat{a}_{i_0} > \left\lfloor \frac{k_0}{2} \right\rfloor hm \geq \frac{k_0 - 1}{2} hm. \tag{150}$$

Also, the way the tasks are indexed guarantees that

$$z + \hat{a}_{i_0} \leq \alpha'_{j_0}. \tag{151}$$

Recall that shelf 1 is in $(\hat{T}_1, \mathcal{S}_T)$, and therefore not in $\mathcal{Y}(j_0)$, so the completion time

of task j_0 in $\mathcal{Y}(j_0)$ is

$$g'_{j_0} = (k_0 - 1)h. \tag{152}$$

Then from (152), (150), and (151), we have

$$g'_{j_0} = (k_0 - 1)h < \frac{2}{m}(z + \hat{a}_{i_0}) \leq \frac{2}{m}\alpha'_{j_0}. \tag{153}$$

□

4.3 An Improved Algorithm for Unweighted Tasks

In the case of unweighted tasks, we can achieve an improved approximation factor for a variant of γ-SMART$_{\text{NFIW}}$. Specifically, instead of assigning tasks to shelves using NFIW, we use *FFIA* (*First Fit Increasing Area*): for each height component, reindex the tasks within the component in order of nondecreasing area and assign them in sequence to shelves, which we again regard as bins of size m. Each task is assigned to the first shelf on which it fits, and the number of shelves is incremented by one whenever a task does not fit on any preceding shelf. (The rest of the algorithm, including the partitioning of the tasks into height components, and the combining of the shelves of the various components, remains the same.) We call this algorithm γ-SMART$_{\text{FFIA}}$.

The main result in this section is the following theorem.

Theorem 4.3.1 *For any nonmalleable task set T, the 2-SMART$_{FFIA}$ algorithm satisfies*

$$R_{FFIA}(T) \leq 6A_T + 5H_T - 5U_T \leq 8R^*(T). \tag{154}$$

Its running time is $O(n \log n)$.

Corollary 4.3.1 *There is a MACT algorithm with approximation factor 8 and running time $O(n^3 + mn)$, and another with approximation factor 16 and running time $O(n^2 + mn)$.*

Proof: The first algorithm follows from Corollary 3.2.1, and the second from Corollary 3.3.1. In both cases $\alpha = 1$, so there are no conditions on T except that $w_i = 1$ for all i. □

We obtain an upper bound on $R_{\text{FFIA}}(T)$ in the same manner as we obtained a bound on $R_{\text{NFIW}}(T)$ in Section 4.2. We partition the task set T into T_1 and T_2 in the same way, and prove bounds for schedules for the two subsets separately. Then we consider the result of combining the two schedules. However, we make use of a slightly different γ-height construction. In particular, we construct a new task set $\tilde{T}$ using the height $\tilde{t}_i = \gamma^{\lceil \log_\gamma t_i \rceil}$ only for tasks $i \in T_1$. For tasks $i \in T_2$, we use the height of the shelf containing task i, i.e., $\tilde{t}_i = \max\{t_j : \mathcal{S}_T(j) = \mathcal{S}_T(i)\}$, where $\mathcal{S}_T$ denotes the shelf assignment that results from applying γ-SMART$_{\text{FFIA}}$ to the task set T. Observe that Lemmas 4.2.1, 4.2.2, and 4.2.3 all apply to the new construction $\tilde{T}$. The improvement in the approximation factor is the result of replacing Lemma 4.2.5 with the following lemma.

Lemma 4.3.1 *If $\gamma \geq 2$, then the partition $\tilde{T}_2$ satisfies $R(\tilde{T}_2, \mathcal{S}_T) \leq \gamma(A_T - A_{T_1})$.*

Given this result, we do not need a bound on $A_{\tilde{T}}$ relative to A_T. However, we still need to bound $H_{\tilde{T}_1}$ relative to H_{T_1}, and for this purpose, we make use of a restricted form of Lemma 4.2.4 adapted to the new construction.

Lemma 4.3.2 *For any task set T, we have $H_{\tilde{T}_1} \leq \gamma H_{T_1}$.*

Proof: For any task $i \in T_1$, we have $\tilde{t}_i = \gamma^{\lceil \log_\gamma t_i \rceil} < \gamma^{\log_\gamma t_i + 1} = \gamma t_i$. Now it follows directly that $H_{\tilde{T}_1} < \gamma H_{T_1}$. □

Now choosing $\gamma \geq 2$ and applying Lemmas 4.2.1, 4.2.2, 4.2.3, 4.3.1, and 4.3.2 yields

$$\begin{aligned}
R_{\text{FFIA}}(T) &\leq R(\tilde{T}, \mathcal{S}_T) + H_T - H_{\tilde{T}} \\
&\leq (\delta+1) \cdot R(\tilde{T}_1, \mathcal{S}_T) + (\frac{1}{\delta}+1) \cdot R(\tilde{T}_2, \mathcal{S}_T) + H_T - H_{\tilde{T}} \\
&\leq \gamma(\frac{1}{\delta}+1)A_T - \gamma(\frac{1}{\delta}+1)A_{T_1} + \frac{\gamma}{\gamma-1}(\delta+1)H_{\tilde{T}_1} - H_{\tilde{T}} + H_T \\
&= \gamma(\frac{1}{\delta}+1)A_T - \gamma(\frac{1}{\delta}+1)A_{T_1} + \left[\frac{\gamma}{\gamma-1}(\delta+1) - 1\right] H_{\tilde{T}_1} - H_{\tilde{T}_2} + H_T \\
&\leq \gamma(\frac{1}{\delta}+1)A_T - \gamma(\frac{1}{\delta}+1)A_{T_1} + \left[\frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma\right] H_{T_1} - H_{T_2} + H_T \\
&= \gamma(\frac{1}{\delta}+1)A_T - \gamma(\frac{1}{\delta}+1)A_{T_1} + \left[\frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1\right] (H_T - H_{T_2}) \\
&\leq \gamma(\frac{1}{\delta}+1)A_T + \left[\frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1\right] H_T \\
&\quad - \gamma(\frac{1}{\delta}+1)U_{T_1} - \left[\frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1\right] U_{T_2} \\
&\leq \gamma(\frac{1}{\delta}+1)A_T + \left[\frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1\right] H_T \\
&\quad - \min\left\{\gamma(\frac{1}{\delta}+1), \frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1\right\} U_T. \qquad (155)
\end{aligned}$$

Then by Corollary 2.1.2, we have

$$R_{\text{FFIA}}(T) \leq \max\left\{\gamma(\frac{1}{\delta}+1), \frac{\gamma}{2}(\frac{1}{\delta}+1) + \frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1\right\} \cdot R^*(T). \qquad (156)$$

Now choosing $\gamma = 2$ and $\delta = \frac{1}{2}$, we get

$$R_{\text{FFIA}}(T) \leq 6A_T + 5H_T - 5U_T \qquad (157)$$

and

$$\begin{aligned}
R_{\text{FFIA}}(T) &\leq \left[\frac{\gamma}{2}(\frac{1}{\delta}+1) + \frac{\gamma^2}{\gamma-1}(\delta+1) - \gamma + 1\right] R^*(T) \\
&\leq 8R^*(T). \qquad (158)
\end{aligned}$$

This completes the proof of the approximation factor of Theorem 4.3.1. As for the running time, note that the FFIA packing of the tasks onto shelves can be computed in $O(n \log n)$ steps using the method described by Johnson [27, 28]. We will conclude this section with a proof Lemma 4.3.1. The proof will make use of the following alternate definition of A_T.

Lemma 4.3.3 *Let T be a set of unweighted nonmalleable tasks, indexed arbitrarily. Then*

$$A_T = \frac{1}{m} \sum_{i=1}^{n} \sum_{j=1}^{i} \min\{a_i, a_j\}. \tag{159}$$

Proof: Observe that $\sum_{i=1}^{n} \sum_{j=1}^{i} \min\{a_i, a_j\} = \sum_{i \geq j} \min\{a_i, a_j\}$ is the sum over all pairs of elements of the minimum of each pair. This sum is independent of the way the elements are indexed. Suppose, then, that they are indexed by nondecreasing area, $a_1 \leq a_2 \leq \cdots \leq a_n$. Then $\frac{1}{m} \sum_{i=1}^{n} \sum_{j=1}^{i} \min\{a_i, a_j\} = \frac{1}{m} \sum_{i=1}^{n} \sum_{j=1}^{i} a_j = A_T$. □

Proof of Lemma 4.3.1: The proof is similar to the proof of Lemma 4.2.5. Let $\nu(k) = \min_{\mathcal{S}_T(i)=k}\{a_i\}$ be the area of the least-area task in T on shelf k. Note that the area in question is the task's *original* area, not its area in $\tilde{T}$. Then reindex the shelves in $(\tilde{T}, \mathcal{S}_T)$ so that $\nu(k) \leq \nu(k+1)$ for all shelves k. That is, we are ordering the shelves by nondecreasing area of the least-area task. Take the shelves of $(\tilde{T}_2, \mathcal{S}_T)$ in the order they are indexed, and call this schedule $\mathcal{Y}$. Let g_i denote the completion time of task $i \in \tilde{T}_2$ in the schedule $\mathcal{Y}$, and let $R^{\mathcal{Y}} = \sum_{i \in \tilde{T}_2} g_i$ be the total completion time of $\mathcal{Y}$.

Our goal is to relate each task's completion time g_i to its contribution to A_T. For this purpose, we reindex the tasks so that $\mathcal{S}_T(i) < \mathcal{S}_T(j) \Rightarrow i < j$. That is, tasks that appear on lower shelves have lower indices. Also, within each shelf, tasks are indexed

by nondecreasing area. Then define

$$\alpha_i = \sum_{j=1}^{i} \min\{a_i, a_j\}. \tag{160}$$

By Lemma 4.3.3, we have

$$A_T = \frac{1}{m} \sum_{i=1}^{n} \alpha_i. \tag{161}$$

Also note that

$$A_{T_1} \leq \frac{1}{m} \sum_{i \in T_1} \alpha_i, \tag{162}$$

and these quantities will be unequal if there is some $i \in T_1$ for which α_i includes tasks that are not in T_1. Our goal is to show that $g_i \leq \frac{\gamma}{m}\alpha_i$ for all $i \in \tilde{T}_2$. Using Lemma 4.1.1 along with (161) and (162), we will then conclude that

$$R(\tilde{T}_2, \mathcal{S}_T) \leq \sum_{i \in \tilde{T}_2} g_i \leq \frac{\gamma}{m} \sum_{i \in \tilde{T}_2} \alpha_i = \gamma \left(\frac{1}{m} \sum_{i \in T} \alpha_i - \frac{1}{m} \sum_{i \in T_1} \alpha_i \right) \leq \gamma (A_T - A_{T_1}). \tag{163}$$

To prove that $g_i \leq \frac{\gamma}{m}\alpha_i$, we will begin by proving a similar result for each individual height component. For a given task i, consider the schedule consisting only of the shelves in $(\tilde{T}_2, \mathcal{S}_T)$ that contain tasks that are in the same height component as i, with the shelves ordered as they are in $\mathcal{Y}$. Note that this is the order in which the shelves are created when the tasks are packed onto shelves using FFIA. Call this schedule $\mathcal{Y}(i)$. For $i \in \tilde{T}_2$, let g_i' denote the completion time of task i in $\mathcal{Y}(i)$. Also let

$$\alpha_i' = \sum_{j \leq i \wedge j \sim i} \min\{a_i, a_j\}, \tag{164}$$

where $i \sim j$ indicates that tasks i and j are in the same height component. Then we have the following result pertaining to individual height components.

Lemma 4.3.4 *$g_i' \leq \frac{\gamma}{m}\alpha_i'$ for all $i \in \tilde{T}_2$.*

Let us delay the proof of Lemma 4.3.4 until the completion of the proof of Lemma 4.3.1.

Now we are ready to show that $g_i \leq \frac{\gamma}{m}\alpha_i$ for all $i \in \tilde{T}_2$. Pick any task $j_0 \in \tilde{T}_2$. Suppose that there are r height components, other than the component containing the task j_0 itself, that contain a shelf that comes before j_0 in the schedule $\mathcal{Y}$. For each such component $l \in [r]$, let i_l be a least-area task on the last shelf of the component l that comes before the shelf containing j_0. Also let i_0 be the task with least index on the shelf containing j_0. Then $a_{i_0} \leq a_{j_0}$. Note that the indexing of the shelves and the indexing of the tasks guarantee that $i_l < j_0$ for all $l \in [r]$. Also note that $a_{i_l} \leq a_{i_0} \leq a_{j_0}$ for all $l \in [r]$. Now applying Lemma 4.3.4, we have

$$g_{j_0} = \sum_{l=0}^{r} g'_{i_l} \leq \frac{\gamma}{m}\sum_{l=0}^{r}\alpha'_{i_l} = \frac{\gamma}{m}\sum_{l=0}^{r}\sum_{j\leq i_l \wedge j\sim i_l} \min\{a_{i_l}, a_j\} \leq \frac{\gamma}{m}\sum_{j=1}^{j_0}\min\{a_{j_0}, a_j\} = \frac{\gamma}{m}\alpha_{j_0}. \tag{165}$$

□

Proof of Lemma 4.3.4: Let j_0 be any task in $\tilde{T}_2$, and let $T^0 = \{i \in \tilde{T} : i \sim j_0\}$ be the height component containing j_0. Index the shelves in $(T^0, \mathcal{S}_T)$ according to the order in which they were created. Then shelf 1 is in $(\tilde{T}_1, \mathcal{S}_T)$ and shelf 2 is the first shelf in $\mathcal{Y}(j_0)$. Note that tasks in $\tilde{T}_1$ contribute to α'_{j_0}, but not to g'_{j_0}.

Let shelf k_0 be the shelf containing the task j_0. Let $K = [k_0 - 1]$ be the set of shelves that precede the task j_0 in the schedule $\mathcal{Y}(j_0)$, along with the shelf from $(T^0, \mathcal{S}_T)$ that is in $(\tilde{T}_1, \mathcal{S}_T)$. We will examine the shelves in K and determine the contribution of each to g'_{j_0} and to α'_{j_0}. For this purpose, we will partition K into three subsets. Some additional notation is required first.

Let $\mathcal{S}_T^{-1}(k)$ denote the set of tasks on shelf k. Define the *gap* of shelf k with respect to task j_0 to be

$$d_k = m - \sum_{\{i\in\mathcal{S}_T^{-1}(k):a_i\leq a_{j_0}\}} p_i. \tag{166}$$

Then d_k is "how much room" is left on shelf k at the time when task j_0 is assigned to a shelf. (If there are tasks on shelf k with the same area as j_0, then d_k may be less than the amount of room that is left on shelf k when j_0 is assigned to a shelf.) Note that $d_k < p_{j_0}$ for all $k \in K$ due to the use of first fit. Also note that there is at most one shelf $k \in K$ such that $d_k \geq m/2$. Let v denote this shelf if it exists; otherwise, let $v = \infty$. Let $K^- = K \setminus \{1\}$. That is, K^- is the set of shelves in $\mathcal{Y}(j_0)$ that come before task j_0.

Now we are ready to partition K into three subsets. Let

$$K_1 = \{k \in K^- : \exists i \in \mathcal{S}_T^{-1}(k) \text{ such that } a_i > a_{j_0}\} \tag{167}$$

be the set of shelves containing a task larger than j_0, but not including shelf 1. Let

$$K_2 = \{k \in K^- \setminus K_1 : k > v\} \tag{168}$$

be the set of shelves that do not contain any tasks larger than j_0 and that are scheduled after the shelf with $d_v \geq m/2$. Finally, let

$$K_3 = \{k \in K^- \setminus K_1 : k \leq v\} \cup \{1\} \tag{169}$$

be the remaining shelves — shelf 1 along with all shelves that do not contain any tasks larger than j_0 and that are scheduled not later than the shelf with $d_v \geq m/2$.

Let $h = \gamma^{\lceil \log_\gamma t_{j_0} \rceil}$ be the maximum possible height of a shelf in $(T^0, \mathcal{S}_T)$, and let the height of shelf k be given by $h/\gamma + x_k$, where $0 < x_k \leq (1 - 1/\gamma)h$. Then the completion time of task j_0 in $\mathcal{Y}(j_0)$ does not exceed h plus the sum of the heights of the shelves that precede it in $\mathcal{Y}(j_0)$. That is,

$$g'_{j_0} \leq \sum_{k \in K^-} (x_k + h/\gamma) + h. \tag{170}$$

$$\alpha'_{j_0} = \sum_{j \le j_0 \wedge j \sim j_0} \min\{a_{j_0}, a_j\} \ge \sum_{k \in K} \sum_{j \in \mathcal{S}_T^{-1}(k)} \min\{a_{j_0}, a_j\} + a_{j_0}. \tag{171}$$

iat

$$\begin{aligned} {}_{j_0} - g'_{j_0} &\ge \frac{\gamma}{m} \sum_{k \in K} \sum_{j \in \mathcal{S}_T^{-1}(k)} \min\{a_{j_0}, a_j\} + \frac{\gamma}{m} a_{j_0} - \sum_{k \in K^-} (x_k + h/\gamma) - h \\ &= \sum_{k \in K^-} \left[\frac{\gamma}{m} \sum_{j \in \mathcal{S}_T^{-1}(k)} \min\{a_{j_0}, a_j\} - (x_k + h/\gamma) \right] \\ &\quad + \frac{\gamma}{m} \sum_{j \in \mathcal{S}_T^{-1}(1)} \min\{a_{j_0}, a_j\} + \frac{\gamma}{m} a_{j_0} - h. \end{aligned} \tag{172}$$

$k \in K^-$, define (173)

$$\lambda_k = \frac{\gamma}{m} \sum_{j \in \mathcal{S}_T^{-1}(k)} \min\{a_{j_0}, a_j\} - (x_k + h/\gamma).$$

efine (174)

$$S_1 = \sum_{j \in K_1} \lambda_k; \tag{175}$$

$$S_2 = \sum_{j \in K_2} \lambda_k; \tag{176}$$

$$S_3 = \sum_{j \in K_3 \setminus \{1\}} \lambda_k + \frac{\gamma}{m} \sum_{j \in \mathcal{S}_T^{-1}(1)} \min\{a_{j_0}, a_j\} + \frac{\gamma}{m} a_{j_0} - h.$$

:n from (172) we have

$$\begin{aligned} \frac{\gamma}{m} \alpha'_{j_0} - g'_{j_0} &\ge \sum_{k \in K^-} \lambda_k + \frac{\gamma}{m} \sum_{j \in \mathcal{S}_T^{-1}(1)} \min\{a_{j_0}, a_j\} + \frac{\gamma}{m} a_{j_0} - h \\ &= S_1 + S_2 + S_3. \end{aligned} \tag{177}$$

Now our goal is to show that $S_1 + S_2 + S_3 \ge 0$. We will do this by showing $S_i \ge 0$ for each $i \in [3]$. To see that $S_1 \ge 0$, consider any $k \in K_1$. There is at least one task on

shelf k with area greater than a_{j_0}, so

$$\begin{aligned}\sum_{j\in\mathcal{S}_T^{-1}(k)} \min\{a_{j_0}, a_j\} &\geq a_{j_0} + \sum_{\{j\in\mathcal{S}_T^{-1}(k):a_j\leq a_{j_0}\}} a_j \\ &\geq a_{j_0} + \frac{h}{\gamma}\left(\sum_{\{j\in\mathcal{S}_T^{-1}(k):a_j\leq a_{j_0}\}} p_j\right) \\ &= a_{j_0} + \frac{h}{\gamma}(m - d_k) \\ &= a_{j_0} - \frac{h}{\gamma}d_k + \frac{h}{\gamma}m. \end{aligned} \tag{178}$$

It follows from (173) and (178) that

$$\begin{aligned}\lambda_k &\geq \frac{\gamma}{m}\left(a_{j_0} - \frac{h}{\gamma}d_k\right) + h - (x_k + h/\gamma) \\ &> 0, \end{aligned} \tag{179}$$

because $a_{j_0} = t_{j_0}p_{j_0} > \frac{h}{\gamma}d_k$, and $h \geq x_k + h/\gamma$. So $S_1 = \sum_{k\in K_1}\lambda_k \geq 0$.

Next consider $k \in K_2$. Suppose $q \in \mathcal{S}_T^{-1}(k)$. By the definition of K_2, we have $\mathcal{S}_T(q) > v$ and $a_q \leq a_{j_0}$, and so $p_q > d_v \geq m/2$. Therefore, shelf k contains only one task q. We conclude that

$$\lambda_k = \frac{\gamma}{m}a_q - t_q \geq \frac{2}{m}p_q t_q - t_q > 0, \tag{180}$$

because $p_q > m/2$. Then $S_2 = \sum_{k\in K_2}\lambda_k \geq 0$.

Finally, consider the shelves in K_3. Let $1 = k_1 < k_2 < \cdots < k_r$ denote all the shelves in K_3. By the definition of K_3, each shelf k_q with $q > 1$ contains no task with area exceeding a_{j_0}. Also observe that the tallest task on shelf k_q has width greater than $d_{k_{q-1}}$ (as do all the tasks on that shelf) and height $h/\gamma + x_{k_q}$. Then for $q > 1$, we have

$$\sum_{j\in\mathcal{S}_T^{-1}(k_q)} \min\{a_{j_0}, a_j\} = \sum_{j\in\mathcal{S}_T^{-1}(k_q)} a_j$$

$$\begin{aligned} &\geq \frac{h}{\gamma} \sum_{j \in \mathcal{S}_T^{-1}(k_q)} p_j + x_{k_q} d_{k_{q-1}} \\ &= \frac{h}{\gamma}(m - d_{k_q}) + x_{k_q} d_{k_{q-1}}. \end{aligned} \tag{181}$$

Now from (173) and (181) it follows that for $q > 1$, we have

$$\begin{aligned} \lambda_{k_q} &\geq h - \frac{h d_{k_q}}{m} + \frac{\gamma x_{k_q} d_{k_{q-1}}}{m} - (x_{k_q} + h/\gamma) \\ &= (1 - 1/\gamma)h - x_{k_q} - \frac{h d_{k_q}}{m} + \frac{\gamma x_{k_q} d_{k_{q-1}}}{m}. \end{aligned} \tag{182}$$

For shelf $k_1 = 1$, we have

$$\begin{aligned} \sum_{j \in \mathcal{S}_T^{-1}(1)} \min\{a_{j_0}, a_j\} &\geq \sum_{\{j \in \mathcal{S}_T^{-1}(1): a_j \leq a_{j_0}\}} a_j \\ &\geq \frac{h}{\gamma} \left(\sum_{\{j \in \mathcal{S}_T^{-1}(1): a_j \leq a_{j_0}\}} p_j \right) \\ &= \frac{h}{\gamma}(m - d_1). \end{aligned} \tag{183}$$

Then from (176), (182), and (183), we have

$$\begin{aligned} S_3 &= \sum_{k \in K_3 \setminus \{1\}} \lambda_k + \frac{\gamma}{m} \sum_{j \in \mathcal{S}_T^{-1}(1)} \min\{a_{j_0}, a_j\} + \frac{\gamma}{m} a_{j_0} - h \\ &\geq \sum_{q=2}^{r} \left[(1 - 1/\gamma)h - x_{k_q} - \frac{h d_{k_q}}{m} + \frac{\gamma x_{k_q} d_{k_{q-1}}}{m} \right] + h - \frac{h d_1}{m} + \frac{\gamma}{m} a_{j_0} - h \\ &= \sum_{q=2}^{r} \left[(1 - 1/\gamma)h - x_{k_q} - \frac{h d_{k_q}}{m} + \frac{\gamma x_{k_q} d_{k_{q-1}}}{m} \right] - \frac{h d_1}{m} + \frac{\gamma}{m} a_{j_0} \\ &= \sum_{q=2}^{r} \left[(1 - 1/\gamma)h - x_{k_q} - \frac{h d_{k_{q-1}}}{m} + \frac{\gamma x_{k_q} d_{k_{q-1}}}{m} \right] - \frac{h d_{k_r}}{m} + \frac{\gamma}{m} a_{j_0} \\ &= \sum_{q=2}^{r} \left[(1 - 1/\gamma)h - x_{k_q} - 2(1 - 1/\gamma)\frac{h d_{k_{q-1}}}{m} + \frac{2 x_{k_q} d_{k_{q-1}}}{m} \right] - \frac{h d_{k_r}}{m} + \frac{\gamma}{m} a_{j_0} \\ &\geq \sum_{q=2}^{r} \left[(1 - 1/\gamma)h - x_{k_q} \right] \left(1 - \frac{2 d_{k_{q-1}}}{m} \right) + \frac{\gamma}{m} \left(a_{j_0} - \frac{h}{\gamma} d_{k_r} \right) \\ &> 0. \end{aligned} \tag{184}$$

For the last inequality, observe that $k_{q-1} < k_r \leq v$, and so $d_{k_{q-1}} < m/2$ for all $q \in \{2, \ldots, r\}$. and also that $a_{j_0} = t_{j_0} p_{j_0} > \frac{h}{\gamma} d_{k_r}$. □

4.4 Worst-Case Examples

We first present a family of unweighted nonmalleable task sets for which 2-SMART$_{\text{FFIA}}$ produces a ratio $R_{\text{FFIA}}(T)/R^*(T)$ approaching 9/2 asymptotically. There are two groups of tasks, a *height* group and an *area* group. The tasks in the height group fit onto $r-1$ shelves of heights $4, 8, 16, \ldots, 2^r$, with the number of tasks on each shelf being equal to the height of that shelf. On the shelf of height 2^k there is one task of full height 2^k. All other tasks on the shelf are of height slightly larger than 2^{k-1}. The width of each task in the height group is 1, and m is chosen large enough so that the total number of tasks in the height group is less than $m/4$. The area group consists of q shelves of height 2, with 2 tasks on each shelf: one of width $m/2$ and height $1 + 4/m$, and the other of width $m/4 + 1$ and height 2. (This is the shelf assignment produced by 2-SMART$_{\text{FFIA}}$.) All shelves k in both groups have $\mathcal{H}_k/\mathcal{W}_k = 1$, so the ordering of the shelves is immaterial.

On the other hand, the optimal schedule consists of the height group tasks placed side by side with three stacks that each contain $q/6$ of the narrower tasks in the area group. On top of these three stacks are placed side by side a stack of the $q/2$ remaining narrow tasks, and a stack of the q wide tasks. Then one can show that

$$\lim_{q,r \to \infty} \frac{R_{\text{FFIA}}(T)}{R^*(T)} = \frac{2 \cdot 4^r + 2q^2 + 4q2^r}{\frac{2}{3}4^r + \frac{4}{3}q^2}. \tag{185}$$

Now choosing $q = q_0 2^r$, we have

$$\lim_{r \to \infty} \frac{R_{\text{FFIA}}(T)}{R^*(T)} = \frac{2 + 2q_0^2 + 4q_0}{\frac{2}{3} + \frac{4}{3}q_0^2}. \tag{186}$$

Therefore, if we choose $q_0 = \frac{1}{2}$, then the result is

$$\lim_{r\to\infty} \frac{R_{\text{FFIA}}(T)}{R^*(T)} = \frac{9}{2}. \tag{187}$$

For the weighted case, there is a similar family of nonmalleable task sets for which 1.718-SMART$_{\text{NFIW}}$ produces a ratio $R_{\text{NFIW}}(T)/R^*(T)$ approaching approximately 6.757 asymptotically. Once again, there is a height group and an area group. The tasks in the height group fit onto r shelves of heights $\gamma, \gamma^2, \gamma^3, \ldots, \gamma^r$, with two tasks on each shelf. On the shelf of height γ^k there is one task of height γ^k and weight ϵ; the other task is of height slightly greater than γ^{k-1} and weight $\gamma^k - \epsilon$, where ϵ is near 0. All tasks in the height group have width 1. Note that the total number of tasks in the height group is $2r$. We will choose m such that $m \gg r$. The area group consists of q shelves of height 1, with 2 tasks on each shelf: one of width $(m-2r)/2$, height slightly greater than $1/\gamma$, and weight $(m-2r)/(m+2r+2)$, and the other of width $2r+1$, height 1, and weight $2(2r+1)/(m+2r+2)$. (This is the shelf assignment produced by γ-SMART$_{\text{NFIW}}$.) Once again, all shelves k in both groups have $\mathcal{H}_k/\mathcal{W}_k = 1$, so the ordering of the shelves is immaterial.

For $m \gg r$, the optimal schedule consists of the height group tasks placed side by side with two stacks that each contain $q/2$ area group tasks of width $(m-2r)/2$. The tasks of the area group of width $2r+1$ are started after the completion of the wider tasks. Then one can show that

$$\lim_{q,r\to\infty} \frac{R_{\text{NFIW}}(T)}{R^*(T)} = \frac{\frac{\gamma(2\gamma-1)}{(\gamma-1)(\gamma^2-1)}\gamma^{2r} + \frac{1}{2}q^2 + \frac{\gamma}{\gamma-1}q\gamma^r}{\frac{\gamma}{\gamma^2-1}\gamma^{2r} + \frac{1}{4\gamma}q^2}. \tag{188}$$

Now choosing $q = q_0\gamma^r$ yields

$$\lim_{r\to\infty} \frac{R_{\text{NFIW}}(T)}{R^*(T)} = \frac{\frac{\gamma(2\gamma-1)}{(\gamma-1)(\gamma^2-1)} + \frac{1}{2}q_0^2 + \frac{\gamma}{\gamma-1}q_0}{\frac{\gamma}{\gamma^2-1} + \frac{1}{4\gamma}q_0^2}. \tag{189}$$

We now choose

$$q_0 = \frac{2\gamma^2 - 4\gamma + 1 + \sqrt{8\gamma^4 \quad 16\gamma^3 + 16\gamma^2 - 8\gamma + 1}}{\gamma^2 - 1} \approx 2.475612, \tag{190}$$

and the result is

$$\lim_{r\to\infty} \frac{R_{\text{NFIW}}(T)}{R^*(T)} \approx 6.7570742. \tag{191}$$

4.5 A Practical Tip

There is a simple modification of our algorithms that is likely to result in improved schedules. In particular, we have used shelf schedules only for the purpose of analysis. There is no reason why we should delay starting the tasks on a particular shelf until every task on the preceding shelf is completed. It is enough if no task begins execution until every task on the preceding shelves has begun execution. This will guarantee a schedule that is at least as good as the SMART schedule. The following algorithm produces schedules that meet this criterion. Use SMART to pack the tasks onto shelves and to order the shelves. Then begin with the first shelf. Whenever processors are free, schedule the first task on the current shelf whose processor requirement does not exceed the number of free processors. When all the tasks on the current shelf have been scheduled, go on to the next shelf.

Chapter 5

Scheduling to Minimize Makespan

Note: The results reported in this chapter were obtained jointly with Tiwari and are reported in [37], as well as in a paper that has been submitted to SIAM Journal on Computing.

Many algorithms for scheduling nonmalleable tasks to minimize makespan (NMM) already exist, including algorithms for NMM on a line of processors. Therefore, we need only to provide algorithms for appropriately defined allotment selection problems in order to transform these existing NMM algorithms into MM algorithms. Our task is simplified by the fact that the approximation factors for many of the existing NMM algorithms are based on the same lower bound. In Section 5.1, we present this lower bound and use it to define an allotment selection problem. Then in Section 5.2, we obtain algorithms for MM by providing algorithms for solving this allotment selection problem.

5.1 The Lower Bound

A common lower bound used in minimum makespan problems is the maximum of two lower bounds: the maximum execution time of any task, and total work divided by the number of processors m. As before, we will let T denote a set of malleable tasks, and $T(\bar{p})$ denote the set of nonmalleable tasks corresponding to the task set T and the allotment $\bar{p}$. Let

$$h_T(\bar{p}) = \max_i\{t_i(p_i)\} \tag{192}$$

be the maximum execution time of a task in $T(\bar{p})$, and let

$$u_T(\bar{p}) = \frac{1}{m}\sum_{i=1}^{n} p_i t_i(p_i) \tag{193}$$

be the total work divided by the number of processors available. Then

$$\omega_T(\bar{p}) = \max\{u_T(\bar{p}), h_T(\bar{p})\} \tag{194}$$

is a lower bound on makespan for the nonmalleable task set $T(\bar{p})$, and

$$\omega_T = \min_{\bar{p}\in M}\{\omega_T(\bar{p})\} \tag{195}$$

is a lower bound on makespan for the malleable task set T, where $M = M_1 \times M_2 \times \cdots \times M_n$ denotes the set of all possible allotments of processors to T. Observe that (195) defines a makespan allotment selection problem.

We will say that an NMM algorithm is *ρ-bounded* if for any nonmalleable task set $T(\bar{p})$ it constructs a schedule with makespan not exceeding $\rho\omega_T(\bar{p})$. Similarly, an MM algorithm is ρ-bounded if for any malleable task set T it constructs a schedule with makespan not exceeding $\rho\omega_T$. Then a *ρ-bounded* algorithm for either problem is also a *ρ-approximate* algorithm.

GMM or MMM?	processors	running time	work
MMM	n	$O(\log^2(mn))$	$O(n\log^2(mn))$
GMM	n	$O(m+\log^2 n)$	$O(mn+n\log^2 n)$
GMM	$mn\log m/\log^2(mn)+n$	$O(\log^2(mn))$	$O(mn\log m+n\log^2 n)$
GMM	mn	$O(\log(mn))$	$O(mn\log(mn))$

Table 3: EREW PRAM algorithms for makespan allotment selection

5.2 Allotment Selection

An algorithm for the makespan allotment selection problem will allow us to transform any ρ-bounded NMM algorithm into a ρ-bounded (and hence ρ-approximate) MM algorithm. Since the best known NMM algorithms for lines and otherwise are ρ-bounded, we can duplicate for MM the approximation factors of these NMM algorithms using the following theorem.

Theorem 5.2.1 *There is an algorithm for the makespan allotment selection problem with running time $O(mn)$. If the tasks satisfy the nondecreasing work and nonincreasing execution time conditions, then the running time is $O(n\log^2 m)$. There are also EREW PRAM algorithms for the makespan allotment selection problem as shown in Table 3.*

Corollary 5.2.1 *Suppose we are given a ρ-bounded NMM algorithm with running time $Q(m,n)$. Then there is a ρ-bounded GMM algorithm with running time $O(mn)+Q(m,n)$, and a ρ-bounded MMM algorithm with running time $O(n\log^2 m)+Q(m,n)$. If the NMM algorithm applies to scheduling on a line, then so do the GMM and MMM algorithms.*

Proof: For any malleable task set T, we can use the algorithm of Theorem 5.2.1 to find an allotment $\bar{p}$ such that $\omega_T(\bar{p})=\omega_T$. Then we can use the ρ-bounded NMM algorithm

to produce a schedule for $T(\bar{p})$ with makespan not greater than $\rho\omega_T(\bar{p}) = \rho\omega_T$. □

The immediate consequences of Corollary 5.2.1 are summarized in Table 2. The MM algorithms in this table are obtained using the NMM algorithms due to Garey and Graham [22], Steinberg [53], and Sleator [51].

The extension of an NMM algorithm to a GMM algorithm given by Corollary 5.2.1 is nearly optimal in the following sense. Any NMM instance can also be formulated as an MMM instance, and therefore any algorithm for MMM or GMM can also be applied to NMM. Now suppose that any ρ-approximate algorithm for NMM has running time $\Omega(Q^*(m,n))$. Then any ρ-approximate algorithm for MMM or GMM also has running time $\Omega(Q^*(m,n))$. Furthermore, an MM algorithm must inspect every value of the function t_i for each task i if it is to guarantee a constant approximation factor. Therefore, its running time is $\Omega(mn + Q^*(m,n))$. The running time of a GMM algorithm based on our makespan allotment selection algorithm comes to within a constant factor of reaching this bound, assuming that there is a *ρ-bounded* NMM algorithm with running time $O(Q^*(m,n))$.

In the rest of this section we give a proof of Theorem 5.2.1. We will first show how to transform a GMM instance into an MMM instance, and then show how to solve the allotment selection problem for MMM. We will then consider how these methods can be adapted to run on an EREW PRAM.

5.2.1 Transforming a GMM Instance into an MMM Instance

The first step in our makespan allotment selection algorithm is to transform a GMM instance into an MMM instance by discarding allotments that violate the nondecreasing work and nonincreasing execution time conditions. In particular, for each task i, create

of possible allotments M_i' by discarding from M_i any allotment to task i for which possible either to reduce the number of processors without increasing the execution , or to increase the number of processors without increasing the work. That is, e $j \prec_i k$ if $j < k$ and $t_i(j) \leq t_i(k)$, or $j > k$ and $jt_i(j) \leq kt_i(k)$, and let

$$M_i' = \{k \in M_i : \nexists j \in M_i \text{ such that } j \prec_i k\}. \tag{196}$$

t $v[i,1] < v[i,2] < \cdots < v[i,z[i]]$ be all the elements of M_i', and let $t[i,j]$ denote e execution time of task i when $v[i,j]$ processors are allotted to it. That is, $t[i,j] = (v[i,j])$. Then for all $i \in [n]$ and $j \in [z[i]-1]$, we have

$$t[i,j] > t[i,j+1] \tag{197}$$

and

$$v[i,j] \cdot t[i,j] < v[i,j+1] \cdot t[i,j+1]. \tag{198}$$

Thus we have an instance of MMM. The following lemma guarantees that this procedure does not discard the optimal allotment.

Lemma 5.2.1 *Let $\bar{p}$ be any allotment for the malleable task set T with $p_i \in M_i \setminus M_i'$ for some $i \in [n]$. Then there is an allotment $\bar{q}$ for T such that $q_i \in M_i'$ and $\omega_T(\bar{q}) \leq \omega_T(\bar{p})$.*

To prove this lemma, we will make use of an alternate characterization of the relation $\prec_i$.

Lemma 5.2.2 *$j \prec_i k$ if and only if $t_i(j) \leq t_i(k)$ and $jt_i(j) \leq kt_i(k)$, with at least one of the inequalities being strict.*

Proof: Suppose that $j \prec_i k$. Then either $j < k$ and $t_i(j) \leq t_i(k)$, or else $j > k$ and $jt_i(j) \leq kt_i(k)$. If $j < k$ and $t_i(j) \leq t_i(k)$, then $jt_i(j) < kt_i(k)$. If instead $j > k$ and $jt_i(j) \leq kt_i(k)$, then $t_i(j) < t_i(k)$. This completes the "only if" part.

For the "if" part, suppose that $t_i(j) \leq t_i(k)$ and $jt_i(j) \leq kt_i(k)$. Since at least one of the inequalities is strict, we have $j \neq k$. Then it follows directly that $j \prec_i k$. □

The next lemma is an immediate consequence of Lemma 5.2.2.

Lemma 5.2.3 *The relation $\prec_i$ has the following properties:*

1. *if $j \prec_i k$, then $k \not\prec_i j$;*

2. *if $j \prec_i k$ and $k \prec_i l$, then $j \prec_i l$.*

Proof of Lemma 5.2.1: By the definition of M_i', for any $k \in M_i \setminus M_i'$, there exists $j \in M_i$ such that $j \prec_i k$. By Lemma 5.2.3, there can be no cycles such that $k_1 \prec_i k_2 \prec_i \cdots \prec_i k_r \prec_i k_1$. We conclude that if $k \in M_i \setminus M_i'$, then there exists $j \in M_i'$ such that $j \prec_i k$.

So define an allotment $\bar{q}$ for T by taking $q_l = p_l$ for all $l \neq i$, and choosing $q_i \in M_i'$ such that $q_i \prec_i p_i$. By Lemma 5.2.2, we have $t_i(q_i) \leq t_i(p_i)$ and $q_i t_i(q_i) \leq p_i t_i(p_i)$. Then $h_T(\bar{q}) \leq h_T(\bar{p})$ and $u_T(\bar{q}) \leq u_T(\bar{p})$, and therefore $\omega_T(\bar{q}) \leq \omega_T(\bar{p})$. □

Now if we let $M' = M_1' \times M_2' \times \cdots \times M_n'$, then the following lemma, which is a direct consequence of Lemma 5.2.1, demonstrates that an optimal allotment is preserved.

Lemma 5.2.4

$$\min_{\bar{p} \in M'} \{\omega_T(\bar{p})\} = \min_{\bar{p} \in M} \{\omega_T(\bar{p})\}. \tag{199}$$

Each set M_i' can be computed in $O(m)$ time, so the total running time of this procedure is $O(mn)$. If we begin with an MMM instance, then it is unnecessary.

5.2.2 Choosing a Target

The main idea in selecting an allotment for an MMM instance is to restrict the possible choices of $\bar{p}$, and in so doing to transform $\omega_T(\bar{p})$ into a function of a single variable.

us begin by rewriting the bound given in (195) as

$$\begin{aligned}\omega_T &= \min_{\bar{p}\in M'}\{\omega_T(\bar{p})\} \\ &= \min_{\bar{p}\in M'}\max\{u_T(\bar{p}), h_T(\bar{p})\} \\ &= \min_{\tau\in\mathbb{R}}\min_{\{\bar{p}\in M':h_T(\bar{p})=\tau\}}\max\{u_T(\bar{p}),\tau\} \\ &= \min_{\tau\in\mathbb{R}}\max\{\tau, \min_{\{\bar{p}\in M':h_T(\bar{p})=\tau\}} u_T(\bar{p})\}.\end{aligned} \tag{200}$$

It follows from (200) that we can restrict τ to the set

$$X = \{h_T(\bar{p}) : \bar{p} \in M'\}. \tag{201}$$

We can view τ as a *target* for the value of ω_T: given a target τ, we consider only allotments $\bar{p}$ such that $h_T(\bar{p}) = \tau$, and attempt to minimize $u_T(\bar{p})$ subject to this restriction. For this purpose, for all $\tau \in X$, let

$$j_i(\tau) = \min\{j : t[i,j] \le \tau\}. \tag{202}$$

Then $v[i, j_i(\tau)]$ is the minimum number of processors that can be assigned to task i so that its execution time is not more than τ. Now a target $\tau \in X$ determines an allotment $\bar{p}$ by taking $p_i = v[i, j_i(\tau)]$ for all $i \in [n]$. The following lemma shows that this is the desired allotment.

Lemma 5.2.5 *Given $\tau \in X$, define an allotment $\bar{p}$ by taking $p_i = v[i, j_i(\tau)]$. Then $h_T(\bar{p}) = \tau$, and $u_T(\bar{p}) = \min_{\{\bar{q}\in M':h_T(\bar{q})=\tau\}}\{u_T(\bar{q})\}$.*

Proof: To see that $h_T(\bar{p}) = \tau$, first observe that since $\tau \in X$, there exist i_0 and j_0 such that $t[i_0, j_0] = \tau$. Now there can be no $j < j_0$ for which $t[i_0, j] < t[i_0, j_0]$, as that would violate the nonincreasing execution time condition. It follows that $t[i_0, j_{i_0}(\tau)] = \tau$. Since $t[i, j_i(\tau)] \le \tau$ for all $i \in [n]$, we conclude that $h_T(\bar{p}) = \tau$.

Now consider any allotment $\bar{q} \in M'$. We will show that either $h_T(\bar{q}) > \tau$, or else $u_T(\bar{q}) \geq u_T(\bar{p})$. Fix some $i \in [n]$. Since $\bar{q} \in M'$, there is some j such that $q_i = v[i,j]$. Suppose that $j < j_i(\tau)$. Then by (202), we have $t_i(q_i) = t[i,j] > \tau$, and therefore $h_T(\bar{q}) > \tau$. Suppose instead that $j \geq j_i(\tau)$. Then by the nondecreasing work condition, we have

$$q_i t_i(q_i) = v[i,j] \cdot t[i,j] \geq v[i,j_i(\tau)] \cdot t[i,j_i(\tau)] = p_i t_i(p_i). \tag{203}$$

□

We conclude that if $h_T(\bar{q}) \leq \tau$, then $u_T(\bar{q}) \geq u_T(\bar{p})$.

For a target $\tau \in X$ and an allotment $\bar{p}$ given by $p_i = v[i,j_i(\tau)]$, define

$$W(\tau) = u_T(\bar{p}) = \frac{1}{m} \sum_{i=1}^{n} v[i,j_i(\tau)] \cdot t[i,j_i(\tau)]. \tag{204}$$

Then it follows directly from Lemma 5.2.5 that

$$W(\tau) = \min_{\{\bar{p} \in M' : h_T(\bar{p}) = \tau\}} \{u_T(\bar{p})\}. \tag{205}$$

Now (200) can be written as

$$\omega_T = \min_{\tau \in X} \max\{\tau, W(\tau)\}. \tag{206}$$

We will view the makespan allotment selection problem in terms of (206) rather than (195). That is, rather than seeking an allotment $\bar{p}$ that satisfies $\omega_T(\bar{p}) = \omega_T$, we will seek a target τ that satisfies $\max\{\tau, W(\tau)\} = \omega_T$. Once we have found such a τ, then taking $p_i = v[i,j_i(\tau)]$ will yield an allotment $\bar{p}$ satisfying $\omega_T(\bar{p}) = \max\{h_T(\bar{p}), u_T(\bar{p})\} = \max\{\tau, W(\tau)\} = \omega_T$.

Now that we have transformed $\omega_T(\bar{p})$ into a function of a single variable, namely $\max\{\tau, W(\tau)\}$, we turn our attention to locating the minimum of this function.

Lemma 5.2.6 *$W(\tau)$ is a nonincreasing function of τ.*

roof: Let $\tau' \geq \tau$, where $\tau, \tau' \in X$. Then for all tasks i, we have $j_i(\tau') \leq j_i(\tau)$. Then t follows from the nondecreasing work condition that $W(\tau') \leq W(\tau)$. □

Lemma 5.2.7 *Let $\tau \in X$ be such that $\tau \geq W(\tau)$. Then for all $\tau' \in X$ such that $\tau' > \tau$, we have* $\max\{\tau', W(\tau')\} > \max\{\tau, W(\tau)\}$.

Proof: By Lemma 5.2.6, $W(\tau) \geq W(\tau')$, and so $\tau' > \tau \geq W(\tau) \geq W(\tau')$. Therefore, $\max\{\tau', W(\tau')\} = \tau' > \tau = \max\{\tau, W(\tau)\}$. □

Lemma 5.2.8 *Let $\tau \in X$ be such that $\tau < W(\tau)$. Then for all $\tau' \in X$ such that $\tau' \leq \tau$, we have* $\max\{\tau', W(\tau')\} \geq \max\{\tau, W(\tau)\}$.

Proof: By Lemma 5.2.6, $W(\tau) \leq W(\tau')$, and so $\tau' \leq \tau < W(\tau) \leq W(\tau')$. Therefore, $\max\{\tau', W(\tau')\} = W(\tau') \geq W(\tau) = \max\{\tau, W(\tau)\}$. □

Lemmas 5.2.7 and 5.2.8 suggest the following strategy for computing ω_T. Since the arrays $t[i, *]$ are sorted, they can be merged in time $O(mn \log n)$ to obtain the set X in decreasing order. Then a binary search on this sequence can be used to find the value of τ that minimizes $\max\{\tau, W(\tau)\}$ as in (206). The binary search requires $O(\log(mn))$ probes. Each probe takes $O(n \log m)$ steps, because evaluating $W(\tau)$ requires n binary searches on sorted lists of length at most m. Therefore, the total running time of this algorithm is dominated by the merge, and is $O(mn \log n)$. We now present an alternative algorithm for computing ω_T that has running time $O(n \log^2 m)$.

5.2.3 A Target Selection Algorithm

As above, we restrict τ to be in the set X, but in this algorithm we do not merge the arrays $t[i, *]$. Rather, we keep them separate, and for each array we maintain a range from which the value of τ may be selected. We say that a task is *active* if its range

is nonempty. Otherwise, it is *inactive*. The idea is to select the next τ value so as to cut the range in half for at least half of the active tasks. The algorithm is given in Figure 7. Note that the variable **target** corresponds to τ, and that for a given value of **target**, **tallot**$[i]$ corresponds to j_i(**target**), and **work**$/m$ corresponds to W(**target**). At any given time during the algorithm's execution, the set of potential target values is given by $\{t[i,j] : \textsf{lower}[i] \leq j \leq \textsf{upper}[i]\}$.

Correctness

In order to demonstrate the algorithm's correctness, we have two goals. One is to show that a potential target is eliminated only if there is a better one. The other is to show that $\max\{\tau, W(\tau)\}$ is improving for both "successful" targets ($\tau \geq W(\tau)$) and "unsuccessful" targets ($\tau < W(\tau)$) as the algorithm progresses. These goals are stated more formally in the three lemmas that follow.

Let τ_k denote the value that is chosen for **target** in the kth iteration of the main loop.

Lemma 5.2.9 *At the end of the kth iteration, $t[i_0, j_0]$ is not an element of the current set of potential target values if and only if it satisfies one of the following three conditions:*

1. $\nexists \bar{p}$ *such that* $h_T(\bar{p}) = t[i_0, j_0]$;
2. $\exists l \leq k$ *such that* $\tau_l \leq t[i_0, j_0]$ *and* $\tau_l \geq W(\tau_l)$;
3. $\exists l \leq k$ *such that* $\tau_l \geq t[i_0, j_0]$ *and* $\tau_l < W(\tau_l)$.

1. **for** each i $\in [n]$ **do** $t[i,0] := \infty$, $t[i,z[i]+1] := 0$;
2. low $:= \max_i\{t[i,z[i]]\}$;
3. **for** each i $\in [n]$ **do** lower[i] $:= 1$, upper[i] $:= \max\{j : t[i,j] \geq \text{low}\}$;
4. successful $:= \infty$; unsuccessful $:= \infty$;
5. **repeat**
   (a) **for** each i such that lower[i] $\leq$ upper[i] **do** mid[i] $:= \lfloor$(lower[i] + upper[i])$/2\rfloor$;
   (b) Let target be the median of the set $\{t[i, \text{mid}[i]] : \text{lower}[i] \leq \text{upper}[i]\}$;
   (c) **for** each i $\in [n]$ **do** tallot[i] $:= \min\{j : t[i,j] \leq \text{target}\}$;
   (d) work $:= \sum_{i=1}^{n} v[i, \text{tallot}[i]] * t[i, \text{tallot}[i]]$;
   (e) achievedbound $:= \max\{\text{work}/m, \text{target}\}$;
   (f) **if** work$/m \leq$ target **then**
      i. successful $:=$ achievedbound;
      ii. **for** each i $\in [n]$ **do** sallot[i] $:=$ tallot[i];
      iii. **for** each i such that lower[i] $\leq$ upper[i] **do** lower[i] $:= \min\{j : t[i,j] < \text{target}\}$;
   (g) **else**
      i. unsuccessful $:=$ achievedbound;
      ii. **for** each i $\in [n]$ **do** uallot[i] $:=$ tallot[i];
      iii. **for** each i such that lower[i] $\leq$ upper[i] **do** upper[i] $:= \max\{j : t[i,j] > \text{target}\}$;
6. **until** lower[i] $>$ upper[i] for all i $\in [n]$;
7. **if** successful $<$ unsuccessful **then for** each i $\in [n]$ **do** allot[i] $:=$ sallot[i] **else for** each i $\in [n]$ **do** allot[i] $:=$ uallot[i];

Figure 7: Solving the makespan allotment selection problem. An efficient implementation requires keeping a list of active tasks.

Lemma 5.2.10 *Let iterations k_1 and k_2 be any two iterations such that $\tau_{k_1} \geq W(\tau_{k_1})$ and $\tau_{k_2} \geq W(\tau_{k_2})$, and suppose that $k_1 < k_2$. Then $\max\{\tau_{k_1}, W(\tau_{k_1})\} > \max\{\tau_{k_2}, W(\tau_{k_2})\}$.*

Lemma 5.2.11 *Let iterations k_1 and k_2 be any two iterations such that $\tau_{k_1} < W(\tau_{k_1})$ and $\tau_{k_2} < W(\tau_{k_2})$, and suppose that $k_1 < k_2$. Then $\max\{\tau_{k_1}, W(\tau_{k_1})\} \geq \max\{\tau_{k_2}, W(\tau_{k_2})\}$.*

Proof of Lemma 5.2.9: The values in the arrays lower and upper are altered only in Steps 3, 5(f)iii, and 5(g)iii; these are the only steps in which potential targets are eliminated. Each of these three steps corresponds to one of the three conditions in the lemma. For each condition in turn, we will show that a potential target is eliminated in the corresponding step if and only if it satisfies the condition.

First we will show that a time $t[i_0, j_0]$ is eliminated in Step 3 if and only if it satisfies Condition 1. The shortest possible execution time of task i is $t[i, z[i]]$. Therefore, for any allotment $\bar{p}$ and any task i, we have $h_T(\bar{p}) \geq t[i, z[i]]$. That is, any allotment $\bar{p}$ satisfies $h_T(\bar{p}) \geq \max_i\{t[i, z[i]]\} = \mathsf{low}$. Now observe that if $t[i_0, j_0]$ is eliminated in Step 3, then it satisfies $t[i_0, j_0] < \mathsf{low}$. We conclude that if a time $t[i_0, j_0]$ is eliminated in Step 3, then it satisfies Condition 1.

If $t[i_0, j_0]$ is not eliminated in Step 3, then it satisfies $t[i_0, j_0] \geq \max_i\{t[i, z[i]]\}$. Construct an allotment $\bar{p}$ as follows. Let $p_{i_0} = v[i_0, j_0]$, and let $p_i = v[i, z[i]]$, for all $i \neq i_0$. Then $h_T(\bar{p}) = t[i_0, j_0]$, and therefore $t[i_0, j_0]$ does not satisfy Condition 1.

Next we show that a time $t[i_0, j_0]$ has been eliminated in Step 5(f)iii by the end of iteration k if and only if there exists $l \leq k$ such that $\tau_l \leq t[i_0, j_0]$ and $\tau_l \geq W(\tau_l)$. Suppose that time $t[i_0, j_0]$ is eliminated in Step 5(f)iii of iteration $l \leq k$. The fact that Step 5(f)iii is executed in iteration l indicates that the test in Step 5f is true;

that is, $\tau_l \geq W(\tau_l)$. Furthermore, Step 5(f)iii of iteration l only eliminates times $t[i,j]$ satisfying $t[i,j] \geq \tau_l$. We conclude that if $t[i_0,j_0]$ is eliminated in Step 5(f)iii of iteration $l \leq k$, then $\tau_l \leq t[i_0,j_0]$ and $\tau_l \geq W(\tau_l)$.

Now suppose that there exists $l \leq k$ such that $\tau_l \leq t[i_0,j_0]$ and $\tau_l \geq W(\tau_l)$. Then in iteration l, the test in Step 5f is true because $\tau_l \geq W(\tau_l)$. After Step 5(f)iii, $t[i_0,\mathsf{lower}[i_0]] < \tau_l \leq t[i_0,j_0]$. Then $j_0 < \mathsf{lower}[i_0]$, and so $t[i_0,j_0]$ has been eliminated. Therefore, time $t[i_0,j_0]$ is eliminated in Step 5(f)iii of some iteration not later than the kth iteration if and only if it satisfies Condition 2.

Finally we show that a time $t[i_0,j_0]$ has been eliminated in Step 5(g)iii by the end of iteration k if and only if there exists $l \leq k$ such that $\tau_l \geq t[i_0,j_0]$ and $\tau_l < W(\tau_l)$. Suppose that time $t[i_0,j_0]$ is eliminated in Step 5(g)iii of iteration $l \leq k$. The fact that Step 5(g)iii is executed in iteration l indicates that the test in Step 5f is false; that is, $\tau_l < W(\tau_l)$. Furthermore, Step 5(g)iii of iteration l only eliminates times $t[i,j]$ satisfying $t[i,j] \leq \tau_l$. We conclude that if $t[i_0,j_0]$ is eliminated in Step 5(g)iii of iteration $l \leq k$, then $\tau_l \geq t[i_0,j_0]$ and $\tau_l < W(\tau_l)$.

Now suppose that there exists $l \leq k$ such that $\tau_l \geq t[i_0,j_0]$ and $\tau_l < W(\tau_l)$. Then in iteration l, the test in Step 5f is false because $\tau_l < W(\tau_l)$. After Step 5(g)iii, $t[i_0,\mathsf{upper}[i_0]] > \tau_l \geq t[i_0,j_0]$. Then $j_0 > \mathsf{upper}[i_0]$, and so $t[i_0,j_0]$ has been eliminated. Therefore, time $t[i_0,j_0]$ is eliminated in Step 5(g)iii of some iteration not later than the kth iteration if and only if it satisfies Condition 3. □

To prove Lemmas 5.2.10 and 5.2.11, we make use of the following lemmas.

Lemma 5.2.12 *Let iteration k_1 be any iteration such that $\tau_{k_1} \geq W(\tau_{k_1})$. Let k_2 be any iteration that follows k_1. Then $\tau_{k_2} < \tau_{k_1}$.*

Proof: Suppose that $\tau_{k_2} \geq \tau_{k_1}$. Then τ_{k_2} is eliminated from the set of potential targets

by the end of iteration k_1, as it satisfies Condition 2 of Lemma 5.2.9. Then τ_{k_2} can not be the value of **target** in a subsequent iteration, which is a contradiction. □

Lemma 5.2.13 *Let iteration k_1 be any iteration such that $\tau_{k_1} < W(\tau_{k_1})$. Let k_2 be any iteration that follows k_1. Then $\tau_{k_2} > \tau_{k_1}$.*

Proof: Suppose that $\tau_{k_2} \leq \tau_{k_1}$. Then τ_{k_2} is eliminated from the set of potential targets by the end of iteration k_1, as it satisfies Condition 3 of Lemma 5.2.9. Then τ_{k_2} can not be the value of **target** in a subsequent iteration, which is a contradiction. □

Proof of Lemma 5.2.10: By Lemma 5.2.12, we have $\tau_{k_2} < \tau_{k_1}$. Then the result follows directly from Lemma 5.2.7. □

Proof of Lemma 5.2.11: By Lemma 5.2.13, we have $\tau_{k_2} > \tau_{k_1}$. Then the result follows directly from Lemma 5.2.8. □

Lemma 5.2.14 *The algorithm of Figure 7 correctly computes ω_T and finds an allotment $\bar{p}$ such that $\omega_T = \omega_T(\bar{p})$.*

Proof: By Lemma 5.2.9, when the main loop is entered, the set of potential targets is the set of possible values of $h_T(\bar{p})$. As we have already noted, (200) shows that we need not consider any other values as potential targets. Furthermore, by Lemmas 5.2.7, 5.2.8, and 5.2.9, a potential target τ is eliminated only if there exists τ_l such that $\max\{\tau_l, W(\tau_l)\} \leq \max\{\tau, W(\tau)\}$. Now at least one potential target is eliminated in each iteration, and the algorithm does not terminate until all potential targets have been eliminated. Therefore, there is some iteration l such that $\max\{\tau_l, W(\tau_l)\} = \min_{\tau \in X} \max\{\tau, W(\tau)\} = \omega_T$.

Let iteration s be the last iteration satisfying $\tau_s \geq W(\tau_s)$. Let iteration u be the last iteration satisfying $\tau_u < W(\tau_u)$. (The value of $\max\{\tau_s, W(\tau_s)\}$ is stored in the

ariable **successful**, and the corresponding allotment is stored in the array **sallot**, while he value of $\max\{\tau_u, W(\tau_u)\}$ is stored in the variable **unsuccessful**, and the corresponding llotment is stored in the array **uallot**.) Now consider any iteration k. If $\tau_k \geq W(\tau_k)$, then by Lemma 5.2.10, we have $\max\{\tau_k, W(\tau_k)\} \geq \max\{\tau_s, W(\tau_s)\}$. If instead $\tau_k < W(\tau_k)$, then by Lemma 5.2.11, we have $\max\{\tau_k, W(\tau_k)\} \geq \max\{\tau_u, W(\tau_u)\}$. Therefore,

$$\begin{aligned} \max\{\tau_k, W(\tau_k)\} &\geq \min\{\max\{\tau_s, W(\tau_s)\}, \max\{\tau_u, W(\tau_u)\}\} \\ &= \min\{\textsf{successful}, \textsf{unsuccessful}\}. \end{aligned} \tag{207}$$

We conclude that $\omega_T = \min\{\textsf{successful}, \textsf{unsuccessful}\}$, and that the allotment computed by the algorithm, $\bar{p} = (v[1, \textsf{allot}[1]], v[2, \textsf{allot}[2]], \ldots, v[n, \textsf{allot}[n]])$, satisfies $\omega_T = \omega_T(\bar{p})$. □

Running Time

Before we analyze the running time, observe that the algorithm can be modified slightly so that the running time of each iteration of the main loop depends on the number of active tasks rather than on n. This is possible because once a task has become inactive, its allotment is determined, according to the following lemma.

Lemma 5.2.15 *Suppose that task i becomes inactive at the end of iteration l. Then for any iteration k following l, we have* $\min\{j : t[i,j] \leq \tau_k\} = \min\{j : t[i,j] \leq \tau_{l+1}\}$.

Proof: Recall that by definition $j_i(\tau) = \min\{j : t[i,j] \leq \tau\}$. Our goal is to show that for any iteration k following l, we have $t[i, j_i(\tau_{l+1})] < \tau_k < t[i, j_i(\tau_{l+1}) - 1]$. From this the lemma follows immediately.

The task i is inactive at the end of iteration l, and therefore at that time all of the values in the array $t[i,*]$ have been eliminated as potential target values. We will first

show $t[i, j_i(\tau_{l+1})] < \tau_k$ by considering what caused $t[i, j_i(\tau_{l+1})]$ to be eliminated as a potential target, and then show $\tau_k < t[i, j_i(\tau_{l+1}) - 1]$ using the same approach.

Consider in turn each of the three possible causes given in Lemma 5.2.9 of the elimination of $t[i, j_i(\tau_{l+1})]$ as a potential target. First suppose that there is no allotment $\bar{p}$ satisfying $h_T(\bar{p}) = t[i, j_i(\tau_{l+1})]$. Then for every iteration k, we have $\tau_k > t[i, j_i(\tau_{l+1})]$, since all of the potential targets that are eliminated in Step 3 are less than all of the potential targets that are not eliminated in that step.

Next suppose that there exists $a \leq l$ such that $\tau_a \leq t[i, j_i(\tau_{l+1})]$ and $\tau_a \geq W(\tau_a)$. By the definition of $j_i(\tau)$, we have $t[i, j_i(\tau_{l+1})] \leq \tau_{l+1}$. Before iteration $l+1$, $t[i, j_i(\tau_{l+1})]$ has already been eliminated as a potential target, and so $t[i, j_i(\tau_{l+1})] \neq \tau_{l+1}$. We conclude that $t[i, j_i(\tau_{l+1})] < \tau_{l+1}$, and therefore also $\tau_a < \tau_{l+1}$, contradicting Lemma 5.2.12.

Finally suppose that there exists $b \leq l$ such that $\tau_b \geq t[i, j_i(\tau_{l+1})]$ and $\tau_b < W(\tau_b)$. Then by Lemma 5.2.13, for all $k \geq l+1$, we have $\tau_k > \tau_b$. Therefore, $\tau_k > t[i, j_i(\tau_{l+1})]$.

We conclude that in any case, for all $k \geq l+1$, we have $\tau_k > t[i, j_i(\tau_{l+1})]$.

Now consider in turn each of the three possible causes of the elimination of $t[i, j_i(\tau_{l+1}) - 1]$ as a potential target. First suppose that there is no allotment $\bar{p}$ such that $h_T(\bar{p}) = t[i, j_i(\tau_{l+1}) - 1]$. Then every target τ selected by the algorithm satisfies $\tau > t[i, j_i(\tau_{l+1}) - 1]$. But by the definition of $j_i(\tau)$, we have $t[i, j_i(\tau_{l+1}) - 1] > \tau_{l+1}$, a contradiction.

Next suppose that there exists $a \leq l$ such that $\tau_a \leq t[i, j_i(\tau_{l+1}) - 1]$ and $\tau_a \geq W(\tau_a)$. Then by Lemma 5.2.12, for all $k \geq l+1$, we have $\tau_k < \tau_a$, and therefore $\tau_k < t[i, j_i(\tau_{l+1}) - 1]$.

Finally suppose that there exists $b \leq l$ such that $\tau_b \geq t[i, j_i(\tau_{l+1}) - 1]$ and $\tau_b < W(\tau_b)$. By the definition of $j_i(\tau)$, we have $t[i, j_i(\tau_{l+1}) - 1] > \tau_{l+1}$, and therefore also $\tau_b > \tau_{l+1}$,

□

ing Lemma 5.2.13.

nclude that for all $k \geq l+1$, we have $\tau_k < t[i, j_i(\tau_{l+1}) - 1]$.

this lemma, we can make the following modification to the algorithm without its correctness. A list of the active tasks is maintained. Tasks that become at the end of an iteration are not removed immediately from the list, but rather re marked. In each iteration, after Step 5c, all marked tasks are removed from t of active tasks. With this modification, the allotment to inactive tasks is not puted in each iteration, and so the running time of each iteration is not affected active tasks.

he algorithm's most time-consuming steps are 5c, 5(f)iii, and 5(g)iii. These all ire binary search for each of the active tasks and can be completed in $O(\log m)$ per k. Note that in Step 5b, the median of the set can be found in time proportional to number of active tasks using a selection algorithm as in [1]. So the running time of e algorithm is $O(\log m)$ per active task per iteration. Now our goal is to bound the m of the number of active tasks over all the iterations of the algorithm's main loop. We will refer to the set of potential targets associated with an active task as its *range*.

The following lemma takes the first step toward our goal.

Lemma 5.2.16 *In each iteration of the main loop, at least half of the elements are eliminated from the ranges of at least half of the active tasks.*

Proof: In each iteration, the value of **target** is chosen to be the median of the medians of the ranges. Suppose that in some iteration, **target** $\geq W($**target**$)$. For at least half of the active tasks, the median of the range is not less than **target**. So for each of these tasks, at least half of the values in its range are not less than **target**. By Lemma 5.2.9, all of these values are eliminated at the end of the iteration.

Similarly, suppose that in some iteration, target $< W$(target). For at least half of the active tasks, the median of the range is not greater than target. So for each of these tasks, at least half of the values in its range are not greater than target. By Lemma 5.2.9, all of these values are eliminated at the end of the iteration. □

Now we can obtain an upper bound on the total number of active tasks over all iterations by considering the following (k, l)-*pruning game*. We are given k items. In each *round* of the game, at least half of the items take a *hit*. After an item has taken l hits, it becomes inactive. When an item becomes inactive, it no longer participates in the game.

Lemma 5.2.17 *The total number of active items summed over all rounds of a* (k, l)*-pruning game does not exceed* $2kl$.

Proof: Let k_i denote the number of active items at the beginning of round i. (Then $k_1 = k$.) Let r denote the number of rounds it takes to eliminate all of the items. Then we wish to get an upper bound on $\sum_{i=1}^{r} k_i$.

In round i, at least $k_i/2$ hits are distributed. Thus the total number of hits that are distributed during the game is at least $\frac{1}{2}\sum_{i=1}^{r} k_i$. Each item can take at most l hits, so the total number of hits distributed during the game is at most kl. Thus we have $\frac{1}{2}\sum_{i=1}^{r} k_i \leq kl$, and the lemma follows immediately. □

A task becomes inactive after it takes $\lfloor \log m \rfloor + 1$ hits, i.e., after the size of its range has been cut in half $\lfloor \log m \rfloor + 1$ times. Therefore, by Lemma 5.2.17, the total number of active tasks summed over all iterations is at most $2n(\lfloor \log m \rfloor + 1)$. Thus the total running time of the algorithm of Figure 7 is $O(n \log^2 m)$.

This completes the proof that the makespan allotment selection problem can be solved in $O(n \log^2 m)$ when the tasks satisfy the nondecreasing work and nonincreasing

ecution time conditions. If the tasks do not satisfy these conditions, then as we ave seen in Section 5.2.1, an additional $O(mn)$ is required, for a total running time of $)(mn)$.

5.2.4 The PRAM Algorithms

The algorithm of Figure 7 can easily be adapted to run on an EREW PRAM. Observe that the only steps in which the tasks are not treated independently are Steps 2, 5b, 5d, and 6. Each of these steps can be completed in $O(\log n)$ time on an n-processor EREW PRAM. The remaining steps can each be completed in $O(\log m)$ time, for a total running time of $O(\log(mn))$ per iteration of the main loop. However, in order to reduce the number of iterations of the main loop, we will modify Step 5b to find the *weighted* median.

The weighted median problem is as follows. We are given a collection of n elements $a_1, a_2, \ldots, a_n$ drawn from a totally ordered set. Each element a_i has an associated weight u_i. Let $U = \sum_{i=1}^{n} u_i$ be the sum of the weights. Then the goal is to find $b \in \{a_1, \ldots, a_n\}$ such that $\sum_{a_i \geq b} u_i \geq U/2$ and $\sum_{a_i \leq b} u_i \geq U/2$.

If in Step 5b we assign weight $\mathsf{upper}[i] - \mathsf{lower}[i] + 1$ to the element $t[i, \mathsf{mid}[i]]$ for each i satisfying $\mathsf{lower}[i] \leq \mathsf{upper}[i]$, and then choose for a target the weighted median of the set $\{t[i, \mathsf{mid}[i]] : \mathsf{lower}[i] \leq \mathsf{upper}[i]\}$, then at least one quarter of the potential target values will be eliminated per iteration. Therefore, the number of iterations of the main loop is $O(\log(mn))$.

One way of computing the weighted median of n items is to first sort the items, and then compute the prefix sums of the weights. Each of these operations can be completed in $O(\log n)$ time on an n-processor EREW PRAM, using algorithms due

to Cole [15] and Ladner and Fischer [33]. (See Karp and Ramachandran's survey [29] for a discussion of parallel selection and sorting.) So after making this change to Step 5b, the running time per iteration remains $O(\log(mn))$. Therefore, we have a makespan allotment selection algorithm for MMM with running time $O(\log^2(mn))$ on an n-processor EREW PRAM.

To obtain a makespan allotment selection algorithm for GMM, we can do the transformation from GMM to MMM in parallel. Note that in this transformation, the sets M_i' are computed independently, and the time required is $O(m)$ per task on a single processor. Then we have an n-processor EREW PRAM algorithm with running time $O(m + \log^2(mn)) = O(m + \log^2 n)$.

If more processors are available, then we can speed up the transformation from GMM to MMM. In order to enforce decreasing execution times, we discard every allotment j to task i for which $t_i(j) \geq \min_{k\in[j-1]}\{t_i(k)\}$. For each task i, the values $\min_{k\in[j-1]}\{t_i(k)\}$ can be computed for all $j \in M_i$ in $O(\log m)$ steps using m processors by applying a prefix operation. We can use a similar approach to enforce increasing work. Therefore, all n sets M_i' can be computed in $O(\log m)$ steps using mn processors. But since the allotment selection step has running time $O(\log^2(mn))$, we can use fewer processors and slow down the transformation without affecting the overall running time. Thus we have an EREW PRAM algorithm with running time $O(\log^2(mn))$ using $mn \log m / \log^2(mn) + n$ processors.

We can reduce the running time by adopting a different approach after the transformation from GMM to MMM has been applied. Let

$$u[i,j] = \begin{cases} v[i,z[i]] \cdot t[i,z[i]] & \text{if } j = z[i] \\ v[i,j] \cdot t[i,j] - v[i,j+1] \cdot t[i,j+1] & \text{if } j < z[i]. \end{cases} \tag{208}$$

Now the algorithm is as follows. Sort the potential targets $t[i,j]$ in nondecreasing order. Compute the prefix sums of the associated values of $u[i,j]$. Then for each potential target $\tau \in \{t[i,j]\}$ satisfying $\tau \geq \max_{k\in[n]}\{t[k,z[k]]\}$, the associated prefix sum is the minimum total work required if every task has execution time not exceeding τ:

$$\begin{aligned}\sum_{\{(i,k):t[i,k]\leq\tau\}} u[i,k] &= \sum_{i=1}^{n}\sum_{k=j_i(\tau)}^{z[i]} u[i,k] \\ &= \sum_{i=1}^{n} v[i,j_i(\tau)]\cdot t[i,j_i(\tau)] \\ &= mW(\tau). \end{aligned} \tag{209}$$

As noted above, sorting [15] and computing prefix sums [33] of mn items can both be done in $O(\log(mn))$ steps on an mn-processor EREW PRAM. Since we now have a sorted list of all potential targets τ along with the corresponding values of $W(\tau)$, we can in constant time determine for all $t[i,j]$ whether $t[i,j] \leq \tau^*$ or $t[i,j] > \tau^*$, where τ^* is a target satisfying $\max\{\tau^*, W(\tau^*)\} = \omega_T$. Then an allotment $\bar{p}$ satisfying $\omega_T(\bar{p}) = \omega_T$ can be constructed in constant time by taking $p_i = v[i,j]$ if $t[i,j] \leq \tau^* < t[i,j-1]$. Therefore, we have an mn-processor EREW PRAM algorithm with running time $O(\log(mn))$.

Chapter 6

Conclusions and Directions for Future Work

We have presented a general approach to scheduling malleable tasks that involves separating the problem into two parts: choosing an allotment of processors to tasks, and scheduling the resulting nonmalleable tasks. We have demonstrated how this method can be applied both to minimizing average completion time and to minimizing makespan. For each of these objectives, we have defined an allotment selection problem and provided algorithms that produce exact or approximate solutions to this problem. We have also described and analyzed three algorithms for scheduling nonmalleable tasks to minimize average completion time. These results lead to several directions for future work, including the following.

1. Extension to specific parallel architectures.

   The problem of scheduling malleable tasks can be extended so that some network topology is specified for the m processors, and each task is scheduled on a subset

of processors with a particular interconnection network, rather than on an arbitrary subset [21, 20]. (Scheduling on a line of processors is one example of this.) With the exception of the semi-oblivious algorithm, our allotment selection algorithms for both average completion time and makespan apply to this extended problem. Therefore, an algorithm for scheduling nonmalleable tasks on a particular architecture can be extended to an algorithm for scheduling malleable tasks on the same architecture. Garey and Graham's [22] list scheduling algorithm can be adjusted to apply to hypercubes by arranging the tasks in nonincreasing order of processor requirement, rather than taking them in an arbitrary order. This yields a 2-approximate algorithm for MM on a hypercube. Li and Cheng [36] give a 46/7-approximate algorithm for packing in three dimensions, which is identical to NMM on a 2-d mesh. The lower bound ω_T is used to prove the approximation factor, and therefore we have a 46/7-approximate algorithm for MM on a 2-d mesh. It remains open to obtain results for NMWACT on these architectures, and for both NMM and NMWACT on other architectures, including the 3-d mesh.

2. Complexity of weighted allotment selection.

   For the weighted average completion time objective, no polynomial-time exact algorithm for the allotment selection problem is known. On the other hand, neither is the problem known to be NP-complete. An interesting open problem is to resolve the complexity of allotment selection for weighted tasks.

3. Complexity of unweighted allotment selection.

   For the unweighted average completion time objective, we have shown that the allotment selection problem can be solved exactly in $O(n^3+mn)$ steps. In order to get faster algorithms, we have settled for approximations. Another open problem

is to find an exact algorithm for allotment selection with running time $o(n^3+mn)$ — or, on the other hand, to show that an $o(n^3+mn)$ allotment selection algorithm leads to an $o(n^3)$ weighted bipartite matching algorithm.

4. Improved analysis of the greedy and semi-oblivious algorithms.

   The following conjecture would yield an improved upper bound on the optimization problem (97).

   Conjecture 6.0.1

$$\max_{\bar{a},\bar{b},\pi}\left\{\frac{\sum a_i}{F(\bar{a},\bar{b},\pi)}\right\} = \max_{\pi}\left\{\frac{n}{F(\bar{1},\bar{1},\pi)}\right\}. \tag{210}$$

   It can be shown that $\min_\pi\{F(\bar{1},\bar{1},\pi)\} > (1-1/e)n$, so from the conjecture we would conclude that the greedy algorithm has approximation factor $e/(e-1) \approx 1.582$, and the semi-oblivious algorithm has approximation factor $2e/(e-1) \approx 3.164$. This results in improved approximation factors for several of our MACT and MWACT algorithms.

5. Improved approximation factor for NMM.

   While Garey and Graham [22] have shown that the approximation factor of 2 for list scheduling is tight when the tasks are taken in an arbitrary order, it may be possible to obtain a better approximation factor by arranging the tasks in a particular way. Ordering the tasks by nonincreasing execution time appears to be a good candidate. Note that unless $P = NP$, there is no ρ-approximate algorithm for NMM with $\rho < 3/2$. (The reduction is from PARTITION with unit task execution times.)

6. Improved approximation factor for NMWACT.

There is a substantial gap between the approximation factors we have shown for 2-$\text{SMART}_{\text{FFIA}}$ and γ-$\text{SMART}_{\text{NFIW}}$ and the worst case examples we constructed for those algorithms. Closing this gap, preferably by improved analysis, is desirable.

Appendix A

Proof of Lemma 3.2.1

Proof of Lemma 3.2.1: Pick some task i_0. For each $j \in [m]$, let ℓ_j denote the line corresponding to the function $C_{i_0}(\mu, j)$. Let $x(j,k)$ denote the x-coordinate of the intersection of the lines ℓ_j and ℓ_k. That is, $C_{i_0}(x(j,k),j) = C_{i_0}(x(j,k),k)$. Let $\Psi(j)$ denote the slope of ℓ_j. Then $\Psi(j) = \theta j t_{i_0}(j)/m$.

Note that if $v_{i_0 j} = v_{i_0,j+1}$, then the interval corresponding to j processors is empty. The algorithm in Figure 8 computes a list of the nonempty intervals for task i_0. Upon termination, the values of j for which the corresponding intervals $(v_{i_0,j+1}, v_{i_0 j}]$ are nonempty are given by $\mathsf{d}[1] < \mathsf{d}[2] < \cdots < \mathsf{d}[\mathsf{r}]$. The interval corresponding to $\mathsf{d}[k]$ is given by $(\mathsf{v}[k+1], \mathsf{v}[k]]$. That is, $C_{i_0}(\mu) = C_{i_0}(\mu, \mathsf{d}[k])$ for all $\mu \in (\mathsf{v}[k+1], \mathsf{v}[k]]$.

For any iteration j, let the subscript j attached to a variable in the algorithm indicate the value of that variable at the end of iteration j of the main loop (Step 2). Then, for example, r_j is the number of entries in the array d after iteration j. The following lemma establishes the algorithm's correctness.

Lemma A.0.18 *For any $j \in [m]$, we have*

1. $\mathsf{r} := 0$;
2. **for** $\mathsf{j} := 1$ to m **do**
   (a) **while** $\mathsf{r} > 0$ and $\{\Psi(\mathsf{j}) \leq \Psi(\mathsf{d}[\mathsf{r}])$ or $x(\mathsf{j}, \mathsf{d}[\mathsf{r}]) \geq \mathsf{v}[\mathsf{r}]\}$ **do** $\mathsf{r} := \mathsf{r} - 1$;
   (b) **if** $\mathsf{r} = 0$ **then**
      i. $\mathsf{r} := \mathsf{r} + 1$;
      ii. $\mathsf{d}[\mathsf{r}] := \mathsf{j}$;
      iii. $\mathsf{v}[\mathsf{r}] := \sum_{i=1}^{n} w_i$; $\mathsf{v}[\mathsf{r}+1] := 0$;
   (c) **else if** $x(\mathsf{j}, \mathsf{d}[\mathsf{r}]) > 0$ **then**
      i. $\mathsf{r} := \mathsf{r} + 1$;
      ii. $\mathsf{d}[\mathsf{r}] := \mathsf{j}$;
      iii. $\mathsf{v}[\mathsf{r}] := x(\mathsf{j}, \mathsf{d}[\mathsf{r}-1])$; $\mathsf{v}[\mathsf{r}+1] := 0$;

Figure 8: Computing the intervals $(v_{i_0,j+1}, v_{i_0 j}]$ on which $C_{i_0}(\mu) = C_{i_0}(\mu, j)$

1. $v_j[r_j + 1] = 0$, $v_j[1] = \sum_{i=1}^{n} w_i$, *and* $v_j[k+1] < v_j[k]$ *for all* $k \in [r_j]$.

2. *For any* $\mu \in (0, \sum_{i=1}^{n} w_i]$, *let* k *be such that* $\mu \in (v_j[k+1], v_j[k]]$. *Then*

$$\min_{l \in [j]} \{C_{i_0}(\mu, l)\} = C_{i_0}(\mu, d_j[k]). \tag{211}$$

Note that by choosing $j = m$, it follows from the lemma that upon termination of the algorithm, $C_{i_0}(\mu) = C_{i_0}(\mu, \mathsf{d}[k])$ for all $\mu \in (\mathsf{v}[k+1], \mathsf{v}[k]]$.

Proof of Lemma A.0.18: We proceed by induction on j. After the first iteration, we have $\mathsf{r}_1 = 1$, $\mathsf{d}_1[1] = 1$, $\mathsf{v}_1[1] = \sum_{i=1}^{n} w_i$, and $\mathsf{v}_1[2] = 0$. Thus the lemma holds for $j = 1$.

Now we will show that the lemma holds for iteration $j > 1$, supposing that it holds for the previous iteration.

In order to prove the first part of the lemma, it is necessary to show that if the value of r is altered in Step 2a, then either the test in Step 2b or the test in Step 2c is true. Let r_j^- denote the value of r in iteration j upon termination of the while loop in

Step 2a. Then we need to prove the following claim[1].

Claim A.0.1 *If* $r_j^- < r_{j-1}$, *then* $r_j^- = 0$ *or* $x(j, d_{j-1}[r_j^-]) > 0$.

Observe that if v is updated in iteration j, then it is updated either in Step 2(b)iii, or else in Step 2(c)iii. By Claim A.0.1, if r is altered in iteration j, then v is also altered. Therefore, to prove the first part of the lemma, we need only consider the possible cases in which v is altered. Suppose that v is updated in Step 2(b)iii. Then $\mathsf{r}_j = 1$, and so $0 = \mathsf{v}_j[2] < \mathsf{v}_j[1] = \sum_{i=1}^n w_i$, as required. Suppose instead that v is updated in Step 2(c)iii. Then for $\mathsf{r} = \mathsf{r}_j^- = \mathsf{r}_j - 1$, the condition in Step 2c is true, the condition in Step 2b is false, and the while loop condition in Step 2a is false. From the first of these three facts, we have $\mathsf{v}_j[\mathsf{r}_j] = x(j, \mathsf{d}_{j-1}[\mathsf{r}_j^-]) > 0 = \mathsf{v}_j[\mathsf{r}_j + 1]$. The second and third of these facts together imply that $x(j, \mathsf{d}_{j-1}[\mathsf{r}_j^-]) < \mathsf{v}_{j-1}[\mathsf{r}_j^-] = \mathsf{v}_j[\mathsf{r}_j - 1]$. We conclude that $\mathsf{v}_j[\mathsf{r}_j] < \mathsf{v}_j[\mathsf{r}_j - 1]$. Since the other entries of v are unchanged, this completes the proof of the first part.

For the second part of the lemma, it is sufficient to show that at the end of j iterations, the following conditions are satisfied for all $\mu \in (0, \sum_{i=1}^n w_i]$. First, μ is in the interval corresponding to j processors if and only if $\min_{l \in [j]}\{C_{i_0}(\mu, l)\} = C_{i_0}(\mu, j)$. Second, if μ is not in the interval corresponding to j processors, then it is in the same interval that it was in after $j - 1$ iterations. These two conditions are stated more formally in the following pair of claims.

Claim A.0.2 *For any* $\mu \in (0, \sum_{i=1}^n w_i]$, *we have* $\mathsf{d}_j[\mathsf{r}_j] = j$ *and* $\mu \in (\mathsf{v}_j[\mathsf{r}_j + 1], \mathsf{v}_j[\mathsf{r}_j]]$ *if and only if* $\min_{l \in [j]}\{C_{i_0}(\mu, l)\} = C_{i_0}(\mu, j)$.

[1]The claims in this section are not self-contained: they rely upon the induction hypothesis for Lemma A.0.18.

Claim A.0.3 *For any* $\mu \in (0, \sum_{i=1}^{n} w_i]$, *let* k *be such that* $\mu \in (\mathsf{v}_{j-1}[k+1], \mathsf{v}_{j-1}[k]]$. *Then if* $\mathsf{d}_j[\mathsf{r}_j] = j$ *but* $\mu \notin (\mathsf{v}_j[\mathsf{r}_j + 1], \mathsf{v}_j[\mathsf{r}_j]]$, *then* $\mu \in (\mathsf{v}_j[k+1], \mathsf{v}_j[k]]$.

Now for the proof of the second part of Lemma A.0.18, pick any $\mu \in (0, \sum_{i=1}^{n} w_i]$, and let k be such that $\mu \in (\mathsf{v}_j[k+1], \mathsf{v}_j[k]]$. (Note that the first part of the lemma guarantees that such a k exists.)

First consider the case $\mathsf{d}_j[\mathsf{r}_j] \neq j$. In this case, d and v are not changed in iteration j. It also follows from Claim A.0.2 that $C_{i_0}(\mu, j) > \min_{l \in [j]}\{C_{i_0}(\mu, l)\}$. Thus we have

$$\begin{aligned} \min_{l \in [j]}\{C_{i_0}(\mu, l)\} &= \min_{l \in [j-1]}\{C_{i_0}(\mu, l)\} \\ &= C_{i_0}(\mu, \mathsf{d}_{j-1}[k]) \\ &= C_{i_0}(\mu, \mathsf{d}_j[k]). \end{aligned} \tag{212}$$

Next consider the case $\mathsf{d}_j[\mathsf{r}_j] = j$ and $k = \mathsf{r}_j$. Then by Claim A.0.2 we have

$$\min_{l \in [j]}\{C_{i_0}(\mu, l)\} = C_{i_0}(\mu, j) = C_{i_0}(\mu, \mathsf{d}_j[k]). \tag{213}$$

Finally, suppose that $\mathsf{d}_j[\mathsf{r}_j] = j$ and $k < \mathsf{r}_j$. The only entry of d that is modified in iteration j is $\mathsf{d}[\mathsf{r}_j]$, and therefore $\mathsf{d}_j[k] = \mathsf{d}_{j-1}[k]$. Since $k \neq \mathsf{r}_j$, it follows from Claim A.0.2 that $C_{i_0}(\mu, j) > \min_{l \in [j]}\{C_{i_0}(\mu, l)\}$. It also follows from Claim A.0.3 that $\mu \in (\mathsf{v}_{j-1}[k+1], \mathsf{v}_{j-1}[k]]$. Then we have

$$\begin{aligned} \min_{l \in [j]}\{C_{i_0}(\mu, l)\} &= \min_{l \in [j-1]}\{C_{i_0}(\mu, l)\} \\ &= C_{i_0}(\mu, \mathsf{d}_{j-1}[k]) \\ &= C_{i_0}(\mu, \mathsf{d}_j[k]). \end{aligned} \tag{214}$$

□

Before presenting proofs of the three claims above, we will complete the proof of Lemma 3.2.1 by showing that the algorithm of Figure 8 has running time $O(m)$. First

we will bound the number of iterations of the while loop in Step 2a. Observe that the variable r is altered only by either adding 1 or subtracting 1. Its initial value is $\mathsf{r} = 0$, and it can be incremented at most once per iteration of the main loop (Step 2). Since r is decremented in each iteration of the while loop, and the loop test guarantees that $\mathsf{r} \geq 0$ at all times, we conclude that the total number of iterations of the while loop over the duration of the algorithm is at most $m-1$, with each iteration taking constant time. Excluding Step 2a, each of the m iterations of the main loop (Step 2) takes constant time. (Note that the time required to compute $\sum_{i=1}^{n} w_i$ can be apportioned over the n tasks so that it takes constant time per task.) Therefore the total time is $O(m)$. □

The four lemmas that follow are useful for proving Claims A.0.1, A.0.2, and A.0.3.

Lemma A.0.19 *If* $\Psi(k_1) < \Psi(k_2)$, *then* $C_{i_0}(\mu, k_1) > C_{i_0}(\mu, k_2)$ *for all* $\mu < x(k_1, k_2)$, *and* $C_{i_0}(\mu, k_1) < C_{i_0}(\mu, k_2)$ *for all* $\mu > x(k_1, k_2)$.

Proof: We have

$$\begin{aligned} C_{i_0}(\mu, k_2) - C_{i_0}(\mu, k_1) &= [C_{i_0}(x(k_1, k_2), k_2) + \Psi(k_2) \cdot (\mu - x(k_1, k_2))] \\ &\quad - [C_{i_0}(x(k_1, k_2), k_1) + \Psi(k_1) \cdot (\mu - x(k_1, k_2))] \\ &= (\Psi(k_2) - \Psi(k_1))(\mu - x(k_1, k_2)), \end{aligned} \tag{215}$$

and the result follows directly. □

Lemma A.0.20 *If* $r_j^- > 0$, *then* $\Psi(j) > \Psi(d_{j-1}[r_j^-])$.

Proof: When the while loop in Step 2a terminates, by definition we have $\mathsf{r} = \mathsf{r}_j^-$, and therefore the loop condition is false for $\mathsf{r} = \mathsf{r}_j^-$. Then the result follows from the fact that $\mathsf{r}_j^- > 0$. □

Lemma A.0.21 *If* $d_j[r_j] = j$ *and* $r_j^- > 0$, *then* $v_j[r_j] = x(j, d_{j-1}[r_j^-]) > 0$.

Proof: Since $\mathsf{d}_j[\mathsf{r}_j] = j$, either the condition in Step 2b is true in iteration j, or else the condition in Step 2c is true in iteration j. Now $\mathsf{r}_j^- > 0$, so it is the latter condition that is true, and thus $x(j, \mathsf{d}_{j-1}[\mathsf{r}_j^-]) > 0$. Then following the assignments to the array v, the value of $\mathsf{v}_j[\mathsf{r}_j]$ will be as required. □

The following lemma states that an entry k is removed from the array d in iteration j only if it is not superior to j anywhere in its interval.

Lemma A.0.22 *If* $k \leq r_{j-1}$ *and* $k > r_j^-$, *then* $C_{i_0}(\mu, j) \leq C_{i_0}(\mu, d_{j-1}[k])$ *for all* $\mu \in (0, v_{j-1}[k]]$.

Proof: We will consider two cases. First suppose that $\Psi(j) \leq \Psi(\mathsf{d}_{j-1}[k])$. Observe that $C_{i_0}(\mu, j) = C_{i_0}(0, j) + \mu \cdot \Psi(j)$. Then it is sufficient to show that $C_{i_0}(0, j) \leq C_{i_0}(0, \mathsf{d}_{j-1}[k])$. By our assumption, we have $j t_{i_0}(j) \leq \mathsf{d}_{j-1}[k] t_{i_0}(\mathsf{d}_{j-1}[k])$. Then

$$\begin{aligned}
C_{i_0}(0, j) &= w_{i_0} t_{i_0}(j) \left[1 - \frac{1}{2}\theta - \frac{\theta j}{2m}\right] \\
&\leq w_{i_0} \frac{\mathsf{d}_{j-1}[k] t_{i_0}(\mathsf{d}_{j-1}[k])}{j} \left[1 - \frac{1}{2}\theta - \frac{\theta j}{2m}\right] \\
&= w_{i_0} t_{i_0}(\mathsf{d}_{j-1}[k]) \left[\frac{\mathsf{d}_{j-1}[k]}{j}(1 - \frac{1}{2}\theta) - \frac{\theta \mathsf{d}_{j-1}[k]}{2m}\right] \\
&< w_{i_0} t_{i_0}(\mathsf{d}_{j-1}[k]) \left[1 - \frac{1}{2}\theta - \frac{\theta \mathsf{d}_{j-1}[k]}{2m}\right] \\
&= C_{i_0}(0, \mathsf{d}_{j-1}[k]). \qquad (216)
\end{aligned}$$

We conclude that $C_{i_0}(\mu, j) \leq C_{i_0}(\mu, \mathsf{d}_{j-1}[k])$ for all $\mu \geq 0$.

Suppose instead that $\Psi(j) > \Psi(\mathsf{d}_{j-1}[k])$. By Lemma A.0.19, we have $C_{i_0}(\mu, j) \leq C_{i_0}(\mu, \mathsf{d}_{j-1}[k])$ for all $\mu \leq x(j, \mathsf{d}_{j-1}[k])$. Since $k > \mathsf{r}_j^-$, we know that the while loop test in Step 2a is true for $\mathsf{r} = k$. This implies that $x(j, \mathsf{d}_{j-1}[k]) \geq \mathsf{v}_{j-1}[k]$. We conclude that $C_{i_0}(\mu, j) \leq C_{i_0}(\mu, \mathsf{d}_{j-1}[k])$ for all $\mu \leq \mathsf{v}_{j-1}[k]$. □

We are now ready to proceed with the proofs of Claims A.0.1, A.0.2, and A.0.3.

Proof of Claim A.0.1: Suppose to the contrary that $0 < \mathsf{r}_j^- < \mathsf{r}_{j-1}$ and $x(j, \mathsf{d}_{j-1}[\mathsf{r}_j^-]) \leq 0$. By Lemma A.0.20, we have $\Psi(j) > \Psi(\mathsf{d}_{j-1}[\mathsf{r}_j^-])$. This and Lemma A.0.19 together imply that

$$C_{i_0}(\mu, j) > C_{i_0}(\mu, \mathsf{d}_{j-1}[\mathsf{r}_j^-]) \text{ for all } \mu > x(j, \mathsf{d}_{j-1}[\mathsf{r}_j^-]). \tag{217}$$

Now since $\mathsf{r}_j^- < \mathsf{r}_{j-1}$, by Lemma A.0.22 we have

$$C_{i_0}(\mu, j) \leq C_{i_0}(\mu, \mathsf{d}_{j-1}[\mathsf{r}_{j-1}]) \text{ for all } \mu \in (0, \mathsf{v}_{j-1}[\mathsf{r}_{j-1}]]. \tag{218}$$

By induction we have $\mathsf{v}_{j-1}[\mathsf{r}_{j-1}] > 0$, and therefore $\mathsf{v}_{j-1}[\mathsf{r}_{j-1}] > x(j, \mathsf{d}_{j-1}[\mathsf{r}_j^-])$. Now it follows from (217) and (218) that

$$C_{i_0}(\mathsf{v}_{j-1}[\mathsf{r}_{j-1}], \mathsf{d}_{j-1}[\mathsf{r}_{j-1}]) > C_{i_0}(\mathsf{v}_{j-1}[\mathsf{r}_{j-1}], \mathsf{d}_{j-1}[\mathsf{r}_j^-]), \tag{219}$$

contradicting the induction hypothesis. □

Proof of Claim A.0.2: Pick any $\mu \in (0, \sum_{i=1}^n w_i]$. For the "only if" part, suppose that $\mathsf{d}_j[\mathsf{r}_j] = j$ and $\mu \in (\mathsf{v}_j[\mathsf{r}_j + 1], \mathsf{v}_j[\mathsf{r}_j]]$. Let k be such that $\mu \in (\mathsf{v}_{j-1}[k+1], \mathsf{v}_{j-1}[k]]$. (The first part of Lemma A.0.18 guarantees that such a k exists.) Then by the induction hypothesis, we have $C_{i_0}(\mu, \mathsf{d}_{j-1}[k]) = \min_{l \in [j-1]}\{C_{i_0}(\mu, l)\}$. Also note that $\mathsf{d}_j[\mathsf{r}_j] = j$ implies that $\mathsf{r}_j = \mathsf{r}_j^- + 1$.

We will consider three cases, starting with $k > \mathsf{r}_j^-$. In this case, it follows from Lemma A.0.22 and the induction hypothesis that

$$C_{i_0}(\mu, j) \leq C_{i_0}(\mu, \mathsf{d}_{j-1}[k]) = \min_{l \in [j-1]}\{C_{i_0}(\mu, l)\}. \tag{220}$$

Next suppose that $k < \mathsf{r}_j^-$. Observe that the only entries of v that are modified in iteration j are $\mathsf{v}[\mathsf{r}_j]$ and $\mathsf{v}[\mathsf{r}_j + 1]$. Therefore $\mathsf{v}_j[k] = \mathsf{v}_{j-1}[k]$ and $\mathsf{v}_j[k+1] = \mathsf{v}_{j-1}[k+1]$,

and so $\mu \in (\mathsf{v}_j[k+1], \mathsf{v}_j[k]]$. This is a contradiction: μ belongs to two intervals that do not overlap. Note that by the first part of Lemma A.0.18, $\mathsf{v}_j[\mathsf{r}_j] < \mathsf{v}_j[k+1]$, since $\mathsf{r}_j > k+1$.

Finally, consider the case $k = \mathsf{r}_j^-$. By Lemma A.0.20, we have $\Psi(j) > \Psi(\mathsf{d}_{j-1}[\mathsf{r}_j^-])$. Also, by Lemma A.0.21 we have $\mathsf{v}_j[\mathsf{r}_j] = x(j, \mathsf{d}_{j-1}[\mathsf{r}_j^-])$. Therefore $\mu \leq x(j, \mathsf{d}_{j-1}[\mathsf{r}_j^-])$. Then using Lemma A.0.19 and the induction hypothesis, we conclude that

$$C_{i_0}(\mu, j) \leq C_{i_0}(\mu, \mathsf{d}_{j-1}[\mathsf{r}_j^-]) = C_{i_0}(\mu, \mathsf{d}_{j-1}[k]) = \min_{l \in [j-1]} \{C_{i_0}(\mu, l)\}. \tag{221}$$

Now for the "if" part we will first consider the case $\mathsf{d}_j[\mathsf{r}_j] \neq j$, and then the case $\mathsf{d}_j[\mathsf{r}_j] = j$ and $\mu \notin (\mathsf{v}_j[\mathsf{r}_j + 1], \mathsf{v}_j[\mathsf{r}_j]]$, and in both cases show that $C_{i_0}(\mu, j) > \min_{l \in [j]} \{C_{i_0}(\mu, l)\}$.

First suppose that $\mathsf{d}_j[\mathsf{r}_j] \neq j$. Then for $\mathsf{r} = \mathsf{r}_j^-$, the conditions in Steps 2b and 2c are both false. That is, $\mathsf{r}_j^- > 0$ and $x(j, \mathsf{d}_{j-1}[\mathsf{r}_j^-]) \leq 0$. Then from Claim A.0.1 we have $\mathsf{r}_j^- = \mathsf{r}_{j-1}$. Also, by Lemma A.0.20 we have $\Psi(j) > \Psi(\mathsf{d}_{j-1}[\mathsf{r}_{j-1}])$. Since $\mu > 0 \geq x(j, \mathsf{d}_{j-1}[\mathsf{r}_{j-1}])$, using Lemma A.0.19 we conclude that $C_{i_0}(\mu, j) > C_{i_0}(\mu, \mathsf{d}_{j-1}[\mathsf{r}_{j-1}]) \geq \min_{l \in [j]} \{C_{i_0}(\mu, l)\}$.

Next suppose that $\mathsf{d}_j[\mathsf{r}_j] = j$ and $\mu \notin (\mathsf{v}_j[\mathsf{r}_j + 1], \mathsf{v}_j[\mathsf{r}_j]]$. Note that $\mathsf{v}_j[\mathsf{r}_j + 1] = 0$, and so it follows that $\mu > \mathsf{v}_j[\mathsf{r}_j]$. Now by the first part of Lemma A.0.18, $\mathsf{v}_j[1] = \sum_{i=1}^n w_i$, and therefore $\mathsf{r}_j > 1$. Then $\mathsf{r}_j^- \geq 1$, and so by Lemma A.0.20 it follows that $\Psi(j) > \Psi(\mathsf{d}_{j-1}[\mathsf{r}_j^-])$. Also, by Lemma A.0.21 we have $\mathsf{v}_j[\mathsf{r}_j] = x(j, \mathsf{d}_{j-1}[\mathsf{r}_j^-])$. Therefore $\mu > x(j, \mathsf{d}_{j-1}[\mathsf{r}_j^-])$, and so it follows from Lemma A.0.19 that $C_{i_0}(\mu, j) > C_{i_0}(\mu, \mathsf{d}_{j-1}[\mathsf{r}_{j-1}]) \geq \min_{l \in [j]} \{C_{i_0}(\mu, l)\}$. □

Proof of Claim A.0.3: We will suppose that $\mathsf{d}_j[\mathsf{r}_j] = j$, and prove that $\mu \in (\mathsf{v}_j[\mathsf{r}_j + 1], \mathsf{v}_j[\mathsf{r}_j]]$ or $\mu \in (\mathsf{v}_j[k+1], \mathsf{v}_j[k]]$. We consider three cases.

For the first case, suppose that $k < \mathsf{r}_j^-$. Note that because $\mathsf{d}_j[\mathsf{r}_j] = j$, we have

$\mathsf{r}_j = \mathsf{r}_j^- + 1$. Observe that the only entries of v that are modified in iteration j are $\mathsf{v}[\mathsf{r}_j^- + 1]$ and $\mathsf{v}[\mathsf{r}_j^- + 2]$. Therefore $\mathsf{v}_j[k+1] = \mathsf{v}_{j-1}[k+1]$ and $\mathsf{v}_j[k] = \mathsf{v}_{j-1}[k]$, and so $\mu \in (\mathsf{v}_j[k+1], \mathsf{v}_j[k]]$.

For the second case, suppose that $k = \mathsf{r}_j^-$. Then $\mathsf{v}_j[k] = \mathsf{v}_{j-1}[k]$ is not modified in iteration j. Then we have $\mathsf{v}_j[\mathsf{r}_j + 1] = 0 \leq \mathsf{v}_{j-1}[k+1] < \mathsf{v}_{j-1}[k] = \mathsf{v}_j[k]$, and so $\mu \in (\mathsf{v}_j[\mathsf{r}_j + 1], \mathsf{v}_j[k]]$. Now since $\mathsf{r}_j = \mathsf{r}_j^- + 1 = k + 1$, we conclude that either $\mu \in (\mathsf{v}_j[\mathsf{r}_j + 1], \mathsf{v}_j[\mathsf{r}_j]]$ or $\mu \in (\mathsf{v}_j[k+1], \mathsf{v}_j[k]]$.

For the third and final case, suppose that $k > \mathsf{r}_j^-$. Then $k \geq \mathsf{r}_j$, so we have $\mathsf{v}_j[\mathsf{r}_j + 1] = 0 \leq \mathsf{v}_{j-1}[k+1] < \mathsf{v}_{j-1}[k] \leq \mathsf{v}_{j-1}[\mathsf{r}_j]$. Now if we can show that $\mathsf{v}_{j-1}[\mathsf{r}_j] \leq \mathsf{v}_j[\mathsf{r}_j]$, then we can conclude that $(\mathsf{v}_{j-1}[k+1], \mathsf{v}_{j-1}[k]] \subseteq (\mathsf{v}_j[\mathsf{r}_j + 1], \mathsf{v}_j[\mathsf{r}_j]]$, and therefore $\mu \in (\mathsf{v}_j[\mathsf{r}_j + 1], \mathsf{v}_j[\mathsf{r}_j]]$. So it only remains to show that $\mathsf{v}_j[\mathsf{r}_j] \geq \mathsf{v}_{j-1}[\mathsf{r}_j]$. We consider two cases.

If $\mathsf{r}_j^- = 0$, then the test in Step 2b is true, and so $\mathsf{v}_j[\mathsf{r}_j] = \sum_{i=1}^n w_i$. Furthermore, $\mathsf{r}_j = 1$, and so by the induction hypothesis, $\mathsf{v}_{j-1}[\mathsf{r}_j] = \mathsf{v}_{j-1}[1] = \sum_{i=1}^n w_i$. We conclude that $\mathsf{v}_j[\mathsf{r}_j] = \mathsf{v}_{j-1}[\mathsf{r}_j]$.

Suppose instead that $\mathsf{r}_j^- > 0$. Note that $\mathsf{r}_j^- < k \leq \mathsf{r}_{j-1}$. Then by Lemma A.0.22, we have

$$C_{i_0}(\mu, j) \leq C_{i_0}(\mu, \mathsf{d}_{j-1}[\mathsf{r}_j^- + 1]) \text{ for all } \mu \in (0, \mathsf{v}_{j-1}[\mathsf{r}_j^- + 1]]. \tag{222}$$

From the induction hypothesis, we have

$$C_{i_0}(\mu, \mathsf{d}_{j-1}[\mathsf{r}_j]) = \min_{l \in [j-1]} \{C_{i_0}(\mu, l)\} \text{ for all } \mu \in (\mathsf{v}_{j-1}[\mathsf{r}_j + 1], \mathsf{v}_{j-1}[\mathsf{r}_j]], \tag{223}$$

from which it follows directly that

$$C_{i_0}(\mu, \mathsf{d}_{j-1}[\mathsf{r}_j]) \leq C_{i_0}(\mu, \mathsf{d}_{j-1}[\mathsf{r}_j^-]) \text{ for all } \mu \in (\mathsf{v}_{j-1}[\mathsf{r}_j + 1], \mathsf{v}_{j-1}[\mathsf{r}_j]]. \tag{224}$$

Since $\mathsf{r}_j = \mathsf{r}_j^- + 1$, it follows from (222) and (224) that

$$C_{i_0}(\mathsf{v}_{j-1}[\mathsf{r}_j], j) \leq C_{i_0}(\mathsf{v}_{j-1}[\mathsf{r}_j], \mathsf{d}_{j-1}[\mathsf{r}_j^-]). \tag{225}$$

By Lemma A.0.20, we have

$$\Psi(j) > \Psi(\mathsf{d}_{j-1}[\mathsf{r}_j^-]). \tag{226}$$

Now (225) and (226) together with Lemma A.0.19 yield

$$\mathsf{v}_{j-1}[\mathsf{r}_j] \leq x(j, \mathsf{d}_{j-1}[\mathsf{r}_j^-]). \tag{227}$$

Since Lemma A.0.21 implies that

$$\mathsf{v}_j[\mathsf{r}_j] = x(j, \mathsf{d}_{j-1}[\mathsf{r}_j^-]), \tag{228}$$

we conclude from (227) and (228) that

$$\mathsf{v}_j[\mathsf{r}_j] \geq \mathsf{v}_{j-1}[\mathsf{r}_j], \tag{229}$$

as required. □

Bibliography

[1] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, 1974.

[2] B. Baker, D. Brown, and H. Katseff. A 5/4 algorithm for two-dimensional packing. *Journal of Algorithms*, 2(4):348–368, December 1981.

[3] B. Baker, E. Coffman, and R. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, November 1980.

[4] B. Baker and J. Schwarz. Shelf algorithms for two-dimensional packing problems. *SIAM Journal on Computing*, 12(3):508–525, August 1983.

[5] 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.

[6] J. Blazewicz, W. Cellary, R. Slowinski, and J. Weglarz. Scheduling under resource constraints — deterministic models. *Annals of Operations Research*, 7, 1986.

[7] J. Blazewicz, M. Drabowski, and J. Weglarz. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Transactions on Computers*, C-35(5):389–393, 1986.

[8] J. Blazewicz, K. Ecker, G. Schmidt, and J. Weglarz. *Scheduling in Computer and Manufacturing Systems.* Springer-Verlag, 1993.

[9] J. Blazewicz, W. Kubiak, H. Rock, and J. Szwarcfiter. Minimizing mean flow-time with parallel processors and resource constraints. *Acta Informatica*, 24:513–524, 1987.

[10] 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.

[11] J. Bruno, E. Coffman, Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17:382–387, 1974.

[12] S. H. Chiang, R. Mansharamani, and M. Vernon. Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, volume 22, 1994.

[13] V. Chvatal. *Linear Programming.* W. H. Freeman and Company, 1983.

[14] E. Coffman, M. Garey, D. Johnson, and R. Tarjan. Performance bounds for level-oriented two-dimensional packing problems. *SIAM Journal on Computing*, 9(4):808–826, November 1980.

[15] R. Cole. Parallel merge sort. *SIAM Journal on Computing*, 17:770–785, 1988.

[16] R. Conway, W. Maxwell, and L. Miller. *Theory of Scheduling.* Addison-Wesley, 1967.

[17] L. Dowdy. On the partitioning of multiprocessor systems. Technical report, Vanderbilt University, July 1988.

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

[19] 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.

[20] 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.

[21] A. Feldmann, J. Sgall, and S.-H. Teng. Dynamic scheduling on parallel machines. In *32nd Annual Symposium on Foundations of Computer Science*, pages 111–120, 1991.

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

[23] M. Garey and D. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4:397–411, 1975.

[24] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[25] D. Ghosal, G. Serazzi, and S. Tripathi. The processor working set and its use in scheduling multiprocessor systems. *IEEE Transactions on Software Engineering*, 17(5):443–453, May 1991.

[26] W. Horn. Minimizing average flow time with parallel machines. *Operations Research*, 21(3):846–847, 1973.

[27] D. Johnson. Near-optimal bin-packing algorithms. Technical Report TR-109, MIT, June 1973.

[28] D. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3):272–314, June 1974.

[29] R. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity.* The MIT Press/Elsevier, 1990.

[30] R. Krishnamurti and E. Ma. The processor partitioning problem in special-purpose partitionable systems. In *Proceedings of the 1988 International Conference on Parallel Processing, vol. I*, pages 434–443, 1988.

[31] R. Krishnamurti and B. Narahari. Preemptive scheduling of independent jobs on partitionable parallel architectures. In *Proceedings of the 1992 International Conference on Parallel Processing, vol. I*, pages 268–275, 1992.

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

[33] R. Ladner and M. Fischer. Parallel prefix computation. *Journal of the ACM*, 27:831–838, 1980.

[34] 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: Logistics of Production and Inventory.* North-Holland, 1993.

[35] S. Leutenegger and M. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, volume 18, pages 226–236, May 1990.

[36] K. Li and K. H. Cheng. On 3-dimensional packing. *SIAM Journal on Computing*, 19(5):847–867, October 1990.

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

[38] S. Majumdar, D. Eager, and R. Bunt. Scheduling in multiprogrammed parallel systems. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, volume 16, pages 104–113, May 1988.

[39] R. K. Mansharamani and M. K. Vernon. Approximate analysis of parallel processor allocation policies. Technical Report 1187, University of Wisconsin–Madison, Computer Sciences Department, November 1993.

[40] V. Naik, S. Setia, and M. Squillante. Performance analysis of job scheduling policies in parallel supercomputing environments. In *Supercomputing '93*, November 1993.

[41] V. Naik, S. Setia, and M. Squillante. Scheduling of large scientific applications on distributed memory multiprocessor systems. In *Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computation*, 1993.

[42] B. Narahari and R. Krishnamurti. Scheduling independent tasks on partitionable hypercube multiprocessors. In *7th International Parallel Processing Symposium*, pages 118–122, 1993.

[43] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization.* Prentice-Hall, 1982.

[44] V. Peris, M. Squillante, and V. Naik. Analysis of the impact of memory in distributed parallel processing systems. Technical Report RC 19336, IBM Research Division, October 1993.

[45] E. Rosti, E. Smirni, L. Dowdy, G. Serazzi, and B. Carlson. Robust partitioning policies of multiprocessor systems. *Performance Evaluation*, 19:141–165, 1994.

[46] U. Schwiegelshohn, W. Ludwig, J. Wolf, J. Turek, and P. Yu. Smart SMART bounds for weighted response time scheduling. Technical report, IBM T. J. Watson Research Center, 1994.

[47] S. Setia and S. Tripathi. An analysis of several processor partitioning policies for parallel computers. Technical Report CS-TR-2684, University of Maryland, May 1991.

[48] S. Setia and S. Tripathi. A comparative analysis of static processor partitioning policies for parallel computers. In *Proceedings of the International Workshop on Modeling and Simulation of Computer and Telecommunication Systems*, January 1993.

[49] K. Sevcik. Characterization of parallelism in applications and their use in scheduling. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, volume 17, pages 171–180, May 1989.

[50] H. Shachnai and J. Glasgow. Minimizing the flow time for parallelizable task systems. Technical Report TR790, Technion, November 1993.

[51] D. Sleator. A 2.5 times optimal algorithm for packing in two dimensions. *Information Processing Letters*, 10(1):37–40, February 1980.

[52] W. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[53] A. Steinberg. A strip packing algorithm with absolute performance bound 2. To appear in *SIAM Journal on Computing.*

[54] A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 159–166, 1989.

[55] J. Turek, W. Ludwig, J. L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, and P. S. Yu. Scheduling parallelizable tasks to minimize average response time. In *6th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1994.

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

[57] 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.

[58] R. Uzsoy. Scheduling a single batch processing machine with non-identical job sizes. *International Journal of Production Research*, 32(7):1615–1635, 1994.

[59] J. Weglarz. Multiprocessor scheduling with memory allocation — a deterministic approach. *IEEE Transactions on Computers*, C-29(8):703–709, August 1980.

[60] J. Zahorjan and C. McCann. Processor scheduling in shared memory multiprocessors. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 214–225, May 1990.