On Minimizing Average Weighted Completion Time: A PTAS for Scheduling General Multiprocessor Tasks *

Aleksei V. Fishkin, Klaus Jansen[1], and Lorant Porkolab[2]

[1] Institut für Informatik und Praktische Mathematik,
Christian-Albrechts-Universität zu Kiel, Olshausenstrasse 40, 24 098 Kiel, Germany,
{avf,kj}@informatik.uni-kiel.de
[2] Applied Decision Analysis, PricewaterhouseCoopers, 1 Embankment Place, London
WC2N 6RH, United Kingdom, lorant.porkolab@uk.pwcglobal.com

Abstract. We study the problem of scheduling n independent general multiprocessor tasks on a fixed number of processors, where the objective is to compute a non-preemptive schedule minimizing the average weighted completion time. For each task, its execution time is given as a function of the subset of processors assigned to the task. We propose here a polynomial-time approximation scheme for the problem that computes a $(1+\epsilon)$-approximate solution in $O(n \log n)$ time for any fixed $\epsilon > 0$ accuracy. This provides a generalization and integration of some recent polynomial-time approximation schemes for scheduling jobs on unrelated machines [1, 18] and multiprocessor tasks on dedicated processors [2], respectively, with the average weighted completion time objective, since the latter models are included as special cases in our problem.

Keywords: Parallel processing, scheduling, multiprocessor tasks.

1 Introduction

In this paper we address multiprocessor scheduling problems, where a set of n tasks has to be executed by a set of m processors such that each processor can work on at most one task at a time and a task can (or may need to be) processed simultaneously by several processors. Here we assume that m is fixed and the objective is to minimize the average weighted completion time $\sum w_j C_j$, where C_j denotes the completion time of task j. In the *dedicated* variant of this model, denoted by $Pm|fix_j|\sum w_j C_j$, each task requires the simultaneous use of a prespecified set of processors. In the *parallel* variant, denoted by $Pm|size_j|\sum w_j C_j$, the multiprocessor architecture is disregarded and for each task there is given a prespecified number which indicates that the task can be processed by any subset of processors of the cardinality equal to this number. In the *general* model

* Supported in part by DFG - Graduiertenkolleg "Effiziente Algorithmen und Mehrskalenmethoden" and by EU project APPOL "Approximation and On-line Algorithms", IST-1999-14084

$Pm|set_j|\sum w_jC_j$, each task can have a number of *alternative modes*, where each processing mode is specified by a subset of processors and the execution time of the task on that particular processor set.

Previous results: Variants of these problems have been studied, but the previous research has mainly focused on the objective of minimizing the makespan $C_{\max} = \max_{j=0}^{n-1} C_j$. Regarding the worst-case time complexity, it is known that $P5|size_j|C_{max}$ [11], $P3|fix_j|C_{max}$ [16] and $P3|set_j|C_{max}$ [16] are strongly NP-hard. However, there is a polynomial-time approximation scheme (PTAS) for $Pm|fix_j|C_{\max}$ [3], and there is a PTAS for $Pm|set_j|C_{\max}$ [10, 17].

The first PTAS for a strongly NP-hard scheduling problem minimizing the average weighted completion time was given for scheduling jobs on identical parallel machines $P||\sum w_jC_j$ [18]. Then recently it was proved in [1] that there are PTASes for many different variants of classical scheduling problems. These results include scheduling on unrelated parallel machines $Rm||\sum w_jC_j$ [4]. In the multiprocessor setting, in contrast to the makespan objective, only few approximation results are known. Furthermore, they concern only minimizing the sum of completion times: There are a 2-approximation algorithm for $P2|fix_j|\sum C_j$ [8], a 32-approximation algorithm for $P|size_j|\sum C_j$ [20], and - as it was shown recently - a PTAS for $Pm|fix_j|\sum C_j$ [2].

New results: The problem of scheduling general multiprocessor tasks $Pm|set_j|\sum w_jC_j$ can be also viewed as a generalization of two well (but mainly independently) studied scheduling problems: scheduling tasks on unrelated parallel machines $Rm||\sum w_jC_j$ and multiprocessor task scheduling with dedicated processors $Pm|fix_j|\sum w_j\,C_j$. In the case of unrelated machines, for each task (job) there are m processing modes, each with a single processor (machine). In the case of dedicated processor sets, each task has only a single processing mode but including (typically) several processors. Since both of the above special cases are strongly NP-hard [7, 8] for general weights and $m \geq 2$, even if there are only a constant number of processors, it is natural to study how closely the optimum can be approximated by efficient algorithms.

In this paper, focusing on the case where m is fixed, we integrate many of the above mentioned recent results that have shown the existence of PTASs for the two special cases, by providing the following generalization:

Theorem 1. *There is a PTAS for $Pm|set_j|\sum w_jC_j$ that computes, for any fixed m and $\epsilon > 0$ accuracy, a $(1+\varepsilon)$-approximate solution in $O(n \log n)$ time.*

The actual running time of the proposed algorithm depends exponentially on both m and $1/\varepsilon$. However, the above result is, in some sense, the strongest possible one someone can expect. First, it shows the existence of a PTAS for a problem with fixed parameter m that cannot have a fully PTAS [14]. Second, following the ideas in [16] and by using the results [5, 6, 12] one can prove that both $P|set_j, p_j = 1|C_{max}$ and $P|set_j, p_j = 1|\sum C_j$ cannot be approximated within a factor of $m^{\frac{1}{2}-\varepsilon}$, neither for some $\varepsilon > 0$, unless P=NP; nor for any $\varepsilon > 0$, unless NP=ZPP. Hence, not only the above results cannot be extended

or generalized for the general variant where m is not fixed, but even substantially weaker approximation results cannot be expected.

Our approach: In this paper we employ a well known idea of *transformations* – simplify instances and schedules with some loss in the objective. We refine some recent approximation techniques developed in [1, 9, 15]. In order to be able to cope with general multiprocessor tasks, we make adjustments of processing times defining the task profiles and classifying the tasks as huge and tiny. This requires the creation of a sequence of gaps in the schedule, where all processors are idle for a certain period of time. In our approach, we also apply some recent makespan minimization tools from [17] to schedule tasks within single intervals. For handling tiny tasks, we use a linear programming formulation along with some rounding, and observe that in some near optimal schedule tiny tasks of the same profile can be scheduled by Smith's rule [19]. Then, by using the task delaying technique presented in [1], we introduce special *compact instances* in which there is only a constant number of tasks can be potentially scheduled in an interval. The final idea is to use dynamic programming which integrates all the previous components. The obtained PTAS is a combination of the instance transformations and the dynamic programming algorithm.

The paper is organized as follows: In Section 2, we give the definitions and discribe some techniques. In Section 3, we consider the subproblem of scheduling in one subinterval, adopt the PTAS for $Pm|set_j|C_{\max}$ [17], and and discuss some consequences. In Section 4, we present a dynamic programming framework which can be used to integrate all the previous components.

Remarks: This work was motivated by [1], where it was also announced that there is a PTAS for scheduling on unrelated parallel machines with release dates $Rm|r_j|\sum w_jC_j$ (to our best knowledge, the full proof of this result has not yet appeared in literature), and our very recent work [13], where we have shown the existence of a PTAS for $Pm|fix_j, r_j|\sum w_jC_j$. Our original goal was to provide a generalization for all previous results on scheduling problems involving a fixed number of processors (machines), release dates and the average weighted completion time objective. In this paper we don't achieve this goal completely, but provide hopefully a major step towards it by presenting a few novel ideas and techniques that might be also of interest for people working on approximation algorithms for other scheduling problems. For instance, the auxiliary algorithm described in Section 3 is actually a PTAS for $Pm|set_j|C_{\max}$ with the running time $O(n) + O(m, \varepsilon)$ that is better than one in [17].

2 Preliminaries

Formally, for each instance I of our problem we are given sets $\mathcal{T} = \{0, 1, \ldots, n-1\}$ of n tasks and $M = \{1, \ldots, m\}$ of m processors. (Let 2^M denote the set of all subsets of M.) Each task j has a positive weight w_j and an associated function $p_{\cdot j} : 2^M \longrightarrow \mathbb{R}^+ \cup \{+\infty\}$ that gives the execution time $p_{\mu j}$ of task j in terms

of the set of processors $\mu \subseteq M$ which is assigned to j. Given the set $\mu(j) \subseteq M$ allotted to task j, the processors of $\mu(j)$ are required to execute task j in union and without preemption, i.e. they all have to start processing of task j at some *starting time* S_j, and finish at the *completion time* $C_j := S_j + p_{\mu(j)j}$. A feasible schedule σ consists of an *allotment* $\mu(j)$ and a starting time S_j for each task $j \in \mathcal{T}$ such that no processor executes more than one task at each time step. Then, the value $C_w(\sigma) := \sum_{j=0}^{n-1} w_j C_j(\sigma)$ is called the *average weighted completion time* of σ. Here, the number of processors m is considered to be a fixed constant, and the objective is to find a feasible schedule that minimizes the average weighted completion time.

Further, we write $\sigma(I)$ to denote a schedule σ with respect to an instance I. Accordingly, a schedule $\sigma_{opt}(I)$ is called *optimal* if $C_w(\sigma_{opt}(I)) = OPT(I)$, where $OPT(I)$ is the minimal average weighted completion time for I. Given a schedule σ, a schedule σ' is called an *ε-schedule* of σ if $C_w(\sigma')/C_w(\sigma) \leq 1 + K\varepsilon$, where K is some constant. We say that one can transform *with $1 + O(\varepsilon)$ loss the objective and in polynomial time* an instance I into instance $\tilde{I}$ if the following holds: $\tilde{I}$ is obtained form I in $p(|I|)$ elementary operations for some polynomial $p(\cdot)$, any feasible schedule of $\tilde{I}$ is also feasible of I and, $OPT(\tilde{I})/OPT(I) \leq 1 + K\varepsilon$, where K is some constant.

We will show below that with $1 + O(\varepsilon)$ loss, any instance I of our problem can be transformed in $O(n \log n)$ time into a special *compact* instance $\tilde{I}$ such that one can find in $O(n)$ time an ϵ-schedule of $\sigma_{opt}(\tilde{I})$. Clearly, this suffices to obtain a PTAS for our original problem.

2.1 Basic techniques

For any fixed accuracy $\varepsilon > 0$, we assume w.l.o.g. that $\log_{1+\varepsilon}(1 + \frac{1}{\varepsilon})$ and $\frac{1}{\varepsilon}$ are integral. We partition the time interval $(0, \infty)$ into disjoint intervals I_x of the form $[R_x, R_{x+1})$, where $R_x = (1 + \varepsilon)^x$ and $x \in \mathbb{Z}$. Notation I_x will also be used to refer to the length εR_x of the interval, thus $I_x = R_{x+1} - R_x = \varepsilon R_x$ and $I_{x+1} = (1 + \varepsilon) I_x$. For a schedule σ, let $y(j) \leq z(j)$ be those indices for which $S_j \in I_{y(j)}$ and $C_j \in I_{z(j)}$, respectively.

In this paper, for a given schedule σ, we always use the constructive method to find ϵ-schedules, i.e. we show directly how tasks can be rescheduled in σ without dramatically increasing the value of the objective function. Next, we describe some techniques that will be applied later. Knowing σ, we construct a new schedule σ' as follows:

Stretching: Set $\mu'(j) = \mu(j)$ and $S'_j = (1 + \varepsilon) C_j - p_{\mu(j)j}$ for each task $j \in \mathcal{T}$. This generates $\varepsilon p_{\mu(j)j}$ idle time on $\mu(j)$.

Rearranging: Set $\mu'(j)$ and S'_j for each task $j \in \mathcal{T}$ such that a new schedule σ' is feasible and $z'(j) = z(j)$, i.e. tasks are rescheduled preserving $C'_j \in I_{z(j)}$. This gives us the ability to rearrange tasks inside intervals. Note that at any time then we use this technique we have to specify the way of rearranging.

Shifting*: Set $\mu'(j) = \mu(j)$ and $S'_j = S_j - R_{y(j)} + R_{y(j)+1}$ for each task $j \in \mathcal{T}$, i.e. in σ' we have $y'(j) = y(j) + 1$, and hence the distance between S_j and the beginning of interval $I_{y(j)}$ is preserved. This generates $\varepsilon(R_{z(j)} - R_{y(j)})$ additional

idle time on $\mu'(j)$ after the completion time C'_j.
Shifting**: Set $\mu'(j) = \mu(j)$ and $S'_j = S_j - R_{z(j)} + R_{z(j)+1}$ for each task $j \in \mathcal{T}$, i.e. $z'(j) = z(j) + 1$, and hence the distance between C'_j and the end of interval $I_{z(j)}$ is preserved. This generates $\varepsilon(R_{z(j)} - R_{y(j)})$ additional idle time on $\mu(j)$ before the start time S'_j.

Proposition 1. *If a schedule σ' is obtained from a schedule σ by stretching, rearranging, or both types of shifting then σ' is an ε-schedule of σ.*

The following lemma that will be used throughout the paper.

Lemma 1. *For any schedule σ there is an ϵ-schedule σ' of σ such that the following holds in σ': each task j starts not earlier than $\varepsilon p_{\mu'(j)j}$; each task j crosses at most a constant number $s^*(\varepsilon) := \log_{1+\varepsilon}(1 + \frac{1}{\varepsilon})$ of intervals; each task j with $p_{\mu'(j)j} \leq \varepsilon^2 I_{y'(j)}$ is non-crossing; each crossing task j starts at one of points $R_{y'(j)} + i\varepsilon^2 I_{y'(j)}$ of $I_{y'(j)}$, where $i \in \{0, 1, \ldots, \frac{1}{\varepsilon^2} - 1\}$.*

2.2 Blocks, gaps and profiles

A *block* $\mathcal{B}_i$ is a set $\{I_{b(i)}, \ldots, I_{b'(i)}\}$ of consecutive intervals. A *block structure* $\mathcal{B}_1, \mathcal{B}_2, \ldots$ is a sequence of blocks such that any block $\mathcal{B}_i$ is of the length $\delta_i := b'(i) - b(i) + 1$ which is is at least $2s^*(\varepsilon)$ and at most $2\delta(\varepsilon)$, any pair $\mathcal{B}_i$, $\mathcal{B}_{i+1}$ of consecutive blocks shares exactly one interval $I_{b'(i)} = I_{b(i)}$ (i.e. $b'(i) = b(i+1)$) and the total length of these two blocks $\delta_i + \delta_{i+1} \geq \delta(\varepsilon)$, where where $\delta(\varepsilon) := \frac{2s(\varepsilon)}{\varepsilon}$. For a schedule σ, a *gap* is an interval in time where all the processors of M are idle in σ. The following lemma shows that with any schedule one can associate a block structure such that there is a gap in the beginning and the end of each block.

Lemma 2. *For a schedule σ there is an ϵ-schedule σ' of σ such that there exists a block structure $\mathcal{B}_1, \mathcal{B}_2, \ldots$ for the schedule σ' with the property that for each dynamic block $\mathcal{B}_i$ there is a gap in the first interval $I_{b(i)}$. This gap starts at one of the points $R_{b(i)} + \ell\epsilon^2 I_{b(i)}$, $\ell = 1 \ldots, \frac{1}{\epsilon^2} - 1$, and has a length which is at least $\epsilon^2 I_{b(i)}$.*

For an instance I and task $j \in \mathcal{T}$, let $p_j^{\min} := \min_{\mu \in 2^M} p_{\mu j}$ be the minimal processing time needed to execute task j. The value $p_j^{\min}$ is called the *length* of j. Then, the following holds.

Lemma 3. *With $1 + O(\varepsilon)$ loss one can transform in $O(n)$ time an instance I into instance $\tilde{I}$ such that for each task $j \in \mathcal{T}$ and $\mu \in 2^M$ the following holds in $\tilde{I}$: the processing time $\tilde{p}_{\mu j} = (1+\varepsilon)^{\omega_{\mu j}}$, where $\omega_{\mu j} \in \mathbb{N} \cup +\infty$; if $\tilde{p}_{\mu j} \neq \infty$ then $\tilde{p}_{\mu j} \leq h(m, \epsilon)\tilde{p}_j^{\min}$, where $h(m, \varepsilon)$ is a constant that depends on m and ε. Furthermore, in $\tilde{I}$ the quotients $\tilde{p}_j^{\min}/\tilde{w}_j$ are different for all tasks in $\mathcal{T}$.*

For simplicity, we will use the following notations throughout the paper. By using the above Lemma, for a task $j \in \mathcal{T}$, the corresponding 2^m-tuple $\omega(j) =<$

$\omega_{\mu j} >_{\mu \in 2^M}$ is called the *execution profile* of j. Accordingly, the set of all distinct possible execution profiles is denoted by $\Omega := \{\omega(j) \mid j \in \mathcal{T}\}$. Then, it is not hard to prove that the number of profiles in Ω is bounded by

$$\nu(m, \epsilon) := \lceil 2 + \log_{1+\epsilon} h(m, \epsilon) \rceil^{2^m}.$$

In addition, for a given instance I, we introduce an index $x(j)$ for each task $j \in \mathcal{T}$ that corresponds to the interval $I_{x(j)}$ earlier which processing of j can not be started. (By using Lemma 1, we assume that $R_{x(j)} \leq \epsilon p_j^{\min} \leq R_{x(j)+1}$ for each task $j \in \mathcal{T}$. However, we revise this property in Lemma 10.) Accordingly, a task j is called *huge* if $p_j^{\min} \geq \varepsilon^2 I_{x(j)}/q^*$, and *tiny* if $p_j^{\min} < \varepsilon^2 I_{x(j)}/q^*$, where the parameter $q^* = q^*(m, \varepsilon) \gg 2^m h(m, \varepsilon)$ is specified later in Section 3.2. To distinguish sets of huge and tiny tasks we write $\mathcal{HT}$ and $\mathcal{TT}$, respectively.

For a task set $\mathcal{X} \subseteq \mathcal{T}$, we will use $p^{\min}(\mathcal{X}) := \sum_{j \in \mathcal{X}} p_j^{\min}$ to denote the total length of the tasks in $\mathcal{X}$, use $\mathcal{X}^\omega$ to denote the tasks of $\mathcal{X}$ of profile $\omega \in \Omega$, and $\mathcal{X}_x$ to denote the set of tasks $j \in \mathcal{X}$ with $x(j) = x$. To indicate that $\mathcal{X}$ is associated with particular instance I, we will write $\mathcal{X}(I)$. Finally, regarding the objective function we use the following.

Proposition 2. *For any instance I, with at most $1 + \epsilon$ loss one can consider the objective function $\sum w_j R_{z(j)}$ instead of the original function $\sum w_j C_j$.*

3 Scheduling inside interval

In the first part of this section, we consider the problem of scheduling in a single interval I_x. More precisely, we present an algorithm that schedules non-crossing tasks with respect to a known schedule of crossing tasks. In order to achieve this, we generalize the PTAS for $Pm|set_j|C_{\max}$ [17]. Basing on some features of the algorithm, we define the value of parameter $q^* = q^*(m, \varepsilon)$ for tiny tasks, and consider the algorithm as a subroutine of the rearranging technique. After that, we show that tiny tasks are all small corresponding to intervals and can be scheduled by Smith's rule [19]. In the last part of this section, we show how an instance of our problem can be transformed into a *compact* instance in which there is only a a constant number of tasks that can be potentially scheduled in an interval.

3.1 Long and short tasks, relative schedules and LP formulation

Let $\sigma(I)$ be a feasible schedule for instance I. For an interval I_x of σ, the following sets of tasks (running in I_x) are defined: set $\mathcal{W}_x$ of non-crossing tasks and set $\mathcal{K}_x = K_x^1 \cup K_x^2 \cup K_x^3$ of crossing tasks, where K_x^1 is the set of *incoming* crossing tasks (that complete in I_x), K_x^2 is the set of *outgoing* tasks (that start in I_x), and K_x^3 is the set of *throughgoing* tasks (that go through I_x). Suppose that we have applied shifting** to $\sigma(I)$ and obtained $\sigma'(I)$. W.l.o.g. assume that in the schedule $\sigma'(I)$ the tasks of $\mathcal{W}_x$ and $\mathcal{K}_x$ run in interval I_{x+1}. Then, there is at

least εI_x idle time on the processors, between the tasks of $\mathcal{W}_x$ and the outgoing tasks of K_x^2 or between the tasks of $\mathcal{W}_x$ and the end of interval R_{x+2}.

Assume for simplicity that $\mathcal{W}_x = \{0, \ldots, n'\}$ with $p_0^{\min} \geq p_1^{\min} \geq \ldots \geq p_{n'}^{\min}$ and $K_x^3 = \emptyset$. Then, it holds that $p^{\min}(\mathcal{W}_x) \leq mI_x$. Furthermore, we fix allotment $\mu(j)$ for each crossing task $j \in \mathcal{K}_x$ and define $p_{\mu(j)j} := C_j - R_x$ for each task $j \in K_x^1$ and $p_{\mu(j)j} := R_{x+1} - S_j$ for each task $j \in K_x^2$.

Partition $\mathcal{W}_x$ into two subsets $\mathcal{L}_x = \{0, \ldots, k_x - 1\}$ and $\mathcal{S}_x = \mathcal{W}_x \setminus \mathcal{L}_x$, where the appropriate value for $k_x = k_x(m, \varepsilon) \gg 1$ will be specified later in Lemma 6. Tasks in $\mathcal{L}_x$ and $\mathcal{S}_x$ are called *long* and *short*, respectively. Let $\mathcal{J}_x = \mathcal{L}_x \cup \mathcal{K}_x$ be the set of long and crossing tasks.

A *processor assignment* of $\mathcal{J}_x$ is a mapping $\mu_x : \mathcal{J}_x \longrightarrow 2^M$ such that $\mu_x(j) = \mu(j)$ for each task $j \in K_x^1 \cup K_x^2$. Two tasks T_k and T_ℓ are called *compatible*, if $\mu_x(k) \cap \mu_x(\ell) = \emptyset$. A *snapshot* of $\mathcal{J}_x$ is a set of compatible tasks of $\mathcal{J}_x$. A *relative schedule* of $\mathcal{J}_x$ is a pair $\mathcal{R}_x$ of a processor assignment μ_x of $\mathcal{J}_x$ and a sequence of snapshots $\mathcal{M}_1, \ldots, \mathcal{M}_g$ of $\mathcal{J}_x$ with respect to μ_x, such that each task $j \in \mathcal{J}_x$ occurs in a subsequence of consecutive snapshots $\mathcal{M}_{u_j}, \ldots, \mathcal{M}_{v_j}$, $1 \leq u_j \leq v_j \leq g$, where any two consecutive snapshots $\mathcal{M}_\ell$ and $\mathcal{M}_{\ell+1}$ are different and it holds that $u_j = 1$ if $j \in K_x^1$, $v_j = g$ if $j \in K_x^2$ (see Figure 1). Then, it is not hard to see that the number of snapshots g is bounded by $2(k_x + m)$ and the number of different (appropriate) relative schedules of $\mathcal{J}_x$ is bounded by a constant depending only on m and the number k_x of long tasks in $\mathcal{L}_x$.

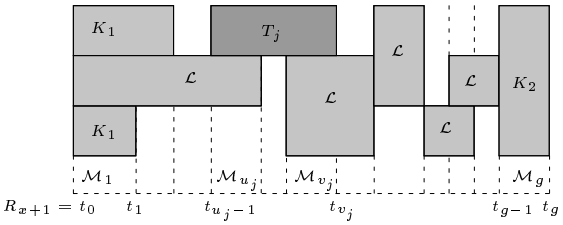

Fig. 1. A relative schedule $\mathcal{R}_x$

Given a set $\mu \subseteq M$, a *μ-configuration* C_μ is a partition of μ into non-empty sets. Let N_μ be the total number of μ-configurations and let $C_{1,\mu}, \ldots, C_{N_\mu,\mu}$ be all μ-configurations. Note that $N_M = B(m)$, where $B(m) \leq m!$ is the mth Bell number.

Consider a relative schedule $\mathcal{R}_x$, where the processor assignment μ_x such that no task j in $\mathcal{J}_x$ has the assigned processing time $p_{\mu_x(j)j} = \infty$. Then, let $F_\ell := M \setminus \bigcup_{j \in \mathcal{M}_\ell} \mu_x(j)$ denotes the set of free processors in snapshot $\mathcal{M}_\ell$, $1 \leq \ell \leq g$. For a $\nu(m, \epsilon)$-vector $D := (D^\omega)_{\omega \in \Omega}$, a relative schedule $\mathcal{R}_x$ is called feasible for D if for any $D^\omega \neq 0$ there exists a snapshot $\mathcal{M}_\ell$ in $\mathcal{R}_x$ such that $\mu \subseteq F_\ell$ and $\omega_\mu \neq +\infty$.

Now assume there is a set of tasks $\mathcal{X} \subseteq \mathcal{T}$ such that $p^{\min}(\mathcal{X}^\omega) = D^\omega$ for each profile $\omega \in \Omega$. (Recall that if a task $j \in \mathcal{X}^\omega$ is assigned to a processor

set μ, then the processing time of j on μ is equal to $p_j^{\min}(1+\varepsilon)^{\omega_\mu}$. Thus, if all tasks of $\mathcal{X}^\omega$ are assigned to μ, the total processing time needed to execute them is equal to $D^\omega(1+\varepsilon)^{\omega_\mu}$.) Then, in order to find a preemptive schedule of $\mathcal{X}$ with respect to the relative schedule $\mathcal{R}_x$, one can formulate the following linear program $LP(\mathcal{R}_x, D)$ in terms of variables associated with F_ℓ-configurations:

$$
\begin{array}{lll}
\textbf{Minimize} & t_g - t_0 & \\
\textbf{s.t.} & (\mathbf{0}) & t_0 = R_{x+1}, \\
& (\mathbf{1}) & t_\ell \ \geq \ t_{\ell-1}, \quad \ell = 1, \ldots, g, \\
& (\mathbf{2}) & t_{v_j} - t_{u_j - 1} \ = \ p_{\mu_x(j)j}, \quad \forall j \in \mathcal{J}_x, \\
& (\mathbf{3}) & \sum_{i=1}^{N_{F_\ell}} x_{i,F_\ell} \ = \ t_\ell - t_{\ell-1}, \quad \ell = 1, \ldots, g, \\
& (\mathbf{4}) & \sum_{\ell=1}^{g} \sum_{i \,:\, \mu \in C_{i,F_\ell}} x_{i,F_\ell} \ \geq \ D(\mu) := \sum_{\omega \in \Omega} D^\omega (1+\varepsilon)^{\omega_\mu} y_{\mu,\omega}, \quad \forall \mu \in 2^M, \\
& (\mathbf{5}) & \sum_{\mu \in 2^M} y_{\mu,\omega} = 1, \quad \forall \omega \in \Omega, \\
& (\mathbf{6}) & x_{i,F_\ell} \ \geq \ 0, \quad \ell = 1, \ldots, g, \ i = 1, \ldots, N_{F_l}, \\
& (\mathbf{7}) & y_{\mu,\omega} \ \geq 0, \quad \forall \omega \in \Omega \ \forall \mu \in 2^M,
\end{array}
$$

where the variables have the following interpretation:

t_ℓ: the time when snapshot $\mathcal{M}_\ell$ ends and $\mathcal{M}_{\ell+1}$ starts. The starting time of the schedule is denoted by $t_0 = R_{x+1}$ and the finishing time by t_g (see Figure 1).

x_{i,F_ℓ}: the length of the configuration C_{i,F_ℓ} in snapshot $\mathcal{M}_\ell$. During an interval of length x_{i,F_ℓ} the tasks of $\mathcal{X}$ can be executed on processor subsets $\mu \in C_{i,F_\ell}$,

$y_{\mu,\omega}$: the assignment variable indicating which part of the tasks of $\mathcal{X}^\omega$ is assigned to be executed on a processor set $\mu \in 2^M$, reflecting that the total processing time needed to execute such tasks of $\mathcal{X}^\omega$ is equal to $D^\omega(1+\varepsilon)^{\omega_\mu} y_{\mu,\omega}$,

$D(\mu)$: the total processing time needed to execute the tasks of $\mathcal{X}$ assigned to a processor set $\mu \in 2^M$.

Less formally, the constraints **(0)-(2)** define a schedule structure with respects to the relative schedule $\mathcal{R}_x$, the constraint **(3)** defines a substructure of configurations inside the snapshots of $\mathcal{R}_x$, the constraints **(4)-(5)** define the balance between the free time available in the configurations and the allotment of work that have to be executed in order to complete all tasks. (One can also associate each set $\mathcal{X}^\omega$ with one task of the length D^ω and think in terms of preemptive scheduling of these $\nu(m,\varepsilon)$ tasks of different profiles such that execution of parts of the same task on different sets in parallel is allowed.)

Let (t^*, x^*, y^*) be an optimum solution of $LP(\mathcal{R}_x, D)$. Then, the following holds [17].

Lemma 4. *For any two non-negative $\nu(m,\varepsilon)$-vectors a and b such that a relative schedule $\mathcal{R}_x$ is feasible for a and $a+b$, the following inequality holds*

$$t_g^*(LP(\mathcal{R}_x, a)) \leq t_g^*(LP(\mathcal{R}_x, a+b)) \leq t_g^*(LP(\mathcal{R}_x, a)) + \max_{\omega_\mu \neq \infty, \omega \in \Omega} (1+\varepsilon)^{\omega_\mu} \sum_{\omega \in \Omega} b_\omega.$$

Lemma 5. *If a relative schedule $\mathcal{R}_x$ is feasible for the $\nu(m,\varepsilon)$-vector $D = (p^{\min}(\mathcal{S}_x^\omega))_{\omega \in \Omega}$, then the optimal value $t_g^* - t_0^*$ of $LP(\mathcal{R}_x, D)$ is not larger than the makespan of any schedule of $\mathcal{J}_x \cup \mathcal{S}_x$ that respects the relative schedule $\mathcal{R}_x$.*

3.2 Generating a schedule, tiny versus short

Let (t^*, x^*, y^*) be an optimum solution of $LP(\mathcal{R}_x, D)$. We construct a schedule for the tasks in $\mathcal{J}_x$ and $\mathcal{S}_x$ as follows. For each profile ω and $y^*_{\mu,\omega} > 0$ we select the tasks of $\mathcal{S}_x$ in a greedy manner until the total length does not exceed $y^*_{\mu,\omega} > 0$ (if the total length plus the length of the last selected task exceeds $y^*_{\mu,\omega} > 0$ we mark this task as unbalanced, there can be at most 2^m such ones for one profile) and assign them to be executed on the subset μ. After that, starting from the first snapshot, by using the values $x^*_{i,F\ell} > 0$ and accordingly with the subsets $\mu \in C_{i,F_\ell}$, we select the tasks that are assigned to μ in a greedy manner until thier total processing time does not exceed $x^*_{i,F\ell} > 0$. If it happens that for a configuration C_{i,F_ℓ} the selected tasks do not fit inside the interval of the length $x^*_{i,F_\ell} > 0$, we increase $x^*_{i,F_\ell} > 0$ by a small amount $\varepsilon_{i,\ell}$, and accordingly, we also adjust the value t^*_ℓ by the total increase ε_{F_ℓ} (see Figure 2).

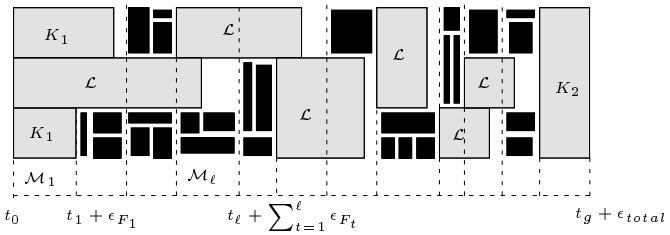

Fig. 2. Generating a schedule

Since there are at most $\nu(m,\varepsilon)$ profiles the total number of unbalanced tasks is at most $\nu(m,\varepsilon)2^m$. Furthermore, since the number of snapshots $g \leq 2(k_x + m)$ and the number of configurations is bounded by the mth Bell number $B(m)$, the total enlargement ε_{total} is at most

$$h(m,\varepsilon)(p^{\min}_{k_x} + p^{\min}_{k_x+1} + \ldots p^{\min}_{k_x+2(k_x+m)B(m)+\nu(m,\varepsilon)2^m-1}),$$

i.e. the maximum processing tasks of the $k_x + 2(k_x+m)B(m) + \nu(m,\varepsilon)2^m$ longest tasks of $\mathcal{S}_x$. The following lemma is a useful tool to select k_x such that ε_{total} is kept small enough [17]:

Lemma 6. *For any interval I_x there is a number k_x with $1 \leq k_x \leq (q+1)^{\frac{mh(m,\varepsilon)}{\varepsilon^2}-1} + d[1 + (q+1) + \ldots + (q+1)^{\frac{mh(m,\varepsilon)}{\varepsilon^2}-2}]$ such that $h(m,\varepsilon)(p^{\min}_{k_x} + \ldots + p^{\min}_{(q+1)k_x+d-1}) \leq \varepsilon^2 I_x$, where $q := 3mB(m)$ and $d := \nu(m,\varepsilon)2^m$.*

The final algorithm follows straightforward: select an appropriate k_x, enumerate all relative schedules together with solving LPs and deducing the corresponding schedules, output the schedule with the smallest makespan.

Now, we define $q^*(m,\varepsilon) := \frac{mh(m,\varepsilon)^2 2^m \nu(m,\varepsilon)}{\varepsilon^2}\left((m+1)^{m+2}+1\right)^{\frac{mh(m,\varepsilon)}{\varepsilon^2}}$. By Lemma 1 we may assume that tiny tasks are non-crossing in $\sigma(I)$. Accordingly,

let $\mathcal{Y}_x \subseteq \mathcal{W}_x$ be the set of tiny tasks that are non-crossing in I_x. Then, the following holds:

Lemma 7. *The number k_x of long tasks in $\mathcal{L}_x$ is not larger than the number of non-crossing huge tasks in I_x with $p_j \geq \frac{\varepsilon^2 I_x}{q^*}$. Hence, all tiny tasks are short, i.e. $\mathcal{Y}_x \subseteq \mathcal{S}_x$.*

Lemma 8. *Let $\tilde{\mathcal{Y}}_x$ be another set of tiny tasks such that for each profile $\omega \in \Omega$, $p^{\min}(\mathcal{Y}_x^\omega) = 0$ implies $p^{\min}(\tilde{\mathcal{Y}}_x^\omega) = 0$, and $p^{\min}(\mathcal{Y}_x^\omega) \neq 0$ implies either $|p^{\min}(\mathcal{Y}_x^\omega) - p^{\min}(\tilde{\mathcal{Y}}_x^\omega)| \leq \frac{\varepsilon I_x}{2h(m,\varepsilon)\nu(m,\varepsilon)}$ or $p^{\min}(\tilde{\mathcal{Y}}_x^\omega)$=0. Then algorithm outputs a feasible schedule of $\tilde{W}_x := (\mathcal{W}_x \setminus \mathcal{Y}_x) \cup \tilde{\mathcal{Y}}_x$ inside I_{x+1} with respect to the schedule of $\mathcal{K}_x$ in $\sigma'(I)$.*

Proof. The proof follows form Lemmas 4,7, Lemma 6 and the fact that the interval I_{x+1} contains at least $\varepsilon I_x > \varepsilon^2 I_x + h(m,\varepsilon) \sum_{\omega \in \Omega} \frac{\varepsilon I_x}{2h(m,\varepsilon)\nu(m,\varepsilon)}$ idle time. (Recall that $\max_{\mu \neq \infty, \omega \in \Omega} p_{\mu j} \leq p_j^{\min} h(m,\varepsilon)$.) □

3.3 Scheduling tiny tasks and compact instances

Now consider the problem of placing tiny tasks in the schedule. In the following we restrict ourselves to the case of scheduling tiny tasks of the same profile, say ω. We say that two tiny tasks $k, \ell \in \mathcal{TT}^\omega$ with $\frac{p_k^{\min}}{w_k} < \frac{p_\ell^{\min}}{w_\ell}$ are *scheduled by Smith's rule* in a schedule σ if it holds either $z(k) \leq z(\ell)$, or $z(\ell) < x(k)$. In other words, if the two tasks are available at the same interval (that means ℓ is not completed before the earliest interval where k can be scheduled), then the task k of smaller value p_k/w_k is scheduled first with respect to intervals. Note that the tasks of $\mathcal{TT}_x^\omega$ have to be scheduled in the order of increasing values $p_j^{\min}/w_j$ [19].

Lemma 9. *For a feasible schedule $\sigma(I)$ there is an ε-schedule $\sigma'(I)$ of $\sigma(I)$ such that for any profile $\omega \in \Omega$, the tiny tasks of $\mathcal{TT}^\omega$ are scheduled by Smith's rule in $\sigma'(I)$.*

Proof. W.l.o.g. let I_1, I_L be the first and the last interval in a feasible schedule σ. By Lemma 1, assume that all tiny tasks are non-crossing in σ and by Lemma 3 all quotients p_j/w_j are different. Let $\mathcal{Y}_x^\omega$ be the set of tiny tasks of profile ω that are scheduled in interval I_x and let $D^\omega := p^{\min}(\mathcal{Y}_x^\omega)$ be the total length of these tasks in I_x. Consider the following $LP(\omega)$:

$$
\begin{array}{lll}
\text{Minimize} & \sum_{j\,:\, j \in \mathcal{TT}^\omega} w_j \sum_{x=x(j)}^{L} y_{j,x} R_x & \\
\text{s.t.} \quad (1) & \sum_{x=x(j)}^{L} y_{j,x} \;=\; 1, & \forall j \in \mathcal{TT}^\omega, \\
(2) & \sum_{j\,:\, j \in \mathcal{TT}^\omega,\; x(j) \leq x} y_{j,x}\, p_j^{\min} \leq D^\omega, & \forall I_x, \\
(3) & y_{j,x} \;\geq\; 0, & \forall j \in \mathcal{TT}^\omega,\; x = x(j), \ldots, L.
\end{array}
$$

First, one can find a feasible solution of $LP(\omega)$ by assigning the values y with respect to the schedule σ, i.e. for each task $j \in \mathcal{TT}^\omega$ set $y_{j,x} = 1$ if $z(j) = x$ and

$y_{j,x} = 0$ otherwise. Hence, the objective value of the linear program $LP(\omega)$ is not larger than the weighted average completion time for the tiny tasks of $\mathcal{TT}^\omega$ in σ. In other words, the value of an optimal fractional solution is a lower bound of the weighted completion time $\sum_{j\in\mathcal{TT}^\omega} w_j R_{z(j)}$. Consider an optimal solution $(y^*_{j,x})$ of the linear program. Suppose that there are two tasks ℓ and k that are not scheduled by Smith's rule. W.l.o.g. we can consider the case when $y^*_{\ell,x_\ell} > 0$, $y^*_{k,x_k} > 0$, $x(k) \le x(\ell) \le x_\ell < x_k$ and $\frac{p_k^{\min}}{w_k} < \frac{p_\ell^{\min}}{w_\ell}$.

Then, there exist values z_ℓ and z_k such that $0 < z_\ell \le y^*_{\ell,x_\ell}$ and $0 < z_k \le y^*_{k,x_k}$ and $z_\ell p_\ell^{\min} = z_k p_k^{\min}$. Now we exchange parts of the variables:

$$y'_{\ell,x_\ell} = y^*_{\ell,x_\ell} - z_\ell \quad y'_{\ell,x_k} = y^*_{\ell,x_k} + z_\ell$$
$$y'_{k,x_k} = y^*_{k,x_k} - z_k \quad y'_{k,x_\ell} = y^*_{k,x_\ell} + z_k$$

and $y'_{j,x} = y^*_{j,x}$ for the remaining variables.

The new solution $(y'_{j,x})$ is feasible and the objective value $\sum_{j\in\mathcal{TT}^\tau} w_j \sum_{x=x(j)}^{L} y'_{j,x}R_x$ is equal to $\sum_{j\in\mathcal{TT}^\tau} w_j \sum_{x=x(j)}^{L} y^*_{j,x}R_x + R_{\ell,k}$, where $R_{\ell,k} = (R_{x_\ell} - R_{x_k})(w_k z_k - z_\ell w_\ell)$. By $z_k = z_\ell \frac{p_\ell^{\min}}{p_k^{\min}}$, the second factor $(w_k z_k - z_\ell w_\ell) = z_\ell(w_k \frac{p_\ell^{\min}}{p_k^{\min}} - w_\ell)$. Since $\frac{p_k^{\min}}{w_k} < \frac{p_\ell^{\min}}{w_\ell}$ and $z_\ell > 0$, the second factor is larger than 0. The inequality $x_\ell < x_k$ implies $R_{x_\ell} < R_{x_k}$ and $R_{\ell,k} < 0$. In other words, the new solution $(y'_{j,x})$ has a lower objective value and gives us a contradiction. This means that the two tasks ℓ and k are scheduled by Smith's rule.

Now we use some properties of the above linear program. There is an optimal solution such that for each interval I_x we have at most one task $j_x \in \mathcal{TT}^\omega$ with $x_{j,x} \in (0,1)$ and that is assigned for the first time. Otherwise we can use the same argument as above (and the fact that the quotients $p_j^{\min}/w_j$ are different) to improve the objective value. Furthermore, such an optimal solution can be obtained by applying Smith's rule in a greedy manner. To turn such a fractional solution into an integral assignment of tiny tasks to intervals, we need only to increase the values $D^\omega = p^{\min}(\mathcal{Y}^\omega_x)$ by $\frac{\varepsilon^2 I_x}{q^*}$. Then, j_x fits completely into I_x. Let $\tilde{\mathcal{Y}}^\omega_x$ be the set of tiny tasks of profile ω that is assigned to an interval I_x with respect to the integral assignment of $LP(\omega)$ (including the task j_x). For each interval I_x of schedule σ we consider the set $\tilde{\mathcal{Y}}_x = \cup_{\omega\in\Omega}\tilde{\mathcal{Y}}^\omega_x$ of all tiny tasks that corresponds to $LP(\omega)$, $\omega \in \Omega$. It is not hard to check that the conditions of Lemma 8 hold. Thus, we apply the shifting* to σ, replace the sets $\mathcal{Y}_x$ by $\tilde{\mathcal{Y}}_x$ and apply the rearranging technique to each interval I_{x+1}. The final schedule is constructed.

To finish the proof, there are two facts. First, the total weighted average completion time of the schedule of tiny tasks which is given by the optimal fractional solutions of $LP(\omega)$, $\omega \in \Omega$ is less than their corresponding total weighted average completion time in σ. Second, we have used only the shifting and rearranging techniques. □

By using the above Lemma and the ideas from [1] we get:

Lemma 10. *With $1+O(\varepsilon)$ loss and in $O(n \log n)$ time, one can transform an instance I into instance $\tilde{I}$ such that for each profile $\omega \in \Omega$ the following holds: $|\mathcal{T}(I)_x| \leq N^*(m,\varepsilon)$ and $p^{\min}(\mathcal{T}(I)_x) \leq \frac{N^*(m,\varepsilon)}{\varepsilon^3} I_x$, where $N^*(m,\varepsilon)$ is a constant that depends on m and ε.*

4 The dynamic programming algorithm

As we mentioned it above, we will use a dynamic program to compute approximate solutions for our scheduling problem. To be able to formulate this dynamic program and show that the algorithm solving it has the desired running time, we need only one final result:

Lemma 11. *For a feasible schedule $\sigma(I)$ there is an ε-schedule $\sigma'(I)$ such that in $\sigma'(I)$ the tasks in $\mathcal{T}(I)_x$ are scheduled within a constant number $d^*(m,\varepsilon)$ of intervals that follow I_x, i.e. for each task $j \in \mathcal{T}$ it holds that $z'(j) - x(j) \leq d^*(m,\varepsilon)$.*

To finish the proof of our main result, the last step is to use dynamic programming with blocks as units. By using Lemmas 1,10 and Lemma 11 there is at most a constant number of tasks in $\mathcal{T}_x$ for each interval I_x, all of these tasks have to be scheduled within a constant number of intervals that follow I_x, hence one can efficiently enumerate all possible schedules of the tasks of $\mathcal{T}_x$. These facts make a dynamic programming straightforward. However, here (due to space limitation) we omit the formal description, leaving it to the full version of the paper or referring to ones in [2, 13].

References

1. F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Millis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko, Approximation schemes for minimizing average weighted completion time with release dates, *Proceedings 40th IEEE Symposium on Foundations of Computer Science* (1999), 32-43.
2. F. Afrati, E. Bampis, A. V. Fishkin, K. Jansen, C. Kenyon, Scheduling to minimize the average completion time of dedicated tasks, *Proceedings 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, LNCS 1974, Springer Verlag (2000), 454-464.
3. A. K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis, Scheduling independent multiprocessor tasks, *Proceedings 5th European Symposium on Algorithms*, LNCS 1284, Springer Verlag (1997), 1-12.
4. F. Afrati, E. Bampis, C. Kenyon and I. Milis, A PTAS for the average weighted completion time problem on unrelated machines, *Journal of Scheduling* 3 (2000), 323-332.
5. A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir, On chromatic sums and distributed resource allocation, *Information and Computation* 140 (1998), 183-202.
6. A. Bar-Noy and M. M. Halldórsson and G. Kortsarz and R. Salman and H. Shachnai, Sum multicoloring of graphs, *Proceedings 7th European Symposium on Algorithms*, LNCS 1643, Springer Verlag (1999), 390-401.

7. J. L. Bruno, E. G. Coffman, Jr., R. Sethi, Algorithms for minimizing mean flow time, *Proceedings of the IFIP Congress*, North-Holland, Amsterdam, (1974), 504-510.
8. X. Cai, C.-Y. Lee, and C.-L. Li, Minimizing total completion time in two-processor task systems with prespecified processor allocation, *Naval Research Logistics* 45 (1998), 231-242.
9. S. Chakrabarti, C. A. Philips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein, Improved scheduling algorithms for minsum criteria, *Proceedings 23rd International Colloquium on Automata, Languages and Programming*, LNCS 1099, Springer Verlag (1996), 646-657.
10. J. Chen and A. Miranda, A polynomial time approximation scheme for general multiprocessor job scheduling, *Proceedings 31st ACM Symposium on the Theory of Computing* (1999), 418-427.
11. J. Du and J. Leung, Complexity of scheduling parallel task systems, *SIAM Journal on Discrete Mathematics* 2 (1989), 473-487.
12. U. Feige and J. Kilian, Zero-knowledge and the chromatic number, in *Journal of Computer and System Science* 57(2) (1998), 187-199.
13. A. V. Fishkin, K. Jansen, L. Porkolab, On minimizing average weighted completion time of multiprocessor tasks with release dates, to appear on *ICALP01*.
14. M. R. Garey and D. S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, Freeman, San Francisco, CA, 1979.
15. L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, Scheduling to minimize average time: Offline and online algorithm, *Mathematics of Operation Research* 22 (1997), 513-544.
16. J. A. Hoogeveen, S. L. Van de Velde, and B. Veltman, Complexity of scheduling multiprocessor tasks with prespecified processor allocations, *Discrete Applied Mathematics* 55 (1994), 259-272.
17. K. Jansen and L. Porkolab, General multiprocessor task scheduling: approximate solution in linear time, *Proceedings 6th International Workshop on Algorithms and Data Structures*, LNCS 1663, Springer Verlag (1999), 110-121.
18. M. Skutella and G. J. Woeginger, A PTAS for minimizing the weighted sum of job completion times on parallel machines, *Proceedings 31st ACM Symposium on Theory of Computing* (1999), 400-407.
19. W. E. Smith, Various optimizers for single-stage production, *Naval Research Logistic Quarterly* 3 (1956), 59-66.
20. J. Turek, W. Ludwig, J. Wolf, and P. Yu, Scheduling parallel tasks to minimize average response times, *Proceedings 5th ACM-SIAM Symposium on Discrete Algorithms* (1994), 112-121.