Minimizing Weighted Mean Completion Time for Malleable Tasks Scheduling

Olivier Beaumont, Nicolas Bonichon, Lionel Eyraud-Dubois
INRIA and University of Bordeaux

Loris Marchal
CNRS and ENS Lyon

***Abstract*—Malleable tasks are jobs that can be scheduled with preemptions on a varying number of resources. We focus on the special case of work-preserving malleable tasks, for which the area of the allocated resources does not depend on the allocation and is equal to the sequential processing time. Moreover, we assume that the number of resources allocated to each task at each time instant is bounded. We consider both the clairvoyant and non-clairvoyant cases, and we focus on minimizing the weighted sum of completion times. In the weighted non-clairvoyant case, we propose an approximation algorithm whose ratio (2) is the same as in the unweighted non-clairvoyant case. In the clairvoyant case, we provide a normal form for the schedule of such malleable tasks, and prove that any valid schedule can be turned into this normal form, based only on the completion times of the tasks. We show that in these normal form schedules, the number of preemptions per task is bounded by 3 on average. At last, we analyze the performance of greedy schedules, and prove that optimal schedules are greedy for a special case of homogeneous instances. We conjecture that there exists an optimal greedy schedule for all instances, which would greatly simplify the study of this problem. Finally, we explore the complexity of the problem restricted to homogeneous instances, which is still open despite its simplicity.**

Keywords: Scheduling, Independent Tasks Scheduling, Approximation Algorithms, Non Clairvoyant Algorithms, Weighted Mean Completion Time, Malleable Tasks.

I. Introduction and Related Works

Parallel tasks models have been introduced in order to deal with the complexity of explicitly scheduling communications. In the most general setting, a parallel task comes with its completion time on any number of resources (see [1], [2], [3] and references therein for complete surveys). The tasks may be either *rigid* (when the number of processors for executing a dedicated code is fixed), *moldable* (when the number of processors is fixed for the whole execution, but may take several values) or *malleable* [4] (when the number of processors may change during the execution due to preemptions).

In this paper, we concentrate on the special case of malleable tasks that are *work-preserving*, *i.e.* parallel tasks such that the overall work (that corresponds to the area allocated to a task in a Gantt Chart) does not depend on the number of processors allocated to a task during its execution. This corresponds to ideal parallel tasks. Although malleability requires advanced capacities of the runtime environment, it may be well suited to multicore machines and it may be used in other contexts, such as sharing a large bandwidth out of a server between small connexions. It is worth noting that in the context of multicore machines as in the context of bandwidth sharing, it is natural to assume that a given task cannot make use of all available resources (since in order to be work-preserving, a task should be allocated on a single multicore processor, or since the bandwidth achievable between a large capacity server and a distant node is typically bounded by the bandwidth of the distant node). Therefore, it is natural to associate to each task the maximal number of resources that it can use simultaneously, that will be denoted by δ in the rest of the paper.

We focus on the following scheduling problem: (i) the system is made of P identical processors on which (ii) n malleable work-preserving tasks $T_1, \ldots, T_n$ are to be scheduled. Task T_i may be scheduled on any number q of processors such that $q \leq \delta_i$, its running time on q processors is given by V_i/q and preemption is allowed without any cost, and the goal is to minimize $\sum w_i C_i$, where w_i denotes the weight of task T_i and C_i denotes its completion time.

In turn, this problem comes into two flavors. Indeed, the work V_i associated to task T_i may either be known in advance or not. In this paper, we consider both the clairvoyant and the non-clairvoyant settings.

Although this paper is written in the perspective of parallel task scheduling, we would like to underline that the use of quality of service mechanisms [5], [6], [7] for TCP bandwidth sharing makes that results presented in this paper can be adapted to the simultaneous transfer of files in large scale distributed networks [8]. For instance, let us consider the case of one server with outgoing bandwidth P that distributes codes of size V_i

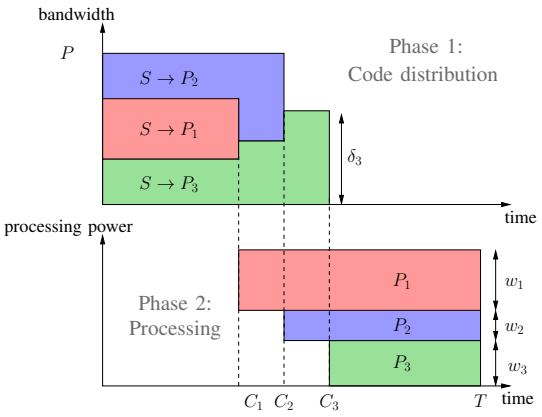

Figure 1. Relationship between weighted mean completion time and bandwidth sharing when considering the work done before T.

to processors $P_1, \ldots, P_n$ that will in turn be responsible for processing tasks at rate w_i and whose incoming bandwidth is δ_i, as illustrated on Figure 1. Then, the natural objective is to maximize the overall number of tasks processed within T time units, what corresponds to maximizing $\sum_i w_i(T - C_i)$, since processor P_i can start processing tasks at rate w_i as soon as it has received the code. In turn, maximizing $\sum_i w_i(T - C_i)$ is equivalent to minimizing $\sum_i w_i C_i$, *i.e.* the weighted mean completion time. Therefore, there is a close relationship between maximizing the number of tasks in a distributed heterogeneous master-workers platform using bandwidth sharing and minimizing the weighted sum of malleable tasks completion times.

Related Works and Contributions:

Let us denote by

- P the total number of processors,
- $T_1, \ldots, T_n$ the set of tasks,
- δ_i the maximal number of processors for task T_i,
- V_i the total volume for task T_i,
- w_i the weight for task T_i,
- and C_i the completion time for task T_i

The problem is generally defined using integer values for the number of processors allocated to each tasks. However, we can transform any fractional schedule which is constant by intervals in an integer schedule with the same completion times, as we will prove it in Theorem 3. Thus, in the following, we focus on the fractional case.

In the non-clairvoyant setting, we propose in Section III an approximation algorithm whose worst case approximation ratio is 2, *i.e.* the same as the best approximation ratios known in the non-clairvoyant cases when either the weights w_is are supposed to be homogeneous or when uniprocessor tasks only are considered ($\delta_i = 1$).

Using the Graham notation extended for parallel tasks by Drowdzoski [2], the problem we consider is $P|pmtn; var; V_i/q, \delta_i| \sum w_i C_i$, that is NP-complete as a generalization of $P|pmtn| \sum w_i C_i$ [9]. In the clairvoyant case, most of scheduling problems for independent malleable work-preserving tasks are polynomially solvable when the goal is to minimize the makespan. For instance, $P|var; V_i/q, \delta_i, r_i|C_{\max}$ (even in presence of release dates r_is) can be solved in time $O(n^2)$ [10]. The maximum lateness problem $P|var; V_i/q, \delta_i, r_i|L_{\max}$ is also solvable in time $O(n^4 P)$ [2]. It is worth noting that the algorithm we propose in Section IV enables to solve this problem in $O(n \log n)$ time if all release dates are equal to zero. On the other hand, problems related to the weighted sum of completion times are NP-Complete [11]. The main contribution of this paper is to propose in Section IV a normal form for malleable task scheduling. More precisely, we prove in Section IV that we can transform any valid schedule into a normal form schedule, that preserves the completion times of all tasks. Therefore, it can be used to reduce the search space [11] when solving any optimization problem involving malleable work-preserving tasks, provided that the objective function only involves task completion times. We show that a consequence of this normal form is that the overall number of preemptions for n tasks in any solution under the normal form is bounded by $3n$. At last, we propose a conjecture that would dramatically reduce the search space for a number of NP-complete problems on malleable tasks, for objectives like the maximum tardiness or the weighted sum of completion times, and we prove the conjecture for a significant class of instances.

The comparison between existing results and those presented in this paper is presented in Table I), where "=" means that the parameters corresponding to the different tasks are the same (homogeneous case) and where "$\neq$" means that the parameters corresponding to the different tasks are different (heterogeneous case), and the context is either "C" (Clairvoyant) or "N-C" (Non-Clairvoyant). For the parameter δ_i, the notation $= 1$ means that only uniprocessor tasks are considered (but multiple processors are available), and $= P$ means that this parameter is ignored, which is equivalent to scheduling on only one processor.

The rest of the paper is organized as follows. In Section II, we describe the model more precisely, and expose an equivalence between integer and fractional schedules. In Section III, we propose a non-clairvoyant 2-approximation algorithm for our problem. In Sec-

δ_i	V_i	Objective	Context	Results
$\neq$	$\neq$	$\sum w_i C_i$	N-C	2-approx (**Section III**)
$=1$	$\neq$	$\sum C_i$	N-C	2-approx [12]
$\neq$	$\neq$	$\sum C_i$	N-C	2-approx [13]
$=P$	$\neq$	$\sum w_i C_i$	N-C	2-approx [14]
$\neq$	=	$\sum C_i$	C	Open (**Section V-B**)
$=P$	$\neq$	$\sum w_i C_i$	C	polynomial [15]
$=1$	$\neq$	$\sum C_i$	C	polynomial [16]
$\neq$	$\neq$	$C_{\max}$	C	$O(n^2)$ [10]
$\neq$	$\neq$	$L_{\max}$	C	$O(n^4 P)$ [2]
$=1$	$\neq$	$\sum w_i C_i$	C	$\frac{1+\sqrt{2}}{2}$-approx[17], [18]

Table I
COMPLEXITY RESULTS FOR DIFFERENT MODELS.

tion IV, we provide an algorithm that computes a normal form for any valid schedule, and we prove a bound on the number of preemptions the schedules produced. In Section V, we define greedy schedules and analyze their performance. In particular, we prove that all optimal schedules are greedy for a certain class of unweighted instances, and we conjecture that at least one optimal schedule is greedy in the general case. Finally, concluding remarks are given in Section VI.

II. MODEL

In this paper, we concentrate on the clairvoyant and non-clairvoyant versions of the following scheduling problem.

Definition 1 (MinWeightedCompletionTimeMalleable (**MWCT**)). *Given an instance $I = (P, (w_i)_{i\leq n}, (V_i)_{i\leq n}, (\delta_i)_{i\leq n})$, find a resource allocation function $d_i(t)$, where $d_i(t)$ denotes the integer number of processors allocated to task T_i at time t (and satisfies $\forall t,\ d_i(t) \leq \delta_i, \quad \forall t,\ \sum_i d_i(t) \leq P, \int_0^\infty d_i(t)dt = V_i$) that minimizes $\sum_i w_i C_i$, where $C_i = \max\{t, d_i(t) \neq 0\}$ denotes the completion time of task T_i.*

This formulation is very general, since there is no assumption about the functions $d_i(t)$. However, between the completion times of two tasks, the exact allocation of resource is not very important: only the total amount allocated to each task defines the schedule. It is thus natural to give a more "compact" description of a schedule by allocating to each task a constant number of processors in each time interval $]C_{\pi(j-1)}, C_{\pi(j)}]$, where π is the order the tasks such that $C_{\pi(1)} \leq C_{\pi(2)} \leq \cdots \leq C_{\pi(n)}$. This time interval is called *column j* in the following. However, since in the general case this constant number is not necessarily an integer, we thus introduce the fractional column-based version of this problem:

Definition 2 (MinWeightedCompletionTimeMalleableColumnBasedFractional (**MWCT-CB-F**)). *Given an instance $I = (P, (w_i)_{i\leq n}, (V_i)_{i\leq n}, (\delta_i)_{i\leq n})$, find an order π and a resource allocation $d_{i,j}$, where $d_{i,j}$ denotes the fractional number of processors allocated to task T_i in column j (and satisfies $\forall j, C_{\pi(j)} \leq C_{\pi(j+1)}, \forall i,j\ d_{i,j} \leq \delta_i, \quad \forall j,\ \sum_i d_{i,j} \leq P, \sum_{j=1}^{n} (C_{\pi(j)} - C_{\pi(j-1)}) d_{i,j} = V_i$) that minimizes $\sum_i w_i C_i$.*

The following theorem shows that both formulations are in fact equivalent.

Theorem 3. *For any instance $I = (P, (w_i)_{i\leq n}, (V_i)_{i\leq n}, (\delta_i)_{i\leq n})$, any valid schedule for* **MWCT** *can be transformed into a valid schedule for* **MWCT-CB-F** *with the same completion times, and vice-versa.*

Proof: Assume without loss of generality, that π is the identity: $\forall i, \pi(i) = i$. Let $d_{i,j}$ be a valid allocation for **MWCT-CB-F**. We construct a solution of **MWCT** which allocates an amount $d_{i,j} \times (C_j - C_{j-1})$ of resource to task T_i in column j, as illustrated in Figure 2. We start with the first processor available or partially available (the one with the lowest index), and we allocate it (or its remaining part) to T_i. We continue with processors with higher indices until $d_{i,j} \times (C_j - C_{j-1})$ is reached (the last processor may be partially allocated to T_i, in this case the earliest part is dedicated to T_i).

Clearly, task T_i is allocated to a set of processors $P_k, \ldots, P_l$, so that at most P_k and P_l may not be fully allocated to T_i, and such that the number of resources $d_i(t)$ allocated to T_i at any time step t is either $\lfloor d_{i,j} \rfloor$ or $\lceil d_{i,j} \rceil$. Moreover, $d_{i,j} \leq \delta_i$ because the original allocation is valid, and since δ_i is an integer, it follows that $\lceil d_{i,j} \rceil \leq \delta_i$. Therefore, $d_i(t)$ is a valid allocation for **MWCT** in which the tasks have the same completion times.

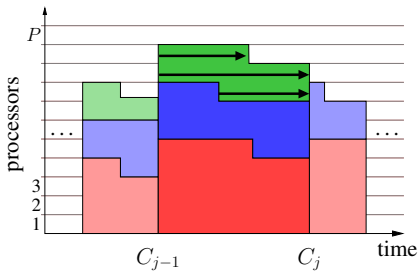

Figure 2. Transforming a fractional solution into an integer solution.

Conversely, let $d_i(t)$ be a valid allocation for **MWCT**. We build an allocation $d_{i,j}$ for **MWCT-CB-F** by allo-

cating to each task i in column j its average allocation in this column

$$d_{i,j} = \frac{\int_{C_{j-1}}^{C_j} d_i(t)dt}{C_j - C_{j-1}}.$$

Then, since $\forall t, \quad d_i(t) \leq \delta_i$, we have that $d_{i,j} \leq \delta_i$ and

$$\sum_i d_{i,j} = \frac{\int_{C_{j-1}}^{C_j} \sum_i d_i(t)dt}{C_j - C_{j-1}} \leq P.$$

Therefore, $d_{i,j}$ is a valid allocation for **MWCT-CB-F** in which tasks have the same completion times. ■

Thanks to this equivalence, the remainder of the paper is mostly focused on the "easiest" (fractional and column-based) version of the problem (**MWCT-CB-F**). Nevertheless, this theorem proves that it is always possible to transform a valid schedule for **MWCT-CB-F** into a valid schedule for **MWCT** (and that in the resulting schedule, the set of processors allocated to T_i in column j changes at most twice). The following result is a direct consequence of this theorem.

Corollary 1. *Consider instance* $I = (P, (w_i)_{i\leq n}, (V_i)_{i\leq n}, (\delta_i)_{i\leq n})$*, and let us assume that the ordering of the completion times* C_i *in a given optimal solution of* **MWCT** *is known. Without loss of generality, we assume that* $C_1 \leq C_2 \leq \cdots \leq C_n$*. Then, optimal solutions for* **MWCT** *and* **MWCT-CB-F** *can be computed in polynomial time.*

Proof: Let us consider the **MWCT-CB-F** version of the problem. The following linear program in which C_i and $x_{i,j}$ are rational variables provides the optimal solution.

$$\text{Minimize} \sum_j w_i C_i, \text{ subject to}$$

$$\begin{cases} C_i \geq C_{i-1} \geq 0, & \forall 1 \leq i \leq n, \\ \sum_{i=1}^n x_{i,j} \leq P(C_j - C_{j-1}) & \forall 1 \leq j \leq n, \\ x_{i,j} \leq \delta_i(C_j - C_{j-1}), & \forall 1 \leq i,j \leq n \\ \sum_{j=1}^n x_{i,j} = V_i, & \forall 1 \leq i \leq n \\ \sum_{j=i+1}^n x_{i,j} = 0, & \forall 1 \leq i \leq n \end{cases}.$$

■

III. Non-clairvoyant approximation algorithm

In this section, we introduce **WDEQ** (Weighted Dynamic EQuipartition), a non-clairvoyant online algorithm to solve our problem. DEQ Algorithm [13] has already been proposed for the unweighted problem ($\forall i, w_i = 1$), and **WDEQ** is an extension to the weighted case.

The idea behind Algorithm **WDEQ** is to perform a *fair* sharing of the platform between all available tasks, in proportion of their weights. Tasks whose share would exceed δ_i are allocated exactly δ_i, and the extra resources are allocated similarly to the remaining tasks. Since this is a dynamic algorithm, the sharing is recomputed each time a task completes.

Algorithm 1: WDEQ algorithm

while $\exists i, \delta_i < \frac{w_i P}{\sum w_i}$ **do**
  Allocate δ_i to i
  Update $P \leftarrow P - \delta_i$ and $w_i \leftarrow 0$
Allocate $\frac{w_i P}{\sum w_i}$ to all remaining tasks i

DEQ is a 2-approximation algorithm for the sum of completion times [13]. Independently, Weighted Round Robin is also a 2-approximation algorithm for the weighted case on a single processor [14]. In this section, we prove that **WDEQ** remains a 2-approximation for the weighted case and for malleable tasks. We start with some lemmas to bound the objective value of an optimal schedule.

Theorem 4. WDEQ *is a 2-approximation algorithm for* **MWCT-CB-F**.

We start with some definitions and lemmas to bound the objective value of an optimal schedule.

Definition 5 (Squashed area bound)**.** *For any instance* I*, sorted such that* $\frac{V_1}{w_1} \leq \frac{V_2}{w_2} \leq \frac{V_n}{w_n}$*, let us set* $A(I) = \sum_i \left(\sum_{j\geq i} w_j\right) \frac{V_i}{P}$.

Definition 6 (Height bound)**.** *For any instance* I*, let us set* $H(I) = \sum_i w_i h_i$*, where* $h_i = \frac{V_i}{\delta_i}$ *is the* height *of task* T_i.

It is a well-known result that $A(I)$ and $H(I)$ are lower bounds of the optimal value $OPT(I)$. Indeed, $A(I)$ is the value of an optimal schedule for the case where $\delta_i = P$ for all tasks i, which is the same problem as uniprocessor scheduling. This schedule is based on Smith's rule [15], but does not take into account the bound δ_i on the number of processors for T_i. Similarly, $H(I)$ is the value of an optimal schedule for the case $P = \infty$. The next lemma shows (as in [13]) that it is possible to mix those two bounds on different parts of the instance.

Definition 7 (Subinstances)**.** *For any instance* I*, and any values* $V_i' \leq V_i$*, we call subinstance and denote by*

$I[V_i']$ the instance similar to I except that task T_i has volume V_i'.

Lemma 1 (Mixed lower bound)**.** *For any instance I, and any subdivision of I in $I[V_i^1]$ and $I[V_i^2]$ such that $V_i^1 + V_i^2 = V_i$, $OPT(I) \geq A(I[V_i^1]) + H(I[V_i^2])$.*

Proof: Let us consider an optimal schedule for I, and define, for all tasks i, t_i^1 as the time instant at which task T_i has processed volume V_i^1. It is clear that $C_i \geq t_i^1 + \frac{V_i^2}{\delta_i}$. Hence, $OPT(I) \geq \sum_i w_i t_i^1 + \sum_i w_i \frac{V_i^2}{\delta_i}$. Since the t_i^1s are a valid schedule for instance $I[V_i^1]$, it follows that

$$OPT(I) \geq OPT(I[V_i^1]) + H(I[V_i^2]), \quad (1)$$
$$\geq A(I[V_i^1]) + H(I[V_i^2]). \quad (2)$$

■

In order to prove the approximation result, let us analyze the schedule produced by **WDEQ** with the remark that the amount of resources allocated to each task is increasing with time, until it is given its full allocation (δ_i). We denote by VF_i^I the volume processed by task T_i in instance I while running with full allocation, and by $\overline{\mathrm{VF}}_i^I$ the volume processed when the allocation is limited by **WDEQ**. Theorem 4 is a direct consequence of the following lemma.

Lemma 2. *Let us denote by $TC_{WD}(I)$ the weighted mean completion time of the schedule produced by* **WDEQ** *on an instance I. For any instance I,*

$$TC_{WD}(I) \leq 2(A(I[VF_i^I]) + H(I[\overline{VF}_i^I]))$$

Proof: The proof is by induction on n, the number of tasks in instance I. Clearly if $n = 1$ the result holds.

Assume that for a given n, the result holds for all instances of size $< n$, and let I be an instance of size n. Let us denote by τ the time at which the first task terminates in the schedule produced by **WDEQ** on instance I. We can decompose the set of tasks in two sets: the set F of the tasks whose allocation $d_{i,1}$ between time 0 and τ is δ_i, and the set $\overline{F}$ of the tasks whose allocation is limited by **WDEQ**. According to Algorithm 1, we know that the value $\frac{d_{i,1}}{w_i}$ is constant for all tasks in $\overline{F}$, we denote this value by c. We also set $W = \sum_i w_i$.

The schedule produced by **WDEQ** after time τ is exactly the same as the one produced on instance $I' = I[V_i - d_{i,1}\tau]$ in which the volume of all tasks is decreased by the amount processed between time 0 and τ. Hence $\mathrm{TC}_{WD}(I) = W\tau + \mathrm{TC}_{WD}(I')$ and since one task ends up at time τ, instance I' contains less than n tasks, and we can apply the induction hypothesis,

$$\mathrm{TC}_{WD}(I') \leq 2 \times \left(A(I'[\overline{\mathrm{VF}}_i^{I'}]) + H(I'[\mathrm{VF}_i^{I'}]) \right). \quad (3)$$

Let us compute more precisely the values on the right hand side,

$$\overline{\mathrm{VF}}_i^{I'} = \begin{cases} \overline{\mathrm{VF}}_i^I - w_i c\tau & \text{if } i \in \overline{F} \\ \overline{\mathrm{VF}}_i^I = 0 & \text{if } i \in F \end{cases},$$
$$\mathrm{VF}_i^{I'} = \begin{cases} \mathrm{VF}_i^I & \text{if } i \in \overline{F} \\ \mathrm{VF}_i^I - \delta_i \tau & \text{if } i \in F \end{cases}.$$

Remark also that tasks are considered in the same order in $A(I[\overline{\mathrm{VF}}_i^I])$ and in $A(I'[\overline{\mathrm{VF}}_i^{I'}])$. Indeed, the ratios $\frac{V_i}{w_i}$, when non-zero, differ only by a constant amount ($c\tau$) between these two instances. This implies that

$$A(I'[\overline{\mathrm{VF}}_i^{I'}]) = A(I[\overline{\mathrm{VF}}_i^I]) - \sum_{i \in \overline{F}} \left(\sum_{j \geq i} w_j \right) \frac{w_i c\tau}{P},$$
$$H(I'[\mathrm{VF}_i^{I'}]) = H(I[\mathrm{VF}_i^I]) - \sum_{i \in F} w_i \tau.$$

Combining these together with (3) yields to

$$\mathrm{TC}_{WD}(I) - 2 \times \left(A(I[\overline{\mathrm{VF}}_i^I]) + H(I[\mathrm{VF}_i^I]) \right)$$
$$\leq W\tau - 2 \left(\sum_{i,j \in \overline{F}, j \geq i} w_j \frac{w_i c\tau}{P} + \sum_{i \in F} w_i \tau \right). \quad (4)$$

From Algorithm 1 we know that $c \geq \frac{P}{W}$. Hence, it remains to prove that

$$W \leq 2 \left(\frac{1}{W} \sum_{i,j \in \overline{F}, j \geq i} w_j w_i + \sum_{i \in F} w_i \right).$$

If we denote by $W_F = \sum_{i \in F} w_i$ and $W_{\overline{F}} = \sum_{i \in \overline{F}} w_i$, we get

$$W^2 = W_F^2 + 2W_F W_{\overline{F}} + W_{\overline{F}}^2$$
$$= W_F(W_F + 2W_{\overline{F}}) + W_{\overline{F}}^2$$
$$\leq 2WW_F + 2 \sum_{i,j \in \overline{F}, i \leq j} w_i w_j.$$

This, together with equation (4), concludes the proof of this lemma. ■

IV. A NORMAL FORM AND ITS APPLICATIONS

In this section, we present a normalization process for schedules of work-preserving malleable tasks. Given a valid schedule $\mathcal{S}$ for such malleable tasks, we build a normalized schedule based only on the completion times of the tasks in $\mathcal{S}$. To do this, we introduce algorithm **WF** (for "Water-Filling" algorithm), described in Algorithm 2. After proving its correctness, we exhibit a few properties of the normalized schedule.

In the following, we assume that tasks are sorted in non-decreasing order of completion time $C_1 \leq C_2 \leq \cdots \leq C_n$. As presented above, we consider that the schedule is organized in *columns*. Each column corresponds to a time slice of the schedule between two task completions. We define column k as the time interval $[C_{k-1}, C_k]$ between the completion of tasks T_{k-1} and T_k (or between time $C_0 = 0$ and C_1 when $k = 1$). We denote by $l_k = C_k - C_{k-1}$ the duration, of column k. In each column, we consider that the rational number of processors allocated to each task is constant (see Theorem 3 in Section II).

A. Algorithm *WF*

Algorithm **WF** proceeds by allocating resources to task T_1 (within time bound C_1), then to task T_2 (within time bound C_1), etc. until all tasks have been scheduled.

After scheduling tasks $T_1, \ldots, T_i$, let h_k^i denote the *height* of allocated resources in column k (the area already allocated in column k is thus $h_k^i \times l_k$). Let us also define $wf_i(h)$ as the maximal amount of resources that can be allocated to task T_i with the constraint that the resulting height of columns where T_i is given some resources does not exceed h, in addition to the bound δ_i on the number of processors allocated to T_i. This function is illustrated in Figure 3 and formally defined as follows

$$wf_i(h) = \sum_{k=1}^{i} l_k \min(h - h_k^{i-1}, \delta_i, 0).$$

This function is used of algorithm **WF**, described in Algorithm 2. The rationale behind this algorithm is as follows: once Tasks $T_1, \ldots, T_{i-1}$ have been allocated, what may prevent Task T_i from being allocated is insufficient resources in columns $1, \ldots, i$ available for T_i. Since the number of processors which can be used by T_i cannot exceed δ_i, we favor *flat* areas of available resources, by balancing the height of allocated columns as much as possible. Let us recall that tasks are considered by increasing completion times C_i. When allocating T_i, the algorithm searches the minimum height h such that $wf_i(h)$ is larger than its volume V_i, as if an amount V_i of water was *poured* into the schedule. However, some columns may reach a height lower than h, because of the additional constraint δ_i on the maximal number of processors allocated to T_i. We say that T_i is *saturated* in these columns.

Algorithm 2: Algorithm **WF** for malleable tasks.

```
for i = 1 ... n do
    if wf_i(P) < V_i then
        return no valid solution
    h^i ← min{h | wf_i(h) = V_i}
    for k = 1 ... i do
        allocate to T_i the volume
        l_k × min(h^i − h_k^{i−1}, δ_i, 0) in column k
return current allocation
```

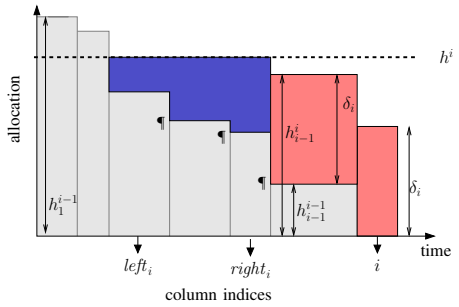

Figure 3. Allocation of task T_i. The last two columns T_i is saturated, whereas in the columns from $left_i$ to $right_i$ is unsaturated.

It is clear that if algorithm **WF** returns an allocation, this allocation is valid. Indeed, i/ the volume allocated to each task is equal to V_i, ii/ the height of each column is at most P and iii/ the volume allocated to task T_i in column j is at most $l_j \times \delta_i$. Moreover, each task T_i completes at time C_i. In the following, we prove the following result.

Theorem 8. *Given an instance I and sequence of completion times (C_i)s, Algorithm **WF** computes a valid allocation with Task T_i completing at time C_i if one exists.*

The proof of this result relies on two lemmas. The first one describes the structure of an allocation produced by Algorithm **WF**.

Lemma 3. *In Algorithm **WF**, after allocated resources to task T_i, the current occupation is a non-increasing function of time $\forall k, h_k^i \geq h_{k+1}^i$.*

Proof: Let us prove this result by induction on i. For $i = 1$ the result holds. Assume now that the result

holds for $i-1$ and let Algorithm **WF** allocate resources to task T_i. Just before allocating resources to task T_i, columns can be divided into 3 groups: columns with height greater than h_i, columns with height between $h_i-\delta_i$ and h_i, and column with height lower than $h_i-\delta_i$. From the induction hypothesis, the columns of the first group precede the columns of the second group, which precede the columns of the third group. After allocating resources to T_i the set of columns in the first group remains unchanged and their height larger than h_i (and in decreasing order), the height of the columns in the second group will be equal to h_i and finally the height of the column of the last group will be smaller than h_i and still in decreasing order. Hence the result also holds for i, what concludes the induction. ■

From the previous lemma we can define $left_i$ and $right_i$ as respectively the first and last columns in which Task T_i is allocated resources but remains unsaturated. If T_i has no unsaturated column, then $left_i = right_i = 0$. Observe that h^i_j is the same for $left_i \leq j \leq right_i$: these contiguous columns are *de facto* merged, since tasks allocated later (those with $C_k > C_i$) will "see" a single column of constant height. This property will be very helpful when proving the correctness of Algorithm **WF** (Theorem 8).

A similar algorithm was previously introduced by Chen et al. in [19] for non-fractional malleable tasks. Actually, applying algorithm **WF** followed by the algorithm presented in the proof of Theorem 3 to transform a fractional schedule into an integer one results in the same schedule than the one produced by the algorithm presented in [19]. However, there are two major improvements in our algorithm. First, Chen's algorithm has a number of steps proportional to the overall allocated area, which may be huge. Second, in the proof of correctness of Chen's algorithm, one important case is missing: when comparing a schedule S and the normalized schedule S', the authors only deal with the case when the first difference in the resources allocated to a task is a block allocated in S' but not in S, but they fail to consider the (more involved) converse case.

The following lemma is crucial in the proof of the theorem, as it states that after allocating a sequence of tasks, the area that remains available for any next task is maximized, whatever its limit δ on maximal simultaneous resource usage.

Lemma 4. *Let $(l_1, l_2, \ldots, l_n)$ denote a sequence of time interval lengths, $T_i = (\delta_i, V_i)_{1\leq i\leq n}$ denote a sequence of tasks and P denote the total number of resources. Let us consider a valid schedule S for the sequence of time intervals (such that Task T_i completes at the end of the i^{th} interval), and let us denote by s^i_k the height in column k occupied by tasks $T_1, \ldots, T_i$. Let us denote by h^i_k the occupied height in column k by tasks $T_1, \ldots, T_i$, if allocated using Algorithm **WF**. Then, we have $\forall \delta \leq P, \forall m < n$,*

$$\sum_{k=1}^{m+1} \min(\delta, P - h^m_k) l_k \geq \sum_{k=1}^{m+1} \min(\delta, P - s^m_k) l_k.$$

Proof: For a given m, let us consider the function $alloc(t)$, that gives the number of processors allocated at time t after scheduling the first m tasks using Algorithm **WF**. Formally, $alloc(t) = h^m_k$ where k is the column where t lies (*i.e.* $\sum_{j=1}^{k-1} l_j < t \leq \sum_{j=1}^{k} l_j$).

Let us now concentrate on the instant t_δ when $alloc(t)$ intersects with the horizontal line of height $P - \delta$, (that corresponds to the end of a column and the completion of a task), as illustrated in Figure 4. If $P - \delta$ exactly corresponds to the height of a column, then we define t_δ as the end of this column.

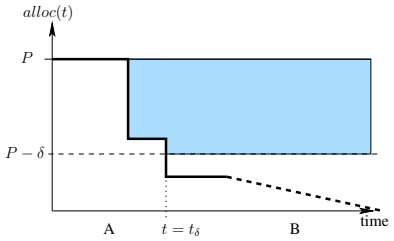

Figure 4. Definition of t_δ in the proof of Lemma 4.

Note that the quantity $\sum_{k=1}^{m+1} \min(\delta, P - h^i_k) l_k$ is equal to the area above both curves (shaded in the figure). We split this sum into two parts, considering first the columns before time t_δ, denoted by $\mathcal{C}_A$, and second the columns after time t_δ, denoted by $\mathcal{C}_B$:

$$\sum_{k=1}^{m+1} \min(\delta, P - h^i_k) l_k = \sum_{k\in\mathcal{C}_A} \min(\delta, P - h^i_k) l_k + \sum_{k\in\mathcal{C}_B} \min(\delta, P - h^i_k) l_k.$$

Let us consider both contributions, starting with the second one.

- In the second sum, for a column $k \in \mathcal{C}_B$, the area is bounded by δ (since $\min(\delta, B - h^i_k) = \delta$), which is a natural upper bound of this quantity. Thus, no other allocation S can have a larger contribution after t_δ.

- In the first sum, for a column $k \in \mathcal{C}_A$, the area is bounded by $alloc(t_\delta) = \min(\delta, P - h_k^i) = P - h_k^i$. Thus,

$$\sum_{k \in \mathcal{C}_A} \min(\delta, P - h_k^i) l_k = P \times \left(\sum_{k \in \mathcal{C}_A} l_k \right) - \left(\sum_{k \in \mathcal{C}_A} l_k h_k^i \right).$$

  We focus on the resources already allocated in columns of set $\mathcal{C}_A$ ($\sum_{k \in \mathcal{C}_A} l_k h_k^i$), and see how this amount could be reduced. There are two cases for tasks allocated in $\mathcal{C}_A$,
  - either their completion time is smaller than or equal to t_δ. In this case, they must be allocated completely before t_δ (in $\mathcal{C}_A$).
  - or their completion time is larger than t_δ. In this case, algorithm **WF** has allocated to these tasks their maximum capacity δ_i in the columns of $\mathcal{C}_B$, and the remaining volume must be allocated in $\mathcal{C}_A$.

  Indeed, there is no third choice, since t_δ corresponds to the beginning of a column lower than the one before, and therefore, there is no task T_j such that $t_\delta \in]C_{left_j - 1}, C_{right_j}[$ (since $h^j_{left_j} = h^j_{right_j}$). In both cases (task completing before or after t_δ), all the resources already allocated in the columns of set $\mathcal{C}_A$ cannot be allocated later than t_δ by any other allocation. Thus, the available resources for a task with limitation δ cannot be improved before t_δ either.

This achieves the proof of Lemma 4. ■

We are now able to prove that Algorithm **WF** always produces a valid allocation if one exists.

Proof of Theorem 8: Let us assume by contradiction that Algorithm **WF** fails to compute a valid solution and let us denote by T_{m+1} the first task that cannot be allocated using Algorithm 2. This implies that $wf_{m+1}(P) < V_{m+1}$:

$$\sum_{k=1}^{m+1} \min(\delta_{m+1}, P - h_k^m) l_k < V_{m+1}.$$

On the other hand, let us consider a valid schedule S for this problem. Using Lemma 4 with $\delta = \delta_{m+1}$ we get

$$\sum_{k=1}^{m+1} \min(\delta_{m+1}, P - h_k^m) l_k \geq \sum_{k=1}^{m+1} \min(\delta_{m+1}, P - s_k^m) l_k.$$

Moreover, for all k, the resources allocated to T_{m+1} in S cannot exceed $\min(\delta_{m+1}, P - s_k^m)$ by construction. Since S is a valid solution for Tasks $T_1, \ldots, T_m, T_{m+1}$, then $\sum_{k=1}^{m+1} \min(\delta_{m+1}, P - s_k^m) l_k \geq V_{m+1}$ and therefore

$$\sum_{k=1}^{m+1} \min(\delta_{m+1}, P - h_k^m) l_k \geq V_{m+1},$$

what contradicts the assumption and achieves the proof of the theorem. ■

B. Bound on the number of preemptions

In this section, we prove that the overall number of preemptions for n malleable tasks scheduled using fractional numbers of processors allocated using Algorithm **WF** is bounded by n. Therefore, since any valid schedule can be turned into a schedule built by algorithm **WF**, the search for optimal malleable schedule can always be restricted to schedules inducing at most one preemption on average, whatever the objective function is.

To obtain this result, we first prove that the number of changes in the number of resources allocated to a task is bounded by one on average. Then, we prove that, given this property, it is indeed possible to assign processors to tasks so that at most one preemption per task takes place on average.

Lemma 5. *The overall number of changes in the quantity of resources allocated to all tasks using algorithm **WF** with fractional number of processors is bounded by the number n of tasks.*

Proof: For a given task, we do not consider that there is a change in the number of allocated resources when the task is scheduled for the first time and for the last time, so that the number of changes is closely related to the number of preemptions.

In what follows, after the allocation of Tasks $T_1, \ldots, T_i$ using algorithm **WF**, we denote by

- N_i the overall number of changes over time in the number of *allocated* resources to Tasks $T_1, \ldots, T_i$,
- M_i the number of changes over time in the number of *available* resources after the allocation of Tasks $T_1, \ldots, T_i$.

We prove that the following inequality holds

$$\forall i \geq 2, \quad N_i + M_i \leq N_{i-1} + M_{i-1} + 1.$$

Let us consider the allocation of Task T_i using Algorithm **WF**. Typically, the allocation consists in two phases, as depicted on Figure 3. During the second phase, that corresponds to saturated columns ($right_i + 1$

to i), the number of resources allocated to T_i is exactly δ_i, so that no change occurs.

During the first phase, that corresponds to unsaturated columns ($left_i$ to $right_i$), the number of resources allocated to T_i changes exactly $right_i - left_i$ times (each change is indicated with a ¶ mark on Figure 3). Simultaneously, these columns are merged, so that the number of changes in the available resources is decreased by $right_i - left_i - 1$. Overall the sum $N_i + M_i$ is increased by one.

Finally, when no task is scheduled, we have $N_0 = M_0 = 0$, so that the previous relation gives $N_n + M_n \leq n$. The observation that $M_n \geq 1$ concludes the proof. ■

The following lemma states that using algorithm **WF**, it is possible to allocate processors to tasks such that the overall number of preemptions is exactly the number of changes in the number of processors allocated to tasks.

Lemma 6. *Let N be the total number of changes in the number of processors allocated to tasks, in a schedule with fractional numbers of processors produced by Algorithm **WF**. Then, there is an allocation of tasks to processor with no more than N preemptions.*

Proof: We note that in the schedule produced by Algorithm **WF**, the number of resources allocated to one task never decreases (except when the task is completed). Thus, it is always possible to allocate resources such that a resource allocated to a task will not be reclaimed until the end of the task. ■

When combining Lemma 5 with Lemma 6, we obtain the following result on the overall number of preemptions.

Theorem 9. *The overall number of preemptions induced by algorithm **WF** using fractional numbers of processors is bounded by the number of tasks.*

More interestingly, this result can be adapted to the case with integer numbers of processors. When a fractional schedule is transformed into a schedule with integer numbers of processors as described in the proof of Theorem 3, at most a small step can be introduced in each column, at a (possibly) different time step for each processor. Overall, this may result in a much larger number of preemptions. However, the previous results can be adapted as follows.

Theorem 10. *The overall number of preemptions induced by algorithm **WF** with **integer** numbers of processors is bounded by $3n$, where n denotes the number of tasks.*

The proof of this result is provided in Appendix A, and is very similar to the one presented for the fractional case. In the integer case, we are able to prove a slightly weaker relation on the N_is and M_is: $N_{i+1} + M_{i+1} \leq N_i + M_i + 3$.

V. Dominance of greedy schedules

One natural way to build a schedule is greedily: given an ordering σ on the tasks, select the first task, allocate to it as much resource as possible, as soon as possible, and so on until every task is scheduled (see Algorithm 3). A schedule is said to be *greedy* if it can be produced greedily for some order. In this section, we first show that for a certain class of instances, all optimal schedules are greedy. Then, we conjecture that this last result is correct for any instance. Finally, we investigate some properties of greedy schedules on a class of instances that seems to concentrate the complexity of the problem, despite their apparent simplicity.

Algorithm 3: Algorithm Greedy(σ)

for $i = 1 \dots n$ **do**
  Allocate resources to Task $T_{\sigma(i)}$ in order to minimize completion time of $T_{\sigma(i)}$
  Update available resources

A. Greedy schedules on instances with homogeneous weight and large capacities.

In this section, we concentrate on the fractional and column-based version of the problem (**MWCT-CB-F**). Thus, all considered schedules will have a constant number of processors allocated to a task between two tasks completions. In the following proofs, for the sake of clarity, we may transform the schedule into a schedule that may not satisfy these constraints. However, such a schedule can always be normalized into a correct schedule for **MWCT-CB-F** by computing the average number of processors used in a column, as described in Section II. However, since Algorithm Greedy naturally produces schedules with integer number of allocated processors, they are also solution of **MWCT**.

Theorem 11. *Let I be an instance of* **MWCT-CB-F** *with homogeneous weight and such that $\forall i, \delta_i > P/2$. Any optimal schedule of I is greedy.*

In order to prove this theorem, we will first prove the following lemmas. We recall that a task T_i is *saturated* in column k if it is at its maximal amount of processing resources in this column, *i.e.* $d_{i,k} = \delta_i(C_k - C_{k-1})$.

Lemma 7. *Let I be an instance with homogeneous weights and such that $\forall i, \delta_i > P/2$. In every optimal*

schedule for **MWCT-CB-F** *on I, each task is saturated in its last column.*

Proof: Let us consider an optimal schedule in which one task is unsaturated in its last column and let us focus on this task T_i. We consider the following two cases:

Case 1: T_i is alone in its last column. It this column, task T_i can increase its resource consumption and so decrease its completion time without changing the completion time of any other task. This transformation leads to a better schedule, and hence the initial schedule is not optimal.

Case 2: There is another task T_j in the last column of task T_i. Let us consider the following transformation. In the last column of task T_i, decrease the completion time of T_i by ϵ by removing resource from T_j. In the former last column of T_i and after the new completion time of T_i, T_j takes as much resource as possible (cf. Figure 5). Thus, its completion time is reduced by ϵ. The amount of resource lost by T_j in this transformation is at most $\epsilon.(P-\delta_j)$ (dashed rectangle A in Figure 5)). To

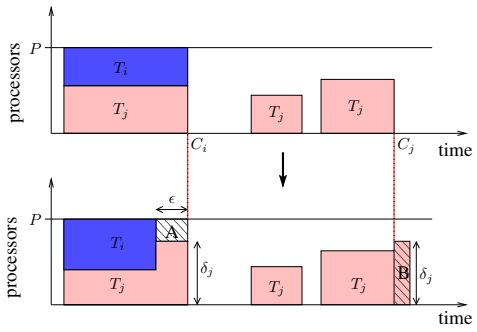

Figure 5. Illustration of the transformation proposed in the second case of Lemma 7

balance this loss, T_j takes it back after its completion time at a rate of δ_j. During this transformation, the completion time of task T_i is decreased by ϵ while the completion time of task T_j is increased by at most $\epsilon.(P-\delta_j)/\delta_j$. Thus, the sum of both completion times is changed by $\epsilon \times (P-2\delta_j)/\delta_j$. Since $\delta_j > P/2$, above transformation leads to a smaller sum of completion time for tasks T_i and T_j.

Let us now focus on the other tasks. Observe that some resources have been allocated to T_j (dashed rectangle B on Figure 5) and the same amount of resource has been released (dashed rectangle A). To complete this transformation, tasks that have been *removed* some resources in B now use the resources released by T_i and T_j in A. Since the height of released resources is less than $P-\delta_j < P/2$ and that for each Task T_k, $\delta_k > P/2$ this transformation leads to a valid schedule. Moreover, the resources released by T_i and T_j are "before" the resources stolen by T_i and T_j. Thus, the completion time of any task T_k with $k \neq i, j$ does not increase with this transformation. Hence, the sum of completion times decreases strictly during this transformation, *i.e.* the original schedule is not optimal.

■

Lemma 8. *Let I be an instance with homogeneous weights and such that $\forall i, \delta_i > P/2$. In an optimal schedule for* **MWCT-CB-F** *on I, if resources are allocated to Task T_i in column k, but T_i does not end up in this column, then T_i is saturated in column $k+1$.*

Proof: By contradiction, let us consider an optimal schedule and assume that some resources are allocated to Task T_i in column k, but T_i does not end up in this column and is unsaturated in column $k+1$. From Lemma 7, Task T_i ends up in a column $k' > k+1$. We consider Tasks T_k and T_{k+1} that ends up in columns k and $k+1$. We first observe that since T_k completes in column k, it is saturated in this column (cf. Lemma 7), and uses $\delta_k > P/2$ processors. Thus, no other tasks can be saturated in this column.

As in the proof of Lemma 7, we consider a transformation in which a task steals some resource. Precisely, Task T_{k+1} steals some resources to Task T_i in column k so that the completion time of T_{k+1} is decreased by ϵ. Task T_i then gets some of the resource released by Task T_{k+1} in column $k+1$. Due to its limitation, task T_i gets $\epsilon \times \min(\delta_i, \delta_{k+1})$ (recall that from Lemma 7, T_{k+1} is saturated in column $k+1$). If this transfer is unbalanced, what is the case if $\delta_{k+1} > \delta_i$, then the length of column $k+1$ is increased by $\epsilon \times (\delta_{k+1} - \min(\delta_i, \delta_{k+1}))/\delta_i$ (see Figure 6). The sum of the completion times of T_i and T_{k+1} is changed by $\epsilon - \epsilon \times (\delta_{k+1} - \min(\delta_i, \delta_{k+1}))/\delta_i$. Since $\delta_i > P/2$ and $\delta_{k+1} < P$, this contribution to the mean-flow is negative. For tasks T_j, with $j \neq i, k+1$ we apply the same transformation as in the end of the proof of Lemma 7. The complete transformation leads to a new schedule with a smaller sum of completion times, and hence proves that the original schedule is not optimal. ■

We are now able to prove that greedy schedules are optimal on such instances.

Proof of Theorem 11: Let us consider an optimal schedule for **MWCT-CB-F**. Let us first assume that there exists a column with two unsaturated tasks. From Lemma 8, both tasks end in the next column. Moreover Lemma 7 states that these two tasks are saturated in this column. This is in contradiction with the fact that $\delta_i > P/2$. Hence, in an optimal schedule, there is at

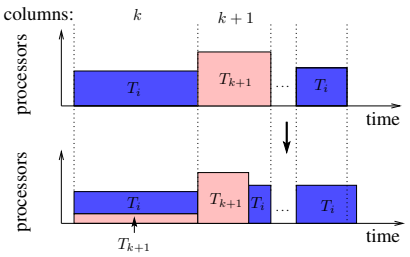

Figure 6. Illustration of the transformation proposed in Lemma 8

most one unsaturated task per column and each task appears in at most two (consecutive) columns.

Let us consider an optimal schedule $\mathcal{O}$. Let us assume without loss of generality that $C_1 \leq C_2 \leq \cdots \leq C_n$. Let σ be the order of completion time ($\forall i, \sigma(i) = i$), and let $\mathcal{G}$ be the greedy schedule given by order σ.

Let us show, by induction on the number k of tasks scheduled by Greedy Algorithm, that the sub-schedules of $\mathcal{O}$ and $\mathcal{G}$ induced by the k first tasks match, in particular the length of column k is the same in both schedules.

For $k = 1$, the property holds since by Lemma 7, Task T_1 is saturated in the first column. The length of the first column is thus V_1/δ_1.

Assume that the property holds for a given value k. In column k of $\mathcal{O}$ (that is the same as column k of $\mathcal{G}$), T_k is saturated and the only other task present in this column is Task T_{k+1}. The volume processed by Task T_{k+1} in column k is the same in $\mathcal{G}$ and in $\mathcal{O}$, *i.e.* $(P - \delta_k) \times (C_k - C_{k-1})$. Hence, the remaining volume to be processed by Task T_{k+1} after column k is also the same in both schedules. Moreover, after time C_k, task T_{k+1} is saturated until its completion time, that is equal in both schedules to $C_k + (V_{k+1} - (P - \delta_k) \times (C_k - C_{k-1}))/\delta_{k+1}$. This implies that the sub-schedules of $\mathcal{O}$ and $\mathcal{G}$ induced by the $k + 1$ first tasks match.

In particular, for $k = n$ we get that $\mathcal{O} = \mathcal{G}$. ■

We have proved in Theorem 11 that for a certain class of instances, we can reduce the search space to greedy schedules. Moreover, we have performed experimentations on random instances. We have considered instances composed of 2, 3, 4 and 5 uniform random tasks (uniform among tasks such that $\delta_i < P, w_i < 1$ and $V_i < 1$). For each set size, we generated 10,000 instances and for each instance the best greedy schedule was numerically indistinguishable from the optimal. We have also successfully performed the same experiments on constant weight instances and on constant weight and constant volume instances. This leads us to formulate the following conjecture.

Conjecture 12. *Given an instance I, there exists a task ordering such that the resulting greedy schedule on I is optimal for* **MWCT-CB-F**.

One can find the code of these experiments, as well as a Java applet to test simple scenarios at the following address http://www.labri.fr/~bonichon/malleable/.

B. What is the complexity of MWCT-CB-F *on instances with homogeneous volumes and weights?*

As mentionned earlier, **MWCT-CB-F** is NP-complete in general. A natural question is the following one. Does **MWCT-CB-F** remains NP-complete when restricted to instances with homogeneous volumes and weights ?

To tackle this question, we have investigated an even more restricted class of instances, *i.e.* instances such that $\forall i, \delta_i \geq 1/2, w_i = 1$, $V_i = 1$ and $P = 1$.

For these instances, we know from Theorem 11 that the problem is equivalent to finding the best ordering σ for Algorithm Greedy. Considered instances are interesting, since greedy schedules are easy to describe. The first task is saturated in the first column and the second one is allocated remaining resources. In column i, task $T_\sigma(i)$ is saturated, and task $T_{\sigma(i+1)}$ is allocated remaining resources. Hence, the sum of completion times is given by the following recurrence formula, $C_{\sigma(1)} = 1/\delta_{\sigma(1)}$, and for $\forall i > 1, C_{\sigma(i)} = C_{\sigma(i-1)} + \frac{1-(1-\delta_{\sigma(i-1)})\times(C_{\sigma(i-1)}-C_{\sigma(i-2)})}{\delta_{\sigma(i)}}$, where $C_{\sigma(0)} = 0$.

These instances are apparently quite simple. Indeed, up to 4 tasks, the optimal orders are easy to describe. However, for a larger number of tasks, the situation is different. Assuming now w.l.o.g. that tasks are such that $\delta_1 \geq \delta_2 \geq \cdots \geq \delta_n$:

- 2 tasks: $1, 2$ and $2, 1$ are both optimal.
- 3 tasks: $1, 3, 2$ and $2, 3, 1$ are both optimal (the smallest in the middle).
- 4 tasks: $1, 3, 2, 4$ and $4, 2, 3, 1$ are both optimal.
- 5 tasks: the situation becomes trickier, and optimal orderings are more complicated to describe. The optimal orderings are determined by the values of δ_i, and not only by their relative order. For instance, among other conditions, if i, j, k, l, m is an optimal order, then $(\delta_l - \delta_j) \times (\delta_i - \delta_m) \leq 0$.

Our (experimental) work on these instances leads us to state the following conjecture.

Conjecture 13. *Let us consider an instance I of* **MWCT-CB-F** *that has homogeneous volume and weight and such that $\forall i, \delta_i \geq P/2$. The weighted sum of completion times of the greedy schedule for a given*

order is equal to the weighted completion time of the greedy schedule in the reversed order.

This conjecture has been formally checked for instances up to 15 tasks using Sage mathematical software (http://www.sagemath.org/).

VI. Conclusion

In this paper, we have studied the scheduling of work-preserving malleable tasks. In the non-clairvoyant case, we have proposed a 2-approximation algorithm, which extend the results of [13] to the weighted case. In the clairvoyant case, we have established several results. First, we have proposed a normal form with the help of an algorithm to reconstruct a valid schedule based only on the completion times of the tasks. In addition to providing a compact representation of any schedule, it is useful to bound the number of preemptions in an optimal schedule. Moreover, we have studied *greedy* schedules, by proving their optimality in a simple case ($\delta_i > P/2$ and $w_i = 1$ for all tasks). Based on our extensive simulations, we conjecture that there exists an optimal *greedy* schedule in all cases, even if this question remains open.

A few interesting questions remain open. It is natural to study the approximation ratio of the greedy schedule based on Smith's ordering (increasing w_i/V_i). As a particular case, bounding the worst-case performance of greedy schedules when $w_i = V_i = 1$ would be interesting. Furthermore, this particular case is the simplest problem whose complexity status remains open.

References

[1] D. Feitelson, "Scheduling parallel jobs on clusters," *High Performance Cluster Computing*, vol. 1, pp. 519–533.

[2] M. Drozdowski, "Scheduling Parallel Tasks–Algorithms and Complexity, chapter 25. Handbook of SCHEDULING Algorithms, Models and Performance Analysis," 2004.

[3] P. Dutot, G. Mounié, and D. Trystram, "Handbook of Scheduling, chapter Scheduling Parallel Tasks-Approximation Algorithms," 2004.

[4] J. Blazewicz, M. Kovalyov, M. Machowiak, D. Trystram, and J. Weglarz, "Preemptable malleable task scheduling problem," *IEEE Transactions on Computers*, pp. 486–490, 2006.

[5] A. B. Downey, "Tcp self-clocking and bandwidth sharing," *Computer Networks*, vol. 51, no. 13, pp. 3844 – 3863, 2007.

[6] D. Abendroth, H. van den Berg, and M. Mandjes, "A versatile model for tcp bandwidth sharing in networks with heterogeneous users," *AEU - International Journal of Electronics and Communications*, vol. 60, no. 4, pp. 267 – 278, 2006.

[7] Y. Zhu, A. Velayutham, O. Oladeji, and R. Sivakumar, "Enhancing tcp for networks with guaranteed bandwidth services," *Computer Networks*, vol. 51, no. 10, pp. 2788 – 2804, 2007.

[8] O. Beaumont and H. Rejeb, "On the importance of bandwidth control mechanisms for scheduling on large scale heterogeneous platforms," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–12.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[10] M. Drozdowski, "New applications of the Muntz and Coffman algorithm," *Journal of Scheduling*, vol. 4, no. 4, pp. 209–223, 2001.

[11] R. Sadykov, "On scheduling malleable jobs to minimise the total weighted completion time," in *Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing, Moscow, Russia*. IFAC-PapersOnLine, 2009, pp. 1497–1499.

[12] R. Motwani, S. Phillips, and E. Torng, "Nonclairvoyant scheduling," *Theoretical Computer Science*, vol. 130, no. 1, pp. 17–47, 1994.

[13] X. Deng, N. Gu, T. Brecht, and K. Lu, "Preemptive scheduling of parallel jobs on multiprocessors," *SIAM J. Comput.*, vol. 30, pp. 145–160, April 2000.

[14] J.-H. Kim and K.-Y. Chwa, "Non-clairvoyant scheduling for weighted flow time," *Information Processing Letters*, vol. 87, no. 1, pp. 31 – 37, 2003.

[15] W. E. Smith, "Various optimizers for single-stage production," *Naval Research Logistics Quarterly*, pp. 59–66, 1956.

[16] R. McNaughton, "Scheduling with deadlines and loss functions," *Management Science*, vol. 6, pp. 1–12, 1959.

[17] T. Kawaguchi and S. Kyan, "Worst case bound of an LRF schedule for the mean weighted flow-time problem," *SIAM Journal on Computing*, vol. 15, p. 1119, 1986.

[18] U. Schwiegelshohn, "An alternative proof of the kawaguchi-kyan bound for the largest-ratio-first rule," *Operations Research Letters*, 2011.

[19] D.-Y. Chen, J.-J. Wu, and P. Liu, "Data-bandwidth-aware job scheduling in grid and cluster environments," in *15th International Conference on Parallel and Distributed Systems (ICPADS'09)*, dec. 2009, pp. 414 –421.

APPENDIX

This appendix is devoted to the proof of Theorem 10, which states that the overall number of preemptions for n malleable tasks scheduled using algorithm **WF** is bounded by $3n$.

As in the fractional case (Section IV-B), to obtain this result, we first prove that the number of changes in the number of resources allocated to a task is bounded by three on average. Then, we prove that it is indeed possible to assign processors to tasks so that at most three preemptions per task take place on average.

The following lemma establishes a bound on the number of resource changes in a schedule.

Lemma 9. *The overall number of changes in the quantity of resources allocated to all tasks using algorithm **WF** is bounded by $3n$, where n denotes the number of tasks.*

Proof: For a given task, we do not consider that there is a change in the number of allocated resources when the task is scheduled for the first time and for the last time, so that the number of changes is closely related to the number of preemptions.

In what follows, after the allocation of tasks $T_1, \ldots, T_i$ using algorithm **WF**, we denote by

- N_i the overall number of changes over time in the quantity of *allocated* resources for tasks $T_1, \ldots, T_i$,
- M_i the number of changes over time in the quantity of *available* resources after the allocation of tasks $T_1, \ldots, T_i$.

The following claim expresses the relation between these two notations.

Claim 1.

$$\forall i \geq 1, \quad N_{i+1} + M_{i+1} \leq N_i + M_i + 3.$$

Proof: Let us consider the allocation of task T_{i+1} using algorithm **WF**. Typically, the allocation consists in two phases, as depicted on Figure 7. During the first phase, that corresponds to columns $right_{i+1}+1$ to $i+1$, the quantity of resources allocated to T_{i+1} is exactly δ_i, so that no change occurs.

During the second phase, that corresponds to columns $left_{i+1}$ to $right_{i+1}$, let us denote by k' the number of columns between $left_{i+1}$ and $right_{i+1}$ that contains a small step (a change of 1 in the number of available resources over time) and by k the number of columns between $left_{i+1}$ and $right_{i+1}$ without such a small step. On Figure 7, the $k' = 3$ columns with a small step are marked with a ★ while the $k = 2$ columns without a step are left empty. Then, each small step, as well as each column, induces a change in the number of processors for T_{i+1}, and there is potentially a new small step at the top of the area (see Figure 7, where each change for T_{i+1} is indicated with a ¶ mark). This gives the following relation on the N_i and M_i:

$$N_{i+1} = N_i + 2k' + k + 1 \text{ and } M_{i+1} = M_i - 2k' - k + 2,$$

so that $N_{i+1} + M_{i+1} \leq N_i + M_i + 3$, which achieves the proof of the claim. ■

When no task is scheduled, we have $N_0 = M_0 = 0$, so that the previous claim gives $N_n + M_n \leq 3n$. The observation that $M_n \geq 1$ concludes the proof of the lemma. ■

The following lemma states that using algorithm **WF**, it is possible to allocate processors to tasks such that the overall number of preemptions is exactly the number of changes in the number of processors allocated to tasks.

Lemma 10. *Let N be the total number of changes in the number of processors allocated to tasks, in the schedule produced by algorithm **WF**. Then, there is an allocation of tasks to processors with no more than N preemptions.*

Proof: Let us define the sequence of instants such that the number of processors allocated to a task changes, or when a task is allocated some processors for the first time, or when the processing of a task completes. By definition, there are at most $N + n$ such instants (since the beginning and the end of tasks always take place at instants C_i by construction). These changes may consist in one of following events: (i) a task sees its number of allocated processors increased, (ii) a new task is allocated some processors, (iii) the number of processors allocated to a task is reduced by one, (iv) the processors allocated to a finishing task are definitely freed.

Let us consider such an instant t and the set of tasks to which resources are allocated, at t^-, t^+ or both. Let us denote by $\mathcal{F}$ the set of resources that are freed by operations (iii) and (iv). The case (iv) does not generate any preemption by definition and case (iii) generates exactly one preemption (one random processor is removed from the set of allocated resources) per change of type (iii) in the resource allocation. By validity of the solution generated by algorithm **WF**, the set $\mathcal{F}$ is enough for all new tasks (case (ii)) and all tasks that see an increase in the quantity of their allocated resources (case (i)). In case (i), the task conserves the set of resources allocated to it and gets some extra resources from $\mathcal{F}$ (that are removed from $\mathcal{F}$), what generates exactly one preemption (new processors allocated to a task) per change of type (i) in the resource allocation.

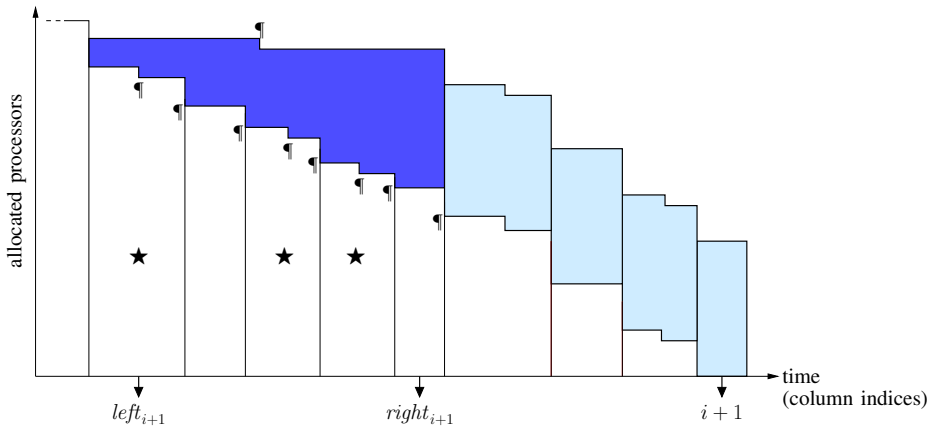

Figure 7. Allocation of task T_{i+1} and number of changes induces.

Therefore, it is possible to allocate processors to tasks such that the number of preemptions exactly corresponds to the number of changes in the quantity of resources allocated to tasks, what achieves the proof of Lemma 10. ■

When combining Lemma 9 with Lemma 10, we obtain the final result on the overall number of preemptions: the overall number of preemptions induced by algorithm **WF** is bounded by $3n$, where n denotes the number of tasks.