McMaster University

Advanced Optimization Laboratory

Title:

Scheduling Malleable Tasks

Authors:

Klaus Jansen and Hu Zhang

AdvOl-Report No. 2005/17

September 2005, Hamilton, Ontario, Canada

Scheduling Malleable Tasks*

Klaus Jansen† Hu Zhang‡

1 Introduction

Nowadays, parallel computing plays a key role in high performance computing and gradually replaces traditional super computers. In a parallel computer system, all computing units (processors), which are are linked by certain networks, have similar structures with certain processing ability [6]. Efficient algorithms with satisfactory performance guarantee to allocate the computing resources is a crucial issue in such a complex system. Unfortunately, in general many traditional scheduling algorithms are not able to fulfil the requirements, due to the large amount of communication which is performed via the networks for data transmission, synchronization and other scheduling overhead among the processors allotted to the same task. Thus, many parallel task (PT) models have been proposed for this problem, among which the *malleable task* (MT) model (sometimes also called moldable task) proposed in [38] is a promising model and that has been employed in real application [2]. In the MT model, the *processing time* of a task depends on the number of processors allotted to it. The overhead of communication and synchronization is implicitly included in the processing time. The idea of MTs is to provide an alternative strategy to model the communication delays. Usually, an MT is a task including elementary operations (e.g., a numerical routine or a nested loop) which contains sufficient parallelism for being processed on multiprocessors. Thus, the sequential task can be regarded as a special case of the MT model. In most cases, the

*Research supported in part by the DFG Graduiertenkolleg 357, Effiziente Algorithmen und Mehrskalenmethoden, by the EU Project CRESCCO, Critical Resource Sharing for Cooperation in Complex Systems, IST-2001-33135, by an MITACS grant of Canada, and by the NSERC Discovery Grant DG 5-48923.

†Institut für Informatik und Praktische Mathematik, Universität zu Kiel, Germany. kj@informatik.uni-kiel.de.

‡Department of Computing and Software, McMaster University, Canada. zhanghu@mcmaster.ca

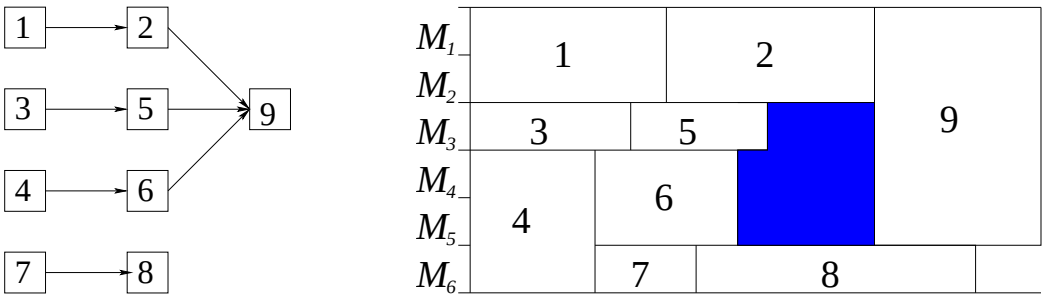

Figure 1: A precedence graph and a corresponding feasible schedule.

users have enough knowledge of the behaviour of the MTs, which leads to enough information of parallelisation. The MT model has been employed to computation for realistic applications [2, 23].

In general, there are three types of MT models based on the actual behaviour of parallel tasks. In the first model (MT1), the processing time of an MT is an arbitrary function of the number of processors allotted. In the second model (MT2), the processing time of an MT is decreasing in the number of processors allotted, while the *work function* (the product of the processing time and the processor number) is increasing in the number of processors allotted. These assumptions are based on the well-known Brent's Lemma [4], which states that the parallel execution of a task achieves some speedup if it is large enough, but does not lead to super-linear speedups. In the third model (MT3), the processing time has the same behaviour as in MT2, while the *speedup function* (the reciprocal of the processing time function) is concave in the number of processors allotted. This is based on the real behaviour of a class of massive parallel computers [33, 34, 35].

We shall study two categories of MTs corresponding to tasks without relations among them, and to applications composed of large tasks, respectively. In the first category, we are given *independent* MTs (called IMTs), such that there is no dependence the tasks. In the second category, a task can be executed by processing MTs linked by their *precedence constraints* (called PCMTs), which are determined in advance by the data flow among MTs. According to the given precedence, we can construct a *precedence graph*, which is a directed acyclic graph, where the vertices represent the set of MTs, and arcs represent the set of precedence constraints among MTs (see Figure1).

In an instance of MT scheduling, given n independent or precedent-constrained MTs (MT1, MT2 or MT3) and m homogeneous (identical) processors, we wish to find a feasible schedule with the minimum *makespan*

$C_{\max}$ (the maximum completion time). In a feasible schedule, each task is executed simultaneously on all processors allotted to it until its completion time without interruption, and each processor executes at most one task at any time. For PCMTs, the precedence constraints are further required.

Since classical parallel task scheduling problems are special cases of MT scheduling, their complexity results apply directly to MT scheduling. Thus, MT scheduling is $\mathcal{NP}$-hard [11]. IMT scheduling is proven to be strongly $\mathcal{NP}$-hard even for the case of only 5 processors, and IMT scheduling for 3 processors is shown pseudo polynomial time solvable [10]. However, the complexity of IMT scheduling for 4 processors is open. On the other hand, PCMT scheduling is shown $\mathcal{NP}$-hard in strong sense [10], and the lower bound on the approximation ratio is 4/3 [21]. Hence, people are interested in approximation algorithms with complexity polynomial in the input size (or ε^{-1}, where ε is the given accuracy).

Most existing approximation algorithms for MT scheduling are based on a two-phase approach proposed by Turek, Wolf and Yu [38]. In the first phase, an allotment problem is solved (approximately) such that each task is allotted a number of processors. Then in the second phase, the resulting non-MT scheduling is to be solved (approximately). It is clear that if we are able to apply an approximation of allotment with a ratio ρ and an approximation of non-MT scheduling with a ratio μ, then we are able to develop an r-approximation algorithm, where $r = \rho\mu$. In general, based on the above strategy, there are two following methodologies for IMT scheduling and PCMT scheduling, focusing on non-MT scheduling (the second phase) and on the allotment problem (the first phase), respectively:

- Since we consider the makespan as the optimization criterion, the second phase for IMT is essentially identical to the 2-dimensional strip packing problem [1, 5, 37, 19], which is to pack (without rotation) a set of given rectangles with width bounded by 1 into a strip with width 1 and unbounded height, and the maximum height of the rectangles is minimized. In this case, the allotment problem is usually formulated as a knapsack problem or its variants. The approximation algorithms in this category can be found in [27, 26, 29, 30, 15, 14].

- For PCMT scheduling, in general the allotment problem is solved by dynamical programming for tree precedence constraints [24, 25] or by approximation algorithms for the discrete time-cost tradeoff problem [36] as in [25, 39, 17, 18]. The non-MT scheduling in the second phase is then delivered by a variant of list scheduling algorithm [12].

We shall review four approximation algorithms for MT scheduling in this chapter. In Section 2 we shall show a 3/2-approximation algorithm for IMT scheduling (MT2) [24], and then an AFPTAS for IMT scheduling (MT1) [14] in Section 3. A $(3+\sqrt{5})/2$-approximation algorithm for PCMT scheduling with tree precedence (MT2) [25] is introduced in Section 4. Finally, we shall discuss in Section 5 a $(3+\sqrt{5})$-approximation algorithm for PCMT scheduling with general precedence (MT2) [25], and its improvement for MT2 and MT3 models [18, 17, 39].

1.1 Notations

Instances of IMT and PCMT scheduling contain a set $\mathcal{T}$ of n tasks and a set of m identical processors. Each task J_j can be executed on any integer number $l \in \{1, \ldots, m\}$ of the given processors, and the corresponding discrete positive processing time is $p_j(l)$. In addition, a PCMT instance contains a directed precedence graph $G = (V, E)$, where $V = \{1, \ldots, n\}$ represents PCMTs, and $E \subseteq V \times V$ represents precedence constraints among PCMTs. If there is an arc $(i, j) \in E$, then task J_j cannot be processed before the completion of task J_i. Task J_i is called a *predecessor* of J_j, while J_j a *successor* of J_i. We denote by $\Gamma^-(j)$ and by $\Gamma^+(j)$ the sets of the predecessors and of the successors of J_j, respectively. We also assume that $p_j(0) = \infty$ as no task J_j can be executed if there is no processor available. Furthermore, given a processing time $p_j(l)$, its *speedup* function is defined as $s_j(l) = p_j(1)/p_j(l)$, and its *work* function is $W_j(l) = lp_j(l)$. In addition, in some cases the work of a task J_j is also regarded as a function of its processing time, denoted by $w_j(p_j) = lp_j(l) = W_j(l)$.

In a schedule, each task J_j has two associated values: the starting time τ_j and the number of processors l_j allotted to task J_j. A task J_j is called *active* during the time interval from its starting time τ_j to its completion time $C_j = \tau_j + p_j(l_j)$. A schedule is *feasible* if at any time t, the number of active processors does not exceed the total number of processors, i.e., $\sum_{j:t\in[\tau_j,C_j]} l_j \leq m$, and if there is no preemption and migration. For the PCMT case, a feasible schedule further requires that the precedence constraints $\tau_i + p_i(l_i) \leq \tau_j$ are fulfilled for all $i \in \Gamma^-(j)$.

Now we consider the models of MTs as follows:

- MT1: There are no special constraints on the processing time or work.
- MT2: The following two constraints are required:
    1. The processing time $p(l)$ of a malleable task J is non-increasing

in the number l of the processors allotted to it, i.e., $p(l) \leq p(l')$, for $l \geq l'$;

2. The work of a malleable task J is non-decreasing in the number l of the processors allotted to it, i.e., $W(l) \leq W(l')$ for $l \leq l'$.

- MT3: The following two constraints are required:

1. The processing time $p(l)$ of a malleable task J is non-increasing in the number l of the processors allotted to it, i.e., $p(l) \leq p(l')$, for $l \geq l'$;
2. The speedup function of a malleable task J is concave in the number l of the processors allotted to it, i.e., $p_j(1)/p_j(l) = s_j(l) \geq [(l-l'')s_j(l') - (l-l')s_j(l'')]/(l'-l'') = p_j(1)[(l-l'')/p_j(l') - (l-l')/p_j(l'')]/(l'-l'')$, for any $0 \leq l'' \leq l \leq l' \leq m$.

2 A 3/2-Approximation Algorithm for IMT Scheduling

The original strategy in [38] for IMT scheduling in general leads to an effective mechanism to design approximation algorithms. However, its power and its limitation are that the ratios of this kind of algorithms strongly depend on the improvement of the ratio for the strip packing problem. Thus, direct application of this approach can only lead to an absolute performance ratio of 2 based on [37]. In order to avoid being limited by the best known approximation result for the strip packing problem, a $(\sqrt{3}+\varepsilon)$-approximation algorithm for IMT scheduling (MT2) was proposed in [29], and the ratio was further improved to 3/2 [30].

The approximation algorithm for IMT scheduling (MT2) applies the dual technique. An r-dual approximation algorithm for IMT scheduling (MT2) which takes a real number d as an entry and

- either delivers a schedule of length at most rd;
- or gives a correct answer that there exists no schedule of length less than d.

Furthermore, a lower bound $\underline{C}_{\max}$ can be computed such that $\underline{C}_{\max} \leq C^*_{\max} \leq 2\underline{C}_{\max}$ [27]. Therefore, a binary search approach can result is an $(r+\varepsilon)$-approximate solution provided an r-dual approximation algorithm, for any $\varepsilon > 0$.

The 3/2-dual approximation algorithm for IMT scheduling (MT2) is shown in Table 1.

- Remove the set $\mathcal{T}_S$ of small tasks whose processing times on one processor are at most $d/2$.
- Find an allotment for the remaining tasks such that any task has a processing time at most d. Partitioned the remaining tasks into two sets $\mathcal{T}_1$ and $\mathcal{T}_2$, consisting of the tasks with processing times strictly greater than $d/2$ and at most $d/2$, respectively.
- Apply basic transformations to generate a feasible 2-level schedule. The allotment may be changed according to the transformations, while no task is allotted more processors than in the initial allotment.
- Schedule the tasks in $\mathcal{T}_S$.

Table 1: The 3/2-dual approximation algorithm for IMT scheduling (MT2) in [30].

Lemma 2.1 *If a schedule of length* $3d/2$ *exists for* $\mathcal{T} \setminus \mathcal{T}_S$ *with work* $md - W_S$*, then there exists an MT schedule of length at most* $3d/2$*.*

In the second and the third steps of the algorithm in Table 1, an allotment for tasks in $\mathcal{T} \setminus \mathcal{T}_S$ is developed with respect to the following constraints:

(C1) The total work of this allotment is at most $md - W_S$.

(C2) The set $\mathcal{T}_1$ of tasks with processing times strictly greater than $d/2$, which are to be scheduled in level S_1, uses at most m processors.

(C3) The set $\mathcal{T}_2$ of tasks with processing times at most $d/2$, which are to be scheduled in level S_2, uses at most m processors.

Obviously, an allotment satisfying all above three constraints for tasks in $\mathcal{T} \setminus \mathcal{T}_S$ defines a 2-level schedule of length at most $3/2d$, which allows an 3/2-approximate solution to IMT scheduling (MT2). This allotment is obtained by the second and the third steps of the algorithm in Table 1.

In the second step, a 2-level schedule for $\mathcal{T} \setminus \mathcal{T}_S$ satisfying constraints (C1) and (C2) is established by a knapsack algorithm. Denote by $l_j(x)$

the smallest number of processing time at most x. Indeed, the following knapsack problem is to be solved: to minimize the total work of the tasks in $\mathcal{T} \setminus \mathcal{T}_S$ subject to the constraint that the tasks in $\mathcal{T}_1$ use at most m processors in total, i.e.,

$$\begin{array}{rl} \min_{\mathcal{T}_1 \subseteq \mathcal{T}} & \sum_{J_j \in \mathcal{T}_1} W_j(l_j(d)) + \sum_{J_j \notin \mathcal{T}_1} W_j(l_j(d/2)) \\ \text{s.t.} & \sum_{J_j \in \mathcal{T}_1} l_j(d) \leq m. \end{array}$$

The knapsack problem is $\mathcal{NP}$-hard [11], but allows a pseudo-polynomial time algorithm [28, 32] by dynamic programming, with a running time $O(nC)$, where C is the capacity of the knapsack problem. In our case, the capacity is m, so we are able to conduct the second step of the algorithm in time $O(mn)$. If the returned solution to the knapsack problem has a total work greater than $md - W_S$, then there exists no solution to the scheduling problem with a makespan at most d, and the dual approximation algorithm gives a correct negative answer to the entry d. Otherwise, a feasible schedule can be obtained by the transformations of the third step of the algorithm in Table 1.

Lemma 2.2 *If there exists a schedule of length at most d, then solving the knapsack formulation of the problem gives an allotment satisfying constraints (C1) and (C2).*

With the allotment attained by solving the knapsack problem, we have obtained a solution with tasks in $\mathcal{T}_1$ in level S_1 and the others $\mathcal{T}_2 = \mathcal{T} \setminus (\mathcal{T}_S \cup \mathcal{T}_1)$ in level S_2. However, the constraint (C3) may be violated. Hence, some transformations are conducted to modify the shape of the 2-level schedule and to create a new area S_0 whose processors are continuously busy in the whole interval $[0, 3d/2]$. The transformations are as follows:

(T1) If a task J_j in S_1 has a processing time at most $3d/4$ and is allotted to $l_j > 1$ processors, then allocate J_j to $l_j - 1$ processors in S_0.

(T2) If J_j and J'_j in S_1 both have processing times at most $3d/4$ and both are allotted to 1 processors, then allocate J_j and J'_j to the same processor in S_0. In the special case that J_j is the only remaining sequential task with processing time at most $3d/4$, it is put on top of a task in S_1 (if one exists) that has a processing time greater than $3d/4$, such that the completion time of J_j does not exceed $3d/2$.

(T3) Denote by q the number of idle processors in S_1. If there exists a task J_j in S_2 such that its processing time on q processors is bounded by

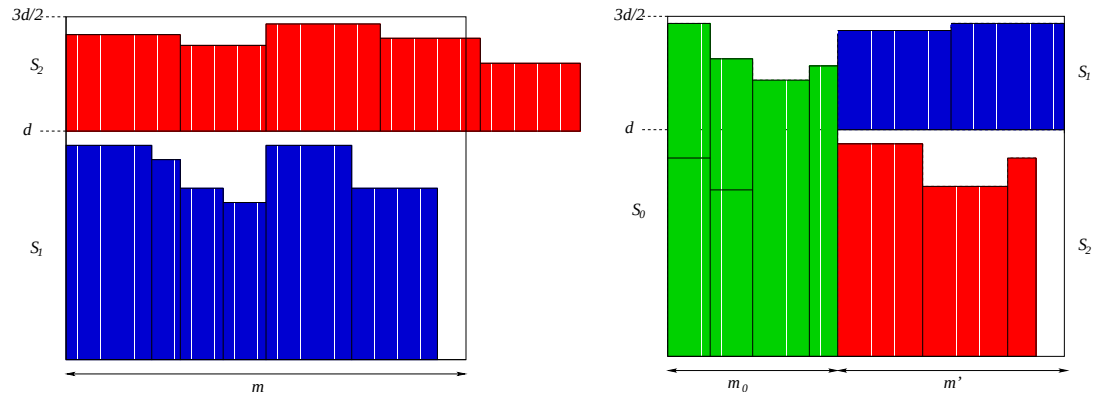

Figure 2: The two-shelf schedule from initial allotment to the final schedule by transformations.

$3d/2$, then place J_j on l_j processors, where l_j is the smallest integer in $\{1, \ldots, m\}$ such that the processing time $p_j(l_j) \leq 3d/2$. According to the resulting processing time, J_j is either scheduled in S_0 or in S_1.

The above transformations can be conducted in any order. Then the resulting schedule is feasible (see Figure 2) and the following lemma holds:

Lemma 2.3 *The algorithm that performs the transformations (T1), (T2) and (T3) for a solution to the knapsack problem delivers a feasible schedule of length at most* $3d/2$ *and total work* $md - W_S$.

Combining Lemma 2.1 and 2.3, together with the binary search strategy, a $(3/2 + \varepsilon)$-approximation algorithm is obtained:

Theorem 2.1 *There exists a* $(3/2 + \varepsilon)$*-approximation algorithm for IMT scheduling (MT2).*

3 An AFPTAS for IMT Scheduling

Given an allotment of IMTs, non-MT scheduling is identical to strip packing. Based on this idea, a 2-approximation algorithm is proposed in [26, 27] from the 2-approximation of strip packing in [37]. Since there is an AFPTAS for strip packing [19], a natural question is whether there exists an AFPTAS for IMT scheduling. Thus in this section we shall review an AFPTAS for IMT scheduling (MT1) with processing times at most 1 presented in [14].

The main ideas of the AFPTAS are as follows: First an approximate preemptive schedule with migration for IMT scheduling (MT1) is constructed by linear programming. Then the preemptive schedule is converted to an approximate non-preemptive schedule by computing a unique allotment for almost all MTs with a new rounding technique. We shall show the details of this AFPTAS in the following.

In the preemptive model, each task can be interrupted at any time before its completion at no cost and resumed later. In addition, migration is allowed for this preemptive schedule, where each task may be assigned to different processor sets during its difference execution phases [3, 10, 9]. Then the preemptive scheduling problem can be formulated as the following linear program [16]:

$$\begin{array}{lll} \min & \sum_{f\in F} x_f & \\ \text{s.t.} & \sum_l \dfrac{1}{p_j(l)} \displaystyle\sum_{f\in F:|f^{-1}(j)|=l} x_f \geq 1, & j=1,\ldots,n; \\ & x_f \geq 0, & \forall f \in F. \end{array} \tag{1}$$

The set F denotes the set of all configurations. The variable x_f indicates the length of configuration f in the schedule, and $|f^{-1}(j)|$ is the number of processors allotted to task J_j in configuration f. Clearly, the optimal objective value of (1), i.e., the length of the optimal preemptive schedule with migration, is a lower bound of the length of the optimal non-preemptive IMT schedule.

The linear program (1) can be approximately solved by binary search on the optimum value and testing in each step the feasibility of a system of (in-)equalities for a given $g \in [d_{\max}, nd_{\max}]$, where $d_{\max} = \max_{j=\{1,\ldots,n\}} d_j$ and $d_j = \min_l p_j(l)$. Notice that the length of an optimal preemptive schedule is at least $d_{\max}$ and at most $nd_{\max}$. Now the system of inequalities is given as follows:

$$\sum_l \frac{1}{p_j(l)} \sum_{f\in F:|f^{-1}(j)|=l} x_f \geq 1, \quad j=1,\ldots,n, \quad (x_f)_{f\in F} \in B,$$

where

$$B = \left\{ (x_f)_{f\in F} | \sum_{f\in F} x_f = g, x_f \geq 0, f \in F \right\}.$$

This can be performed approximately by computing an approximation solution to the following problem:

$$\lambda^* = \max\left\{\lambda | \sum_l \frac{1}{p_j(l)} \sum_{f\in F:|f^{-1}(j)|=l} x_f \geq \lambda, j = 1,\ldots,n, (x_f)_{f\in F} \in B\right\}, \tag{2}$$

which can be viewed as a fractional covering problem with convex set B and n covering constraints. Thus, the $(1-\varepsilon)$-approximation algorithm for fractional packing problems in [13] can be employed, provided a block solver that, for a price vector $y \in \mathbb{R}^n_+$, computes

$$\begin{array}{lll} \max & \sum_{j=1}^n \sum_l \frac{y_j}{p_j(l)} \cdot x_{jl} & \\ \text{s.t.} & \sum_{j=1}^n \sum_l l \cdot x_{jl} \leq m; & \\ & \sum_l x_{jl} \leq 1, & j = 1,\ldots,n; \\ & x_{jl} \in \{0,1\}, & l = 1,\ldots,m, j = 1,\ldots,n. \end{array} \tag{3}$$

This is indeed the multiple-choice knapsack problem (a generalized knapsack problem with different choices for tasks). Lawler [20] showed a $(1-\varepsilon)$-approximation algorithm for this problem. Hence, we have developed an FPTAS for the preemptive IMT scheduling (MT1) with migrations. In the second phase of the algorithm, the resulting preemptive schedule is converted to a non-preemptive schedule.

The conversion phase consists of two steps. First, a unique processor allotment for almost all tasks is generated. Afterwards a new rounding technique is applied and a constant number of tasks with non-unique processor numbers are removed. Then an instance of strip packing with one rectangle for any remaining tasks is obtained, and its approximate solution leads to an approximate solution to IMT scheduling (MT1).

Let $p_{\max} = \max_j \max_l p_j(l)$. For each task $J_j \in \mathcal{T}$ and each processor number l, the fraction is defined as $x_{j,l} = \sum_{f\in F:|f^{-1}(j)|=l} x_f/p_j(l) \geq 0$. Without loss of generality, we suppose that $\sum_l x_{j,l} = 1$. Then an instance of strip packing is constructed, with rectangles of height $x_{j,l}p_j(l)$ and width l, for $x_{j,l} > 0$. Denote by L_w the set of rectangles with width $l > \varepsilon' m$ and L_n the remaining rectangles, where ε' is defined later. Similar to the AFPTAS for strip packing in [19], for the wide rectangles in L_w, a stack packing can be introduced, such that the total height of wide rectangles $H = H(L_w)$ is the height of the stack, and the stack is divided into M groups, where M is defined later (see Figure 3). Let $W_{j,l}$ be the set of widths in group i corresponding to task J_j. The total area of narrow tasks is $A_n = \sum_{j=1}^n \sum_{l\leq\varepsilon' m} x_{j,l}p_j(l)l$, and they are placed in group 0. For each

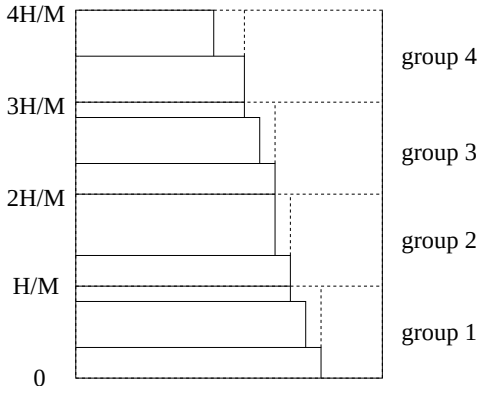

Figure 3: The stack packing for rounding.

group $i \in \{0, \ldots, M\}$ and task J_j, let $z_{j,i} = \sum_{w:w \in W_{j,i}} y_{j,i}(w)$ be the fraction of task J_j executed in group i. Denote by a_i and b_i the smallest and the largest widths of group i. To obtain a unique processor numbers for almost all tasks, the following steps need to be conducted:

---

(1) For each group i and task J_j with at least two widths in group i, compute the smallest processing time $p_j(l)$ among all processor numbers $l \in [a_i, b_i]$. Let $l_{j,i}$ be such a processor number. Now place the rectangles corresponding to task J_j in group i by $(z_{j,i}p_j(l_{j,i}), l_{j,i})$.

(2) For each task J_j with at least two widths in group 0: compute the smallest area $p_j(l)l$ among all small processor numbers $l \leq \varepsilon' m$. Let $l_{j,0}$ be such a processor number. Then replace all rectangles corresponding to task J_j in group 0 by $(z_{j,0}p_j(l_{j,0}), l_{j,0})$.

(3) Round all tasks over the groups using a general assignment problem:

$$\begin{array}{rcll} \sum_{j=1}^{n} z_{j,0} p_j(l_{j,0}) l_{j,0} & \leq & A_n; & \\ \sum_{j=1}^{n} z_{j,i} p_j(l_{j,i}) & \leq & H/M, & i = 1, \ldots, M; \\ \sum_{i=0}^{M} z_{j,i} & = & 1, & j = 1, \ldots, n; \\ z_{j,i} & \geq & 0, & j = 1, \ldots, n, i = 1, \ldots, M. \end{array} \tag{4}$$

This formulation is closely related to scheduling tasks on unrelated machines [22]. Thus, one can now round the variables $z_{j,i}$ such that there are at

most M fractional variables [22]. For tasks with integer variables, unique processor numbers are allotted. The remaining tasks are executed at the end of the schedule with processing time at most M. We now obtain a rectangle packing instance with a set $L'_w = \{(p_j(l_{j,i}), l_{j,i}) | z_{j,i} = 1, i > 0\}$ of wide rectangles and a set $L'_n = \{(p_j(l_{j,0}), l_{j,0}) | z_{j,0} = 1\}$ for narrow rectangles.

---

(0) set $\delta \leq \min\{1, \varepsilon/8\}$, $\varepsilon' = \delta/(\delta+2)$, $M = 1/\varepsilon'^2$;

(1) compute the values $x_{j,l} = \sum_{f \in F : |f^{-1}(j)| = l} x_f / p_j(l) \in [0,1]$ for each task $T_j \in T$ and l;

(2) construct an instance of strip packing with rectangles $(x_{j,l} p_j(l), l)$ for $x_{j,l} > 0$, and let $L_w = \{(x_{j,l} p_j(l), l) | l > \varepsilon' m\}$ and $L_n = \{(x_{j,l} p_j(l), l) | l \leq \varepsilon' m\}$.

(3) apply the rounding technique to obtain a strip packing with instance $L'_w = \{(p_j(l_{j,i}), l_{j,i}) | z_{j,i} = 1, i > 0\}$ and $L'_n = \{(p_j(l_{j,0}), l_{j,0}) | z_{j,0} = 1\}$ and set $F = \{T_j | z_{j,i} \in (0,1)$ for at least one $i \in \{0, \ldots, M\}\}$;

(4) construct instance $sup(L'_w)$ (via strip packing in [19]) with a constant number M of distinct widths;

(5) solve fractional strip packing for $sup(L'_w)$ approximately with ratio $(1+\delta)$ by the algorithm in [13] and round the solution to obtain only M non-zero variables x_j;

(6) place the wide rectangles of L'_w into the space generated by the non-zero variables x_j;

(7) insert the narrow rectangles of L'_n using modified next fit decreasing height [5, 19];

(8) schedule the tasks in F at the end of the schedule.

Table 2: The algorithm to convert a preemptive schedule to a non-preemptive schedule in [14].

The algorithm to convert the preemptive schedule to non-preemptive schedule is shown in Table 2. Hence, for any $\varepsilon > 0$, the AFPTAS for IMT scheduling (MT1) works as follows: First we set $\delta = \min\{1, \varepsilon/8\}$. Then we

compute a $(1+\delta)$-approximate preemptive schedule, and finally we convert the preemptive schedule into a non-preemptive schedule.

Theorem 3.1 *If the maximum processing time is at most* 1*, then there exists an AFPTAS for IMT scheduling (MT1) .*

In [14], to improve the running time, a preprocessing step is added, and speedup techniques in binary search and in solving linear program are studied. Finally, a simplified AFPTAS for IMT scheduling (MT2) is also presented.

4 An Approximation Algorithm for PCMT Scheduling with Tree Precedence

We study a special case of PCMT scheduling where the precedence graph is a tree, a series parallel graph [31] or a bounded width graph. In [24] the resulting schedule is carefully analyzed such that the bound of the increase of the schedule length in the second phase (non-MT schedule) is estimated and a 4-approximation algorithm for PCMT scheduling with tree precedence is obtained. The performance ratio is further improved to $(3+\sqrt{5})/2 \approx 2.61803$ in [25].

As usual, the approximation algorithm in [25] for PCMT scheduling with tree precedence has two phases. In the first phase the allotment problem is solved such that each task is allotted a certain number of processors. In the second phase, for the given allotment (maybe with slight modification from the solution to the allotment problem), a non-MT schedule is obtained by classical scheduling algorithms.

The allotment problem in this case is to find for each task a number of processors, to minimize the following cost:

$$\text{cost} = \max\{L, W/m\}, \tag{5}$$

where L is the length of a *critical path* (the longest path in the precedence graph under the allotment), and W is the sum of the works of all tasks.

We first study the second phase, i.e., given an allotment, to schedule PCMTs according to their precedence. In fact in [25, 17, 18], a variant of the list scheduling algorithm [12] is performed. Denoting by l'_j the number of processors allotted to a task J_j in the allotment α' generated in the first phase (i.e., a solution to the allotment problem), the algorithm constructs

LIST (J, m, α', μ)

- generate new allotment α: $l_j = \min\{l'_j, \mu\}$ for $j \in \{1, \ldots, n\}$;
- $SCHEDULED = \emptyset$;
- **if** $SCHEDULED \neq \mathcal{T}$ **then**
  - $READY = \{J_j | \Gamma^-(j) \subseteq SCHEDULED\}$;
  - compute the earliest possible starting time under α for all tasks in $READY$;
  - schedule the task $J_j \in READY$ with the smallest earliest starting time;
  - $SCHEDULED = SCHEDULED \cup \{J_j\}$;
- **end**

Table 3: Algorithm **LIST**

a new allotment α with a parameter $\mu \in \{1, \ldots, \lfloor (m+1)/2 \rfloor\}$ defined later, and then schedules all tasks. The algorithm is shown in Table 3.

Denote by α^* the optimal (fractional) solution to the allotment problem, and by L^* and W^* its critical path length and total work. In addition, let L and W denote the critical path length and total work of the final schedule corresponding to allotment α.

The following bound holds by using (5) for α^*

$$\max\{L^*, W^*/m\} \leq C^*_{\max}, \tag{6}$$

where $C^*_{\max}$ is the length of α^*. Clearly, it is also a lower bound of an optimal PCMT schedule. Suppose that we have already a λ-approximate solution of the allotment problem, i.e.,

$$\max\{L', W'/m\} \leq \lambda \max\{L^*, W^*/m\}. \tag{7}$$

Our goal is to find the link between the makespan of the final schedule and the allotment in the first phase. Since in the second phase, each task J_j is allotted $l_j \leq l'_j$ processors, according to the property of MT2, the work does not increase. Therefore

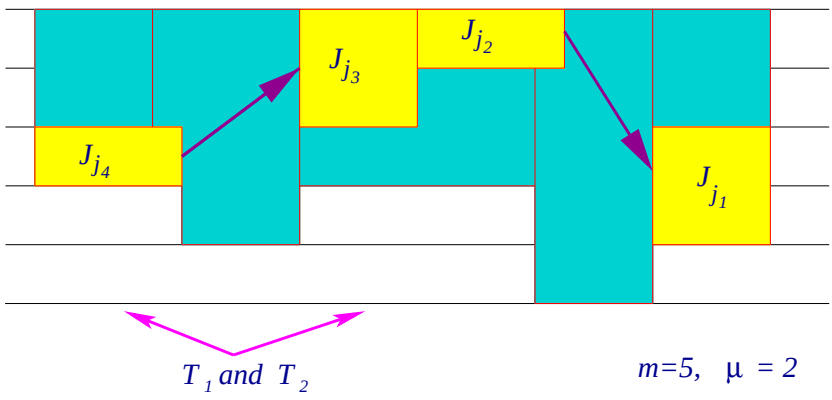

Figure 4: An example of the "heavy" path.

$$W \leq W' \leq m\lambda C^*_{\max}. \tag{8}$$

In the final schedule, the time interval $[0, C_{\max}]$ consists of three types of time slots. In the first type of time slots, at most $\mu - 1$ processors are busy. In the second type of time slots, at least μ and at most $m - \mu$ processors are busy. In the third type at least $m - \mu + 1$ processors are busy. Denote by T_1, T_2 and T_3 the sets of the three types time slots, and by $|T_i|$ the overall lengths for $i \in \{1, 2, 3\}$. In the case that $\mu = (m+1)/2$ and m odd, $T_2 = \emptyset$. In other cases all three types of time slots may exist. Clearly,

$$C_{\max} = |T_1| + |T_2| + |T_3|. \tag{9}$$

Since during the first (respectively, the second and the third) type of time slots, at least one (respectively, μ and $m - \mu + 1$) processors are busy, we have

$$W \geq |T_1| + \mu|T_2| + (m - \mu + 1)|T_3|. \tag{10}$$

Lemma 4.1 $|T_1| + \mu|T_2|/m \leq L'$.

Proof: We construct a "heavy" directed path $\mathcal{P}$ in the final schedule. The last task in the path $\mathcal{P}$ is any multiprocessor task J_{j_1} that completes at time $C_{\max}$ (the makespan of the final schedule). After we have defined the last $i \geq 1$ tasks $J_{j_i} \rightarrow J_{j_{i-1}} \rightarrow \cdots \rightarrow J_{j_2} \rightarrow J_{j_1}$ on the path $\mathcal{P}$, we can determine the next task $J_{j_{i+1}}$ as follows: Consider the latest time slot t in $T_1 \cup T_2$ that

is before the starting time of task J_{j_i} in the final schedule. Let V' be the set of task J_{j_i} and its predecessor tasks that start after time t in the schedule. Since during time slot t at most $m - \mu$ processors are busy, and since at most μ processors are allotted to any task in V', none of the tasks in V' can be ready for execution during the time slot t. Therefore for every task in V' some predecessor is being executed during the time slot t. Then we select any predecessor of task J_{j_i} that is running during slot t as the next task $J_{j_{i+1}}$ on the path $\mathcal{P}$. This search procedure stops when $\mathcal{P}$ contains a task that starts before any time slot in $T_1 \cup T_2$. An example of the "heavy" path is illustrated in Figure 4. Now we examine the stretch of processing time for all jobs in $\mathcal{P}$ in the rounding procedure of the first phase and in the new allotment α of the second phase.

Consider a task J_j in the resulting path $\mathcal{P}$. If in the final schedule $l_j < \mu$, then in the first phase $l'_j = l_j < \mu$. If $l_j = \mu$, then $\mu \leq l'_j \leq m$. Since $l_j p_j(l_j) \leq l'_j p_j(l'_j)$ by the property of MT2, $p_j(l'_j)/p_j(l_j) \geq \mu/l'_j \geq \mu/m$. With the construction of the directed path $\mathcal{P}$, it covers all time slots in $T_1 \cup T_2$ in the final schedule. In addition, denote by $L'(\mathcal{P})$ the length of path $\mathcal{P}$ in allotment α'. The tasks during time slots in T_1 contribute a total length of at least $|T_1|$ to $L'(\mathcal{P})$, and tasks in T_2 contribute at least $\mu|T_2|/m$ to $L'(\mathcal{P})$. Because $L'(\mathcal{P}) \leq L'$, the lemma follows. □

Combining the above bounds, the following theorem holds:

Theorem 4.1 *If there exists a λ-approximation algorithm for the allotment problem, then there exists a λr-approximation algorithm for PCMT scheduling (MT1), where $r = \min_{1 \leq \mu \leq \lfloor (m+1)/2 \rfloor} \max\{m/\mu, (2m-\mu)/(m-\mu+1)\}$.*

Proof: Multiplying (9) by $m - \mu + 1$ and subtracting (10) from it yields

$$(m - \mu + 1)C_{\max} \leq W + (m - \mu)|T_1| + (m - 2\mu + 1)|T_2|. \qquad (11)$$

We consider two cases. In the first case, $m/\mu \leq (2m-\mu)/(m-\mu+1)$. Thus, $r = (2m-\mu)/(m-\mu+1)$ and $(m-2\mu+1) \leq \mu(m-\mu)/m$. Substituting this into (11), using Lemma 4.1, (8), (7) and (6) we have $(m-\mu+1)C_{\max} \leq (2m-\mu)\lambda C^*_{\max}$, and the resulting schedule is indeed a λr-approximate solution. The analysis of the other case is analogous. □

Furthermore, it can be proven that for all $m \geq 2$, μ equals either the integer above or the integer below $(3m - \sqrt{5m^2 - 4m})/2$, and $r < (3 + \sqrt{5})/2 \approx 2.61803$. As m tends to infinity, μ tends to $(3 - \sqrt{5})/2 \approx 0.38196$, and r tends to $(3 + \sqrt{5})/2$.

We have now established the relation between the approximation ratio for the allotment problem and PCMT scheduling (MT2). When the prece-

dence graph is a tree, a series parallel graph, or a bounded width graph, the allotment problem can be solved approximately with ratio $1+\varepsilon$ (for any $\varepsilon > 0$) by dynamic programming [24, 25]. In this way, we have already obtained a 2.61803-approximation algorithm for PCMT with tree precedence (MT2).

5 Approximation Algorithm for PCMT Scheduling

In the case of arbitrary precedence graphs, the allotment problem is related to the *discrete time-cost tradeoff* problem [7, 36], which is $\mathcal{NP}$-complete [8]. In fact, the approximation algorithm in [36] for the discrete time-cost tradeoff problem is employed to solve the allotment problem in [25] for a 5.23606-approximation algorithm for PCMT Scheduling (MT2).

An instance of the discrete time-cost tradeoff problem is a *project* given by a finite set J of *activities* with a *partial order* $(J, \prec)$ on the set of activities. All activities have to be executed in accordance with the precedence constraints given by the partial order. Each activity $J_j \in J$ has a set of feasible *durations* $\{d_{j_1}, \ldots, d_{j_{k(j)}}\}$ sorted in a non-decreasing order, and has a non-increasing non-negative *cost* function $c_j : \mathbb{R}_+ \to \mathbb{R}_+ \cup \{\infty\}$, where $c_j(x_j)$ is the amount paid to run J_j with duration x_j.

Skutella [36] presented approximation algorithms for above problems, in particular, an algorithm for the bicriteria problem such that $c(x) < B/(1-\rho)$ and $t(x) \leq L/\rho$ for a fixed $\rho \in (0, 1)$. The budget problem can be described as the following integer linear program:

$$\begin{array}{lll} \min & L & \\ \text{s.t.} & 0 \leq C_j \leq L, & \text{for all } j; \\ & C_i + x_j \leq C_j, & \text{for all } i \in \Gamma^-(j) \text{ and all } j; \\ & c(x) = \sum_{j=1}^{n} c_j(x_j) \leq B; & \\ & x_j \in \{d_{j_1}, \ldots, d_{j_{k(j)}}\}, & \text{for all } j. \end{array} \tag{12}$$

Here C_j is the completion time of activity J_j, $c_j(x_j)$ the cost of the duration x_j and d_{j_p} the p-th feasible duration of activity J_j. The first set of constraints shows that completion times of any tasks are bounded by the project length. The second set is related to the precedence constraints. In addition, the third set of constraint means that the total cost should be bounded by the budget B.

To solve it, first a "reduced" cost function $\hat{c}$ is set such that for any duration d_{j_l}, $\hat{c}_j(d_{j_l}) = c_j(d_{j_l}) - c_j(d_{j_{k(j)}})$. Since $d_{j_{k(j)}}$ is the maximum duration for activity J_j, $c_j(d_{j_{k(j)}})$ is the minimum over all durations for activity J_j and the "reduced" cost $\hat{c}$ is also positive. The amount of $P = \sum_{j=1}^{n} c_j(d_{j_{k(j)}})$ is called the "fixed" cost. In the second step, the "reduced" instance is transformed to a two-duration instance such that each given "virtual" activity has only at most two feasible durations. For any activity J_j, we introduce the first "virtual" activity J_{j_1} as follows: J_{j_1} has only two fixed feasible durations $s_j(1) = t_j(1) = d_{j_1}$, and the corresponding "virtual" costs are $\bar{c}_j(s_j(1)) = \bar{c}_j(t_j(1)) = 0$. Then for each $1 < i \leq k(j)$ a "virtual" activity J_{j_i} is introduced. Each "virtual" activity J_{j_i} has a duration x_{j_i} chosen from only two feasible "virtual" durations $s_j(i) = 0$ and $t_j(i) = d_{j_i}$, and the corresponding "virtual" costs are $\bar{c}_{j_i}(s_j(i)) = \hat{c}_j(d_{j_{i-1}}) - \hat{c}_j(d_{j_i})$ and $\bar{c}_{j_i}(t_j(i)) = 0$. It is easy to verify that for each activity J_j, the sum of corresponding "virtual" costs equals to the "reduced" cost. Thus the activity J_j can be modelled as $k(j)$ parallel "virtual" activities, which are represented by parallel edges in the edge diagram. Then there is a canonical mapping of feasible durations x_j for activity J_j to tuples of feasible durations $x_{j_1}, \ldots, x_{j_{k(j)}}$ for "virtual" activities $J_{j_1}, \ldots, J_{j_{k(j)}}$ such that the duration of the activity J_j is the maximum over all durations of the corresponding "virtual" activities and the cost of J_j is the sum of the costs of the "virtual" activities, i.e., $x_j = \max\{x_{j_1}, \ldots, x_{j_{k(j)}}\}$ and $\hat{c}_j(x_j) = \sum_{i=1}^{k(j)} \bar{c}_{j_i}(x_{j_i})$. Moreover, this mapping is bijective if we restrict ourselves without loss of generality to tuples of durations $x_{j_1}, \ldots, x_{j_{k(j)}}$ satisfying $x_{j_i} = t_j(i)$ if $t_j(i) \leq \max\{x_{j_1}, \ldots, x_{j_{k(j)}}\}$. In this way we have obtained a two-duration instance such that each activity has at most two feasible durations, and the solution of this instance can be transformed to a solution of the "reduced" instance.

Finally, we consider the linear relaxation of the two-duration instance, where for each "virtual"activity J_{j_i}, the "virtual" cost function is linear and non-increasing within the interval $[s_j(i), t_j(i)]$ as follows:

$$\bar{c}_{j_i}(y) = \begin{cases} \infty, & \text{if } y < s_j(i); \\ \dfrac{t_j(i) - y}{t_j(i) - s_j(i)} \bar{c}_{j_i}(s_j(i)) + \dfrac{y - s_j(i)}{t_j(i) - s_j(i)} \bar{c}_{j_i}(t_j(i)), & \text{if } s_j(i) \leq y \leq t_j(i); \\ \bar{c}_{j_i}(t_j(i)) = 0, & \text{if } y \geq t_j(i). \end{cases} \tag{13}$$

Therefore we are able to find a fractional solution of the linear relaxation of the two-duration instance. To obtain a feasible (integer) solution of (12), we need to take some rounding technique. For a given $\rho \in (0, 1)$, if for a

"virtual" activity J_{j_i} the fractional solution $x_{j_i} \in [0, \rho d_{j_{k(j)}})$, we round it to $\bar{x}_{j_i} = 0$. Otherwise if $x_{j_i} \in [\rho d_{j_{k(j)}}, d_{j_{k(j)}}]$ we round it to $\bar{x}_{j_i} = d_{j_{k(j)}}$, where $\bar{x}$ is the rounded solution. In this way, Skutella shows that $\hat{c}(x) \leq (B - P)/(1 - \rho)$ and $C_{\max} \leq L/\rho$, where $B - P$ is the optimal cost for the linear relaxation of (12) with a deadline L.

In [25], the above algorithm is applied together with binary search to solve the allotment problem approximately. The rounding parameter $\rho = 1/2$ is set to obtain a 2-approximation algorithm for the allotment problem. According to Theorem 4.1, the following theorem holds:

Theorem 5.1 *There exists a* 5.23606*-approximation algorithm for PCMT scheduling (MT2).*

5.1 Improved Approximation Algorithm for PCMT Scheduling

Jansen and Zhang [18] noticed the following two facts: First, solving the discrete time-cost tradeoff problem for the two optimization criteria with the same ratio in the first phase in [25] does not necessarily lead to the best possible ratio for the whole algorithm; Second, the "fixed" cost in solving the discrete time-cost tradeoff problem does not change during the rounding procedure. Thus, they further improved the approximation ratio for PCMT scheduling (MT2) to 4.730598.

In their algorithm, the rounding parameter ρ is not fixed to be $1/2$, but is used as a variable parameter in the second phase. In this way, a min-max nonlinear program is obtained, whose objective value is the approximation ratio. In this way the ratio can only be improved to 5.162 [39]. In addition, the "fixed" cost is carefully separated, and a new linear relaxation for the allotted problem is developed to avoid the binary search.

For any malleable task J_j of MT2, we also denote by $w(\cdot)$ the work function in processing time, i.e., $w_j(p_j(l)) = W_j(l)$. Denote by x_j the fractional duration of the allotment problem (or the processing time). The new linear relaxation of the allotment problem is:

$$
\begin{array}{lll}
\min & C = \max\{L, W/m\} & \\
\text{s.t.} & 0 \le C_j \le L, & \text{for all } j; \\
& C_j + x_k \le C_k, & \text{for all } j \text{ and } k \in \Gamma^+(j); \\
& x_j \le p_j(1), & \text{for all } j; \\
& x_{j_i} \le x_j, & \text{for all } j \text{ and } i = 1, \ldots, m; \\
& 0 \le x_{j_i} \le p_j(i), & \text{for all } j \text{ and } i = 2, \ldots, m; \\
& x_{j_1} = p_j(m), & \text{for all } j; \\
& \hat{w}_j(x_j) = \sum_{i=1}^{m} \bar{w}_{j_i}(x_{j_i}), & \text{for all } j; \\
& P = \sum_{j=1}^{n} p_j(1); & \\
& \sum_{j=1}^{n} \hat{w}_j(x_j) + P \le W, &
\end{array}
\tag{14}
$$

where the "virtual" work function $\bar{w}_j(x_{j_m}) = 0$, and for all j and $i = 1, \ldots, m-1$:

$$
\bar{w}_{j_i}(x_{j_i}) = [W_j(i+1) - W_j(i)](p_j(i) - x_{j_i})/p_j(i). \tag{15}
$$

The rounding technique in [36] is applied here, and one can show

$$
L' \le L^*/\rho \quad \text{and} \quad (W' - P) \le (W^* - P)/(1-\rho). \tag{16}
$$

Furthermore, the bounds (6), (8) and (9) still holds. We can prove the following bound similar to (4.1):

Lemma 5.1 $\rho|T_1| + \min\{\mu/m, \rho\}|T_2| \le C^*_{\max}$.

Denote by $x_i = |T_i|/C^*_{\max}$ the normalized lengths of the i-th type of time slots, it can be shown that the approximation ratio is the objective value of the following min-max nonlinear program:

$$
\begin{array}{lll}
\min_{\mu,\rho} \max_{x_1,x_2} & \dfrac{x_1(m-\mu)(1-\rho) + x_2(1-\rho)(m-2\mu+1) + m}{(m-\mu)(1-\rho)+1} & \\
\text{s.t.} & \rho x_1 + \min\{\rho, \mu/m\} x_2 \le 1; & \\
& x_1 + x_2(\mu(1-\rho)+\rho) \le m; & \\
& x_1, x_2 \ge 0; & \\
& \rho \in (0,1); & \\
& \mu \in \{1, \ldots, \lfloor (m+1)/2 \rfloor\}. &
\end{array}
\tag{17}
$$

A complicated analysis is needed to solve (17). It can be shown that to obtain the optimal value, analytic roots of a polynomial of degree 6 with parameter-coefficients are required, which is in general impossible. Thus in [18] the parameters are set as $\rho = 0.43$ and $\mu = (93m - \sqrt{4349m^2 - 4300m})/100$. Hence they show

Theorem 5.2 *There exists an algorithm for PCMT scheduling (MT2) with an approximation ratio*

$$\begin{aligned} r &\leq \frac{100}{43} + \frac{100}{43}\frac{(43m-100)(57\sqrt{4349m^2-4300m}-399m-4300)}{139707m^2-174021m-184900} \\ &\leq \frac{100}{43} + \frac{100(\sqrt{4349}-7)}{2451} \approx 4.730598. \end{aligned}$$

It is also proved that when m tends to infinity, the optimal asymptotic choices are $r = 0.430991$ and $m \to 0.270875m$. In this case the ratio $r \to 4.730577$.

5.2 Approximation Algorithm for PCMT Scheduling (MT3)

Jansen and Zhang [17] also studied the model MT3, which is a generalization of the continuous model by [33, 34, 35]. The ideas are similar to those for MT2 in [18].

First, one can show that MT3 is a special case of MT2 where the work function is convex in the processing times. Then a piecewise linear program with convex constraints is developed for the relaxed allotment problem, which is polynomial time solvable. Finally the variant of the list scheduling algorithm is applied to obtain a feasible schedule.

The relaxed allotment problem can be formulated as

$$\begin{array}{lll} \min & C = \max\{L, W/m\} & \\ \text{s.t.} & C_i + x_j \leq C_j, & \text{for all } j \text{ and } i \in \Gamma^-(j); \\ & 0 \leq C_j \leq L, & \text{for all } j; \\ & W/m = \sum_{j=1}^{n} w_j(x_j)/m; & \\ & x_j \in [p_j(m), p_j(1)], & \text{for all } j, \end{array} \tag{18}$$

where the continuous work function is defined as follows: If $x = p_j(l)$ for $l = 1, \ldots, m$, then $w_j(x_j) = w_j(p_j(l))$. If $x \in (p_j(l+1), p_j(l))$ for $l = 1, \ldots, m-1$, then

$$w_j(x_j) = \frac{w_j(p_j(l+1)) - w_j(p_j(l))}{p_j(l+1) - p_j(l)} x_j + \frac{w_j(p_j(l))p_j(l+1) - w_j(p_j(l+1))p_j(l)}{p_j(l+1) - p_j(l)} \tag{19}$$

In the interval $[p_j(l+1), p_j(l)]$, we define a critical point l_c such that $l_c = l + 1 - \rho$ for the rounding parameter $\rho \in [0, 1]$. The processing time $p_j(l_c)$ is given by $p_j(l_c) = p_j(l+1-\rho) = \rho p_j(l) + (1-\rho)p_j(l+1)$, and its work is $w_j(p_j(l_c)) = (1-\rho)w_j(p_j(l+1)) + \rho w_j(p_j(l)) = (1-\rho)(l+1)p_j(l+1) + \rho l p_j(l)$.

We apply the following rounding technique for the fractional solution to (18): If $x_j^* \geq p_j(l_c)$ it is rounded up to $p_j(l)$, and otherwise rounded down to $p_j(l+1)$.

We have the following new bounds:

Lemma 5.2 *For any job J_j, in the allotment α' its processing time $p_j(l'_j) \leq 2x_j^*/(1+\rho)$, and the its work $w_j(p_j(l'_j)) = l'_j p_j(l'_j) \leq 2l_j^* x_j^*/(2-\rho) = 2w_j(x_j^*)/(2-\rho)$, where x_j^* is the optimal solution to (18).*

Lemma 5.3 $(1+\rho)|T_1|/2 + \min\{\mu/m, (1+\rho)/2\}|T_2| \leq C_{\max}^*$.

Then the the approximation ratio is the objective value of the following min-max nonlinear program:

$$\begin{array}{rl} \min_{\mu,\rho} \max_{x_1,x_2} & \dfrac{2m/(2-\rho) + (m-\mu)x_1 + (m-2\mu+1)x_2}{m-\mu+1} \\ \text{s.t.} & (1+\rho)x_1/2 + \min\{\mu/m, (1+\rho)/2\}x_2 \leq 1; \\ & x_1, x_2 \geq 0; \\ & \rho \in [0,1]; \\ & \mu \in \{1, \ldots, \lfloor (m+1)/2 \rfloor\}. \end{array} \tag{20}$$

Again, the optimal solution can not be obtained analytically in general. Hence the parameter settings are $r = 0.26$ and $m = (113m - \sqrt{6469m^2 - 6300m})/100$.

Theorem 5.3 *There exists an algorithm for PCMT scheduling (MT3) with an approximation ratio*

$$\begin{aligned} r &\leq \frac{100}{63} + \frac{100}{345303} \frac{(63m-87)(\sqrt{6469m^2-6300m}+13m)}{m^2-m} \\ &\leq \frac{100}{63} + \frac{100(\sqrt{6469}+13)}{5481} \approx 3.291919. \end{aligned}$$

It is also proved that when m tends to infinity, the optimal asymptotic choices are $r = 0.261917$ and $m \to 0.325907m$. In this case the ratio $r \to 3.291913$.

Acknowledgement

The authors thank Ulrich Michael Schwarz for helpful comments.

References

[1] R. Baker, E. G. Coffman and R. L. Rivest, Orthogonal packing in two dimensions, *SIAM Journal on Computing*, 9(4) (1980), 846–855.

[2] E. Blayo, L. Debrue, G. Mounié and D. Trystram, Dynamic load balancing for ocean circulation with adaptive meshing, *Proceedings of the 5th European Conference on Parallel Computing*, Euro-Par 1999, LNCS 1685, 303–312.

[3] J. Blazewicz, M. Drabowski and J. Weglarz, Scheduling multiprocessor tasks to minimize schedule length, *IEEE Transactions on Computers*, C-35(5) (1986), 389–393.

[4] R. Brent, *The Parallel Evaluation of Arithmetic Expressions in Logarithmic Time*, Academic Press, New York, 1973.

[5] E. G. Coffman, M. R. Garey, D. S. Johnson and R. E. Tarjam, Performance bounds for level-oriented two-dimensional packing algorithms, *SIAM Journal on Computing*, 9(4) (1980), 808–826.

[6] D. E. Culler, J. P. Singh and A. Gupta, *Parallel computer architecture: A hardware/software approach*, Morgan Kaufmann Publishers, San Francisco, 1999.

[7] P. De, E. J. Dunne, J. B. Gosh and C. E. Wells, The discrete time-cost tradeoff problem revisited, *European Journal of Operational Research*, 81 (1995), 225–238.

[8] P. De, E. J. Dunne, J. B. Gosh and C. E. Wells, Complexity of the discrete time-cost tradeoff problem for project networks, *Operations Research*, 45 (1997), 302–306.

[9] M. Drozdowski, Scheduling multiprocessor tasks – an overview, *European Journal on Operations Research*, 94 (1996), 215–230.

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

[11] M. R. Garey and D. S. Johnson, *Computer and intractability: a guide to the theory of $\mathcal{NP}$-completeness*, W. H. Freeman, New York, 1979.

[12] R. L. Graham, Bounds for certain multiprocessing anomalies, *Bell System Technical Journal*, 45 (1966), 1563–1581.

[13] M. D. Grigoriadis, L. G. Khachiyan, L. Porkolab and J. Villavicencio, Approximate max-min resource sharing for structured concave optimization, *SIAM Journal on Optimization*, 11 (2001), 1081–1091.

[14] K. Jansen, Scheduling malleable parallel tasks: an asymptotic fully polynomial-time approximation scheme, *Algorithmica*, 39 (2004), 59–81.

[15] K. Jansen and L. Porkolab, Linear-time approximation schemes for scheduling malleable parallel tasks, *Algorithmica*, 32 (2002), 507–520.

[16] K. Jansen and L. Porkolab, Computing optimal preemptive schedules for parallel tasks: linear programming approaches, *Mathematical Programming*, 95 (2003), 617–630.

[17] K. Jansen and H. Zhang, Scheduling malleable tasks with precedence constraints, *Proceedings of the 17th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA 2005, 86–95.

[18] K. Jansen and H. Zhang, An approximation algorithm for scheduling malleable tasks under general precedence constraints", *to appear in Proceedings of the 16th Annual International Symposium on Algorithms and Computation*, ISAAC 2005, LNCS.

[19] C. Kenyon and E. Rémila, A near-optimal solution to a two-dimensional cutting stock problem, *Mathematics of Operations Research*, 25(4) (2000), 645–656.

[20] E. Lawler, Fast approximation algorithms for knapsack problems, *Mathematics of Operations Research*, 4 (1979), 339–356.

[21] J. K. Lenstra and A. H. G. Rinnooy Kan, Complexity of scheduling under precedence constraints, *Operations Research*, 26 (1978), 22–35.

[22] J. K. Lenstra, D. B. Shmoys and E. Tardos, Approximation algorithms for scheduling unrelated parallel machines, *Mathematical Programming*, 24 (1990), 259–272.

[23] R. Lepère, G. Mounié, B. Robič and D. Trystram, Malleable tasks: an electromagnetic efficient model for solving actual parallel applications, *Proceedings of the 1999 International Conference on Parallel Computing*, Parco 1999, Imperial College Press, 598–605.

[24] R. Lepère, G. Mounié and D. Trystram, An approximation algorithm for scheduling trees of malleable tasks, *European Journal of Operational Research*, 142(2) (2002), 242–249.

[25] R. Lepère, D. Trystram and G. J. Woeginger, Approximation algorithms for scheduling malleable tasks under precedence constraints, *International Journal of Foundations of Computer Science*, 13(4) (2002), 613–627.

[26] W. Ludwig, Algorithms for scheduling malleable and nonmalleable parallel tasks, *Ph.D. Thesis*, University of Wisconsin – Madison, 1995.

[27] W. Ludwig and P. Tiwari, Scheduling malleable and nonmalleable parallel tasks, *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, SODA 1994, 167–176.

[28] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*, Wiley, New York, 1990.

[29] G. Mounié, C. Rapine and D. Trystram, Efficient approximation algorithms for scheduling malleable tasks, *Proceedings of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA 1999, 23–32.

[30] G. Mounié, C. Rapine and D. Trystram, A 3/2-dual approximation algorithm for scheduling independent monotonic malleable tasks, *manuscript*.

[31] R. H. Möhring, Computationally tractable classes of ordered sets, in I. Rival (Eds.) *Algorithms and Order*, Kluwer Academic Publishers, 105–193.

[32] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.

[33] G. N. S. Prasanna and B. R. Musicus, Generalised multiprocessor scheduling using optimal control, *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA 1991, 216–228.

[34] G. N. S. Prasanna and B. R. Musicus, Generalized multiprocessor scheduling for directed acyclic graphs, *Proceedings of Supercomputing 1994*, 237–246.

[35] G. N. S. Prasanna and B. R. Musicus, The optimal control approach to generalized multiprocessor scheduling, *Algorithmica*, 15(1), (1996), 17–49.

[36] M. Skutella, Approximation algorithms for the discrete time-cost tradeoff problem, *Mathematics of Operations Research*, 23 (1998), 909–929.

[37] A. Steinberg, A strip-packing algorithm with absolute performance bound 2, *SIAM Journal on Computing*, 26(2) (1997), 401–409.

[38] J. Turel, J. Wolf and P. Yu, Approximate algorithms for scheduling parallelizable tasks, *Proceedings of the 4th Annual Symposium on Parallel Algorithms and Architectures*, SPAA 1992, 323–332.

[39] H. Zhang, Approximation Algorithms for Min-Max Resource Sharing and Malleable Tasks Scheduling, *Ph.D. Thesis*, University of Kiel, Germany, 2004.