Algorithms for hierarchical and semi-partitioned parallel scheduling

Vincenzo Bonifaci, Gianlorenzo d'Angelo, Alberto Marchetti-Spaccamela

Algorithms for hierarchical and semi-partitioned parallel scheduling

Vincenzo Bonifaci
IASI-CNR
Rome, Italy
Email: vincenzo.bonifaci@iasi.cnr.it

Gianlorenzo D'Angelo
Gran Sasso Science Institute
L'Aquila, Italy
Email: gianlorenzo.dangelo@gssi.infn.it

Alberto Marchetti-Spaccamela
Sapienza Università di Roma
Rome, Italy
Email: alberto@diag.uniroma1.it

***Abstract*—We propose a model for scheduling jobs in a parallel machine setting that takes into account the cost of migrations by assuming that the processing time of a job may depend on the specific set of machines among which the job is migrated. For the makespan minimization objective, the model generalizes classical scheduling problems such as unrelated parallel machine scheduling, as well as novel ones such as semi-partitioned and clustered scheduling. In the case of a hierarchical family of machines, we derive a compact integer linear programming formulation of the problem and leverage its fractional relaxation to obtain a polynomial-time 2-approximation algorithm. Extensions that incorporate memory capacity constraints are also discussed.**

***Keywords*-processor affinities; makespan minimization; unrelated machines; laminar family; wrap-around rule; clustered scheduling**

I. Introduction

Multicore architectures have become the standard computing platform in many domains: multicore processors that speed up application performance by dividing the workload among multiple processing cores instead of using one "super fast" single processor. A hierarchical organization of chips of clusters of symmetric multiprocessing (SMP) nodes with multicore chip-multiprocessors (CMP), also known as SMP-CMP clusters, is common today. For example, consider the architecture of Intel's Dual-Core Xeon. In this architecture there are three levels of communication: the communication between two processors on the same chip (intra-CMP communication); the communication across chips but within a node (inter-CMP communication), and the communication between two processors on different nodes (inter-node communication). Intra-CMP communication has higher performance than inter-CMP, which in turn has higher performance than inter-node communication: communications between cores within a chip can usually achieve lower latency and higher bandwidth than communications between cores in different chips.

The objective of how to efficiently exploit the available hardware parallelism for scheduling jobs is crucial. Experimental work (see for example [24] and references therein) has shown that a dynamic scheduler that tries to balance the processes among the available resources to ensure fair distribution of CPU time and to minimize idle cores is not sufficient. The fundamental flaw with this approach is that a core is not an independent processor, but is part of a larger on-chip system that shares resources (such as caches and buses) with other cores. For example, in the multicore system of the dual-core Xeon, cores on the same chip share the same L2 cache and memory controller, and all the cores access the main memory through a shared bus.

Since the communication cost is not uniform, the cost of preempting a job and resuming its execution should take into account the involved cores: resuming execution of a job on the same core is lower than the cost of resuming on a different core; moreover, the cost of migration is not uniform and depends on the communication cost between the two involved cores. For all these reasons, scheduling policies are needed that not only limit the number of migrations, but that are aware of the costs involved in migrations.

Finally, we note that there is a trend in the design of multicore architectures towards heterogenous architectures, providing more flexibility to meet specific performance/energy consumption goals. In fact, heterogeneous multicore architectures have been shown to require significantly less energy without a significant degradation of performance. This results in higher overall efficiency with respect to conventional homogeneous architectures, but implies that the processing time of a job cannot be regarded as a constant.

In this paper, we propose a theoretical model for scheduling jobs in a multicore architecture that can capture the cost of migrations by assuming that *the processing time of a job depends on the specific set of machines on which the job is scheduled*. Namely, we are given a family of admissible sets of machines $\mathcal{A}$, and, for each job j and for each set $\alpha \in \mathcal{A}$, a value $P_j(\alpha)$ denoting the processing time required by j if its execution is limited to the machines in α. We assume that jobs that are assigned to a set α can be executed on any machine in the set; they can be preempted and possibly migrated to another machine in α, but simultaneous processing of the same job by two machines is not allowed (see Section II for details).

This setting opens up a whole new class of scheduling models with their own particular challenges and subsumes well-known problems. For example, if there are m machines and the admissible family $\mathcal{A}$ consists of singletons (i.e., $\mathcal{A} = \{\{1\}, \{2\}, \ldots, \{m\}\}$), then we obtain the unrelated machine scheduling problem [13]; if $\mathcal{A}$ consists of one set containing

all machines (i.e., $\mathcal{A} = \{\{1, 2, \ldots, m\}\}$ then we have the (preemptive) parallel machine scheduling problem.

While the model presented here does not account exactly for the number of migrations incurred, this number can be bounded (see, for example, Proposition III.2), allowing migration costs to be accounted for in the processing times, if desired. Differently from other approaches, this allows for a flexible input representation and easily accommodates heterogeneous processors.

Related work. Much of the prior work on multiprocessor scheduling theory has focused on either the *partitioned* or the *global* approach. Under partitioning, each job is statically assigned to a machine; if the cost of processing a job depends on the specific machine on which the job is executed, we have the unrelated machine scheduling problem; if, for each job j, the processing cost is either p_j or ∞, then we have the restricted assignment problem. Under global scheduling, on the other hand, task migration is allowed with no restrictions and with no additional costs.

It is well-known that partitioning incurs lower runtime overheads (as there are no migrations), but produces schedules that may be unnecessarily constrained; global scheduling, vice versa, entails higher runtime costs that should be properly taken into account (see for example [23]). We now review other approaches that have been proposed and experimentally tested to overcome the above tradeoff between better scheduling policies and higher costs.

Semi-partitioned scheduling was proposed as a compromise between pure partitioned and global scheduling [3]. Semi-partitioning relaxes partitioned scheduling by allowing a small number of jobs to migrate, thereby improving schedulability. Such tasks are called migratory, in contrast to fixed tasks that do not migrate. The common goal in this line of work is to circumvent the algorithmic limitations and resulting capacity loss of partitioning, while avoiding the overhead of global scheduling by limiting migrations.

Clustered scheduling is another proposal that aims to alleviate limitations of partitioned and global algorithms; it exploits the grouping of cores into clusters of symmetric multiprocessing nodes with multicore chip multiprocessors: tasks are statically assigned to clusters (like in partitioning), but are globally scheduled within each cluster [2], [20].

Semi-partitioned and clustered scheduling are not the only two proposals; we briefly mention other proposals. *Federated scheduling* was introduced in [14] to deal with parallel real-time tasks, where each task is a DAG whose nodes represent jobs and edges represent precedence constraints among jobs. Forcing the execution of a single task on a single processor restricts all jobs of a task to execute on the same processor, and forbids to deal with tasks with a (parallelizable) computational demand exceeding the capacity of a single processor. The federated scheduling approach [1], [14] is advocated as a reasonable extension of partitioned scheduling to parallel real-time task sets: there are tasks that are permitted to execute upon more than one processor and are granted exclusive access to the processors upon which they execute, while the remaining tasks are partitioned amongst a pool of shared processors.

We finally observe that contemporary commodity operating systems (such as Linux and Windows) implement more complex migration strategies by defining *processor affinity masks*, which specify on a per-process basis on which processors a job may be scheduled. Namely, processor affinities allow binding a process to an arbitrary subset of processors in the system and a process can only be scheduled on the processors that it is bound to. It is known that processor affinity is useful for increasing the performance of a parallel system in several contexts, such as application performance, fault-tolerance, security and real-time systems [17], [21], [5].

Processor affinities yield a general framework that can be used to realize global, partitioned, or clustered scheduling. For example, in partitioned scheduling, each task's processor affinity includes exactly one processor, while in global scheduling, each task's processor affinity is set to all processors. The new feature is that arbitrary processor affinities can be assigned on a job-by-job basis, which permits the specification of migration strategies that are more flexible than those usually studied in the literature.

To the best of our knowledge, the model we propose has not been considered theoretically. We already observed that the problem of scheduling unrelated machines is a special case of our model; more recently, in [7], [19] a nonpreemptive scheduling problem is considered that is a special case of scheduling unrelated machines (and, thus, of our model). Namely, the problem of non-preemptively scheduling to minimize makespan when for each job j a *unique* set of machines $M(j)$ is given to process the job (i.e., the processing time of job j on machine i is p_j if $i \in M(j)$ or ∞ otherwise); such machine sets have a laminar structure. Glass and Kellerer [7] give a $(2 - 1/m)$-approximation algorithm; Muratore et al. [19] improve this to a polynomial-time approximation scheme.

In [6] (see also references therein) the nonpreemptive scheduling of jobs on a clustered architecture is considered. The authors assume that each cluster is formed by m processors and that the execution of job j requires q_j processors belonging to the same cluster for p_j time units; [6] shows a $7/3$ approximation algorithm for minimizing the makespan. In our model we allow preemption and we consider the more challenging case when for each job there are *many* sets of machines that could execute it, with different processing times; indeed, one of the main difficulties lies in selecting the best processor affinity mask for each job.

Hwang et al. [9] study a model of parallel nonpreemptive scheduling on identical machines, where interprocessor migration costs are explicitly given as part of the input. While potentially more accurate with respect to the migration costs,

we observe that in order to be applied to a preemptive setting, such a model would require to break down each job into a possibly exponential number of unit-size jobs; additionally, the model of Hwang et al. would require a significant extension in the heterogeneous case.

Finally, we remark that in this work we focus on the load balancing and runtime scheduling aspects rather than memory accesses and cache complexity. The reader is referred to [4] and references therein for models that focus on hierarchical cache performance.

Our results. We focus on preemptively scheduling jobs to minimize the makespan assuming a hierarchical architecture, and we first show how this problem generalizes several classical and new problems (Section II).

In Section III we consider, as a warm-up, the case of semi-partitioned scheduling, and we identify necessary and sufficient conditions for schedulability. Namely, we provide an ILP formulation of the assignment problem that, for each job, will specify whether the job is assigned to a specific machine or executed globally. We also provide an efficient scheduler that, given a feasible solution to the ILP, constructs a schedule with the same makespan, thus setting the times for executing and possibly migrating jobs.

In Section IV we consider the more general case of hierarchical scheduling. We provide an ILP formulation of the assignment problem that, for each job, will specify the *affinity mask* that will be used for scheduling the job, and a scheduler that, given a feasible solution to the ILP, constructs a schedule with the same makespan, again setting the times for executing and possibly migrating jobs. We remark that our proposed scheduler is combinatorial and that the schedule cannot be trivially constructed by using standard network flow formulations for scheduling on identical machines.

In Section V we prove an upper bound on the approximation ratio of the problem for the hierarchical setting. We show how a fractional relaxation of the ILP can be leveraged to obtain a polynomial-time 2-approximation algorithm by building on existing algorithms for the unrelated machine scheduling problem; the key lemma of the proof shows how to redistribute the variables' values in a feasible fractional solution across the levels of the machine hierarchy.

Finally, in Section VI we consider extensions of the model to handle additional memory constraints: each job is characterized by a memory requirement in addition to its processing times, and there is a constraint on the available memory at each machine, or at each level of the hierarchy.

Due to space constraints, most of the proofs are given in the full paper.

II. Notation and problem formulation

We are given a set of n jobs $J := \{1, \dots, n\}$ and a set of m machines $M := \{1, \dots, m\}$. Each job needs to be assigned to a *set* of machines on which the job is allowed to schedule, and the job can be preempted and migrated among any such machines. However, its processing time depends on the set of machines on which it is assigned. In detail, we are given a family of admissible sets $\mathcal{A} \subseteq 2^M$, and for each job $j \in J$, a processing time function $P_j : \mathcal{A} \to \mathbb{Z}_+$ with the constraint that the function must be *monotone* on $\mathcal{A}$, i.e., if $\alpha, \beta \in \mathcal{A}$ and $\alpha \subseteq \beta$, then $P_j(\alpha) \leq P_j(\beta)$, modeling the fact that processing overheads (caused, e.g., by migration) increase if the job is executed using a larger set of machines. We often use the shorthand $p_{\alpha j} := P_j(\alpha)$. Moreover, to avoid cumbersome notation, when α is a singleton, such as $\alpha = \{i\}$, we write p_{ij} instead of $p_{\{i\}j}$.

We therefore stipulate that, when a job $j \in J$ is run on a set of machines $\alpha \in \mathcal{A}$, the total processing time it receives must be $P_j(\alpha)$. In general, if a job is run on machine set M' (which may or may not be in $\mathcal{A}$), its processing time must be $p_{\alpha j}$, where α is the inclusion-wise minimal set in $\mathcal{A}$ that contains M' (if there is no such α, then j cannot be run on M').

Given J and $\mathcal{A}$, an *assignment* of jobs in J to sets in $\mathcal{A}$ is a function that assigns each job in J to a set in $\mathcal{A}$. If a job j is assigned to a set α, then its *processing time* is $P_j(\alpha)$. The set α to which a job j is assigned is also called the *affinity mask* of j. Given an assignment of jobs in J to sets in $\mathcal{A}$, a schedule is *valid* with respect to the assignment if each job is scheduled on time slots of machines in its affinity mask, no job is processed in parallel on more than one machine in the same time interval (though it may be preempted or migrated), each job receives the required amount of processing time (i.e. $p_{\alpha j}$, if job j is assigned to set α), and no machine processes more than one job in a time slot. We assume that schedules start at time 0 and allow preemptions and migrations to occur only at integer time points. If, in a given schedule, a job j completes at time C_j, then $T := \max_{j \in J} C_j$ is called the *makespan* of the schedule.

In this paper, we consider the problem of finding an assignment of jobs in J to sets in $\mathcal{A}$ and a corresponding valid schedule that minimizes the makespan. We divide the problem into two subproblems: given J and $\mathcal{A}$, find an assignment of jobs in J to sets in $\mathcal{A}$ that admits a valid schedule in the interval $[0, T]$ and minimizes T; and given an assignment of jobs in J to sets in $\mathcal{A}$ that admits some valid schedule in the interval $[0, T]$, construct a valid schedule in the same interval.

In this paper, we restrict the discussion to *laminar* (or *hierarchical*) instances of the problem, where, for each $\alpha, \alpha' \in \mathcal{A}$, either $\alpha \subseteq \alpha'$ or $\alpha' \subseteq \alpha$ or $\alpha \cap \alpha' = \emptyset$. Without loss of generality, we assume that all sets in the family $\mathcal{A}$ are distinct. In a laminar instance, the *level* of a set β is the number of sets $\alpha \in \mathcal{A}$ such that $\beta \subseteq \alpha$ and the level of the instance is the maximum level among the sets in $\mathcal{A}$. We call the problem with laminar instances the *hierarchical scheduling problem*. The hierarchical scheduling problem generalizes some well-known and new scheduling problems.

- *Identical parallel machines* scheduling with preemption ($P|pmtn|C_{\max}$) [18]: take $\mathcal{A} = \{M\}$. Then each job j can be migrated freely among the machines in M, as long as it receives the processing time p_{Mj}.
- *Unrelated parallel machines* scheduling ($R||C_{\max}$) [13]: take $\mathcal{A}$ to be a family of m singletons, i.e., $\mathcal{A} = \{\{1\}, \{2\}, \ldots, \{m\}\}$. Then each job must be assigned to a single machine (no migration) and its processing time is a function of the machine.
- *Semi-partitioned* scheduling [3]: take $\mathcal{A} = \{M, \{1\}, \{2\}, \ldots, \{m\}\}$. Then each job can either be run *globally* (i.e., freely migrated) on M with processing time p_{Mj}, or assigned *locally* to a specific machine $i \in M$, with processing time $p_{ij} \leq p_{Mj}$.
- *Clustered* scheduling [2]: let $m = kq$. Take $\mathcal{A} = \{M, \{1\}, \ldots, \{m\}, \{1, \ldots, q\}, \{q+1, \ldots, 2q\}, \ldots, \{(k-1)q, \ldots, kq\}\}$. Then each job can be run globally, or locally to a single machine, or locally to a *cluster* of q machines.

Semi-partitioned scheduling generalizes scheduling on unrelated parallel machines (by taking sufficiently large values of p_{Mj}); hence, the following proposition is implied by existing results for the $R||C_{\max}$ problem [13].

Proposition II.1. *Hierarchical and semi-partitioned scheduling are* NP*-hard to approximate within any constant factor less than* $3/2$*.*

In the case of general (non-hierarchical) affinity masks, one can obtain a simple 8-approximation algorithm as follows. Given an instance of the general problem, construct an instance of the unrelated machine problem by assigning for each job j and for each machine i the minimum processing time of job j on machine i among all sets in $\mathcal{A}$ that include i. The optimal *preemptive* makespan of such an instance is a lower bound on the optimum of the original instance with affinity masks, while the optimal *non-preemptive* makespan can be approximated within a factor 2. Considering that there is a factor not greater than 4 between optimal preemptive and non-preemptive schedules [15], we obtain that the 2-approximate solution to the non-preemptive problem is an 8-approximation to the original problem.

The following example shows that, not surprisingly, hierarchical scheduling instances may admit shorter schedules than the corresponding unrelated parallel machine instances.

Example II.1. Consider a semi-partitioned instance with three jobs and two machines: job 1 has $p_{M1} = \infty$, $p_{11} = 1$, $p_{21} = \infty$; job 2 has $p_{M2} = \infty$, $p_{12} = \infty$, $p_{22} = 1$; job 3 has $p_{M3} = p_{13} = p_{23} = 2$ (∞ represents a sufficiently large constant). It is easy to see that the semi-partitioned instance has a schedule with makespan 2, while the corresponding unrelated machine instance has an optimal makespan of 3.

In the sequel, to more easily illustrate the ideas behind our approach, we first discuss the two-level case in Section III. Section IV is devoted to the more general hierarchical setting. We describe the details of the rounding procedure in Section V, which leads to our main result (Theorem V.2). Finally, in Section VI, we discuss how the model can be extended to incorporate memory capacity constraints.

III. Semi-partitioned scheduling

Let $j = 1, \ldots, n$ be a job and $i = 1, \ldots, m$ be a machine; in this section, the special "machine" index 0 will represent the set M, i.e., global processing. We use the following integer linear program (ILP) to determine the minimum makespan. Binary variable x_{ij} encodes the assignment of job j to machine i (or to the set M, if $i = 0$).

$$\min T \tag{IP-1}$$

$$\sum_{i=0}^{m} x_{ij} = 1 \qquad \text{for } j \in J \tag{1a}$$

$$\sum_{j=1}^{n} \sum_{i=0}^{m} p_{ij} x_{ij} \leq mT \tag{1b}$$

$$\sum_{j=1}^{n} p_{ij} x_{ij} \leq T \qquad \text{for } i \in M \tag{1c}$$

$$p_{ij} x_{ij} \leq T \qquad \text{for } j \in J \text{ and } i \in \{0\} \cup M. \tag{1d}$$

It should be clear from the description of the model that the above constraints are necessary for the existence of a valid schedule with makespan T; the fact that they are sufficient is the nontrivial claim that we prove in this section. We show algorithmically that a feasible solution to (IP-1) ensures that a valid schedule with makespan T exists.

Example III.1. Consider the instance of Example II.1. For any finite value of T, the ILP constraints imply $x_{11} = 1$, $x_{22} = 1$; thus, for job 3, we obtain the processing time constraints $2x_{03} \leq T$, $2x_{13} \leq T - 1$, $2x_{23} \leq T - 1$, $2x_{13} + 2x_{23} + 2x_{03} \leq 2T - 2$. The optimal integral solution has $T = 2$ and assigns job 1 to machine 1, job 2 to machine 2, and job 3 globally to both machines. The following schedule has a makespan value of 2: job 1 is scheduled on machine 1 during $[1, 2)$; job 2 is scheduled on machine 2 during $[0, 1)$; finally, job 3 is scheduled on machine 1 during $[0, 1)$, then migrated to machine 2 where it is scheduled during $[1, 2)$.

The pseudo-code of our scheduler is reported in Algorithm 1. The scheduler takes as input a feasible solution to (IP-1) and assigns the jobs to time slots of the machines according to the affinity masks defined in the solution.

Algorithm 1 first schedules the *global jobs* (i.e. jobs j such that $x_{0j} = 1$) so that no job is scheduled simultaneously on two machines. Namely, it computes the total volume V of global jobs and initializes a variable t to 0 (lines 1–2). Then, it iterates on all the machines and assigns to each of them a suitable amount δ of global volume. While

Algorithm 1: Job scheduling (for a given assignment $\mathbf{x}$)

```
1  t ← 0;
2  V ← Σ_{j=1}^{n} p_{0j} x_{0j};
3  while V > 0 do
4      i ← an empty machine (in 1, ..., m);
5      δ ← min(V, T − Σ_{j=1}^{n} p_{ij} x_{ij});
6      Assign δ units of global work to i, in the interval [t, t + δ
         (mod T)];
       /* (The remaining T − δ units on i will be
          used by local jobs)                          */
7      t ← t + δ (mod T);
8      V ← V − δ;
9  foreach machine i ∈ M and job j ∈ J such that x_ij = 1 do
10     Schedule j on machine i in the free time of interval [0, T];
```

$V > 0$, the algorithm looks for an empty machine $i > 0$ (line 4) and schedules δ global volume in the interval $[t, t+\delta \pmod T)]$ on i (line 6). Then it increases the value of t by $\delta \pmod T$ and decreases the volume V of global jobs still to be scheduled by δ (lines 7–8).

The value of δ in each iteration is computed as follows. The total volume of *local* jobs assigned to machine i is $\sum_{j=1}^{n} p_{ij}x_{ij}$, so we can schedule at most $T - \sum_{j=1}^{n} p_{ij}x_{ij}$ volume of *global* jobs on i in the interval $[0, T]$. Therefore, if the volume V of global jobs that still needs to be scheduled is smaller than $T - \sum_{j=1}^{n} p_{ij}x_{ij}$, then $\delta = V$, otherwise, we exploit all the possible empty space on i and then $\delta = T - \sum_{j=1}^{n} p_{ij}x_{ij}$ (line 5).

Having scheduled the global jobs, the algorithm then schedules the *local* jobs (i.e. jobs j such that $x_{ij} = 1$, for $i > 0$) in the free time of each machine (line 10).

Theorem III.1. *Given a feasible solution* $(\mathbf{x}, T)$ *to* (IP-1)*, Algorithm 1 produces a valid schedule in the interval* $[0, T]$.

The following proposition bounds the number of migrations and preemptions that occur in the worst case; this value can be used for a priori bounding the worst-case processing time of a job that is migrated among multiple machines.

Proposition III.2. *The number of job migrations in the schedule produced by Algorithm 1 is at most* $m - 1$*. The number of job preemptions and migrations is at most* $2m-2$*.*

IV. Hierarchical scheduling

The following ILP expresses necessary conditions on the minimum makespan T and an optimal assignment $\mathbf{x}$ in the hierarchical scheduling problem.

$$\min T \qquad \text{(IP-2)}$$

$$\sum_{\alpha \in \mathcal{A}} x_{\alpha j} = 1 \qquad \text{for } j \in J \qquad \text{(2a)}$$

$$\sum_{j=1}^{n} \sum_{\beta \subseteq \alpha} p_{\beta j} x_{\beta j} \leq |\alpha| T \qquad \text{for } \alpha \in \mathcal{A} \qquad \text{(2b)}$$

$$p_{\alpha j} x_{\alpha j} \leq T \qquad \text{for } \alpha \in \mathcal{A},\ j \in J \qquad \text{(2c)}$$

$$x_{\alpha j} \in \{0, 1\} \qquad \text{for } \alpha \in \mathcal{A},\ j \in J \qquad \text{(2d)}$$

Similarly to the previous section, we give an algorithm that takes as input a feasible solution $(\mathbf{x}, T)$ to (IP-2) and constructs a valid schedule with makespan T. The algorithm works in two phases. The first phase (Algorithm 2) proceeds bottom-up (i.e., from the smallest sets up to the largest set) and, for each set of machines $\alpha \in \mathcal{A}$ and for each machine $i \in \alpha$, it *determines the load* $\textsc{load}[i, \alpha]$ *of machine* i *due to jobs assigned to set* α *by the ILP* (i.e., jobs j s.t. $x_{\alpha j} = 1$). The load is assigned in such a way that for each affinity mask the overall load is equal to the sum of the required processing time, that is $\sum_{i \in \alpha} \textsc{load}[i, \alpha] = \sum_{j=1}^{n} p_{\alpha j} x_{\alpha j}$.[1] The second phase (Algorithm 3) proceeds top-down (i.e., from the largest set down to the smallest ones) and, *for each set* $\alpha \in \mathcal{A}$*, determines the schedule of each job* j *such that* $x_{\alpha j} = 1$ *on each machine* $i \in \alpha$, that is, the time slots of each machine $i \in \alpha$ that are assigned to job j.

The crucial observation is that the first phase computes LOAD in such a way that the second phase is able identify only one machine, for each affinity mask, that must be checked in order to avoid to schedule more than one job in the same time interval of the same machine. In fact, the first phase ensures that for each affinity mask $\beta \in \mathcal{A}$ there exists at most one machine $i \in \beta$ that is loaded with some jobs assigned to a superset of β, that is $\textsc{load}[i, \beta] > 0$ and $\textsc{load}[i, \alpha] > 0$, for some $\alpha \in \mathcal{A}$ such that $\beta \subset \alpha$. Since the second phase proceeds top-down, when affinity mask β is analyzed, the schedule of jobs assigned to α in such a machine i is already determined.

Assume that the jobs assigned to α are scheduled in interval $[t, t_{i\alpha}]$, where $t_{i\alpha} = t + \textsc{load}[i, \alpha] \pmod T$; the algorithm first schedules the jobs assigned to β in machine i, in the interval $[t_{i\alpha}, t_{i\beta}]$, where $t_{i\beta} = t_{i\alpha} + \textsc{load}[i, \beta] \pmod T$. Then, it schedules the remaining load by filling up machines $\ell \in \beta \setminus \{i\}$ starting from time $t_{i\beta}$. Indeed, if we assume that the machines in $\beta \setminus \{i\}$ are sorted in an arbitrary way, $\beta \setminus \{i\} = (\ell_1, \ell_2, \ldots, \ell_{|\beta|-1})$, then jobs assigned to β

[1] An alternative approach could be to modify (IP-2) by adding additional fractional variables of the form $y_{\alpha ij}$ and constraints of the form $\sum_i y_{\alpha ij} = x_{\alpha j}$; $y_{\alpha ij}$ represents the fractional share of job j on machine i if job j is scheduled using affinity mask α. However, this may not suffice to guarantee that a valid schedule exists, since a job can only be scheduled on one machine at a time. Conversely, our approach guarantees that a valid schedule exists (Theorem IV.3); moreover, the method is combinatorial and avoids the complication of a larger number of variables.

Algorithm 2: First phase (bottom-up volume allocation)

1 $\text{LOAD}[i,\alpha] = 0$, for each $\alpha \in \mathcal{A}$ and $i \in \alpha$;
2 $\text{TOT-LOAD}[i,\alpha] = 0$, $\alpha \in \mathcal{A}$ and $i \in \alpha$;
3 $\text{MARK}[\alpha] = \texttt{false}$ for each $\alpha \in \mathcal{A}$;
4 **while** $\exists \alpha \in \mathcal{A}$ such that $\neg\text{MARK}[\alpha]$ **do**
5 Let α such that $\neg\text{MARK}[\alpha]$ and ($\text{MARK}[\beta]$, for each $\beta \subset \alpha$);
6 $V \leftarrow \sum_{j=1}^{n} p_{\alpha j} x_{\alpha j}$;
7 **foreach** $i \in \alpha$ in ascending order **do**
8 Let β be the maximal set $\beta \subset \alpha$ such that $i \in \beta$;
9 (if no such β exists, set $\beta = \emptyset$ and $\text{TOT-LOAD}[i,\emptyset] = 0$);
10 $\text{LOAD}[i,\alpha] \leftarrow \min\{V, T - \text{TOT-LOAD}[i,\beta]\}$;
11 $\text{TOT-LOAD}[i,\alpha] \leftarrow \text{TOT-LOAD}[i,\beta] + \text{LOAD}[i,\alpha]$;
12 $V \leftarrow V - \text{LOAD}[i,\alpha]$;
13 $\text{MARK}[\alpha] \leftarrow \texttt{true}$;

Algorithm 3: Second phase (top-down job scheduling)

1 $\text{MARK}[\alpha] \leftarrow \texttt{false}$ for each $\alpha \in \mathcal{A}$;
2 **while** $\exists \beta \in \mathcal{A}$ such that $\neg\text{MARK}[\beta]$ **do**
3 Let β such that $\neg\text{MARK}[\beta]$ and ($\text{MARK}[\alpha]$, for each α such that $\beta \subset \alpha$);
4 **if** $\exists i \in \beta$ such that $\text{LOAD}[i,\beta] > 0$ and $\text{LOAD}[i,\alpha] > 0$, for some set $\alpha \in \mathcal{A}$ such that $\beta \subset \alpha$ **then**
5 Let α be the minimal set such that $\beta \subset \alpha$ and $\text{LOAD}[i,\alpha] > 0$;
6 $t_\beta \leftarrow t_{i\alpha}$;
7 $\ell \leftarrow i$;
8 **else**
9 $t_\beta \leftarrow 0$;
10 $\ell \leftarrow \min \beta$;
11 **foreach** $k \in \beta$ in any order starting from ℓ **do**
12 Assign $\text{LOAD}[k,\beta]$ units of time of jobs j such that $x_{\beta j} = 1$ to machine k, in the interval $[t_\beta, t_\beta + \text{LOAD}[k,\beta] \pmod T]$;
13 $t_\beta \leftarrow t_\beta + \text{LOAD}[k,\beta] \pmod T$;
14 $t_{k\beta} \leftarrow t_\beta$;
15 $\text{MARK}[\beta] \leftarrow \texttt{true}$;

are scheduled in interval $[t_{\ell_{k-1}\beta}, t_{\ell_k\beta}]$ of machine ℓ_k, where $t_{\ell_0\beta} = t_{i\beta}$ and $t_{\ell_k\beta} = t_{\ell_{k-1}\beta} + \text{LOAD}[\ell_k,\beta] \pmod T$. This guarantees that no jobs is scheduled in parallel with itself and that no machine has more than one job scheduled in the same time interval.

The pseudo-code of the first phase is given in Algorithm 2. First, the algorithm initializes variable LOAD (line 1). Then, for each $\alpha \in \mathcal{A}$ and $i \in \alpha$, it initializes variable $\text{TOT-LOAD}[i,\alpha]$, which stores the cumulative load of machine i due to all sets $\beta \subseteq \alpha$ (line 2). Variable $\text{MARK}[\alpha]$ is used to determine whether set $\alpha \in \mathcal{A}$ has been visited or not by the algorithm and it is initialized at line 3. The while loop at lines 4–13 visits all the sets in $\mathcal{A}$ in a bottom-up order: at each iteration it selects a set α such that all its subsets have been already visited, i.e such that $\text{MARK}[\alpha] = \texttt{false}$ and, for each $\beta \subset \alpha$, $\text{MARK}[\beta] = \texttt{true}$ (line 5). At each iteration, variable V stores the volume of jobs assigned to α (i.e. jobs j such that $x_{\alpha j} = 1$) that still needs to be scheduled. Variable V is initialized to the total volume of jobs assigned to α at line 6. The loop at lines 7–12 iterates for each machine $i \in \alpha$ in ascending order and, in order to compute $\text{LOAD}[i,\alpha]$, first selects the maximal subset β of α that contains machine i (if it exists, see line 8–9). The value of $\text{LOAD}[i,\alpha]$ is computed at line 10 as follows. The total volume of jobs already assigned to machine i is equal to the cumulative load of machine i due to all sets $\beta \subset \alpha$, that is $\text{TOT-LOAD}[i,\beta]$. Then, we can schedule at most $T - \text{TOT-LOAD}[i,\beta]$ volume of jobs assigned to α on i. Therefore, if the volume V of global jobs that still needs to be scheduled is smaller than $T - \text{TOT-LOAD}[i,\beta]$, then we assign the entire volume to i and set $\text{LOAD}[i,\alpha] = V$, otherwise, we exploit all the possible empty space and set $\text{LOAD}[i,\alpha] = T - \text{TOT-LOAD}[i,\beta]$. Next, the algorithm computes the value of $\text{TOT-LOAD}[i,\alpha]$ by adding $\text{LOAD}[i,\alpha]$ to $\text{TOT-LOAD}[i,\beta]$ (line 11). Note that, $\text{TOT-LOAD}[i,\alpha] = \sum_{\beta\subseteq\alpha: i\in\beta} \text{LOAD}[i,\beta]$ and, eventually, $\sum_{i\in\alpha} \text{TOT-LOAD}[i,\alpha] = \sum_{\beta\subseteq\alpha} \sum_{i\in\beta} \text{LOAD}[i,\beta]$. Finally, variables V and $\text{MARK}[\alpha]$ are updated at lines 12 and 13. The next lemma shows that the cumulative load on each machine i is at most T and that the volume of the jobs assigned to set α is assigned entirely to variables $\text{LOAD}[i,\alpha]$, for all $i \in \alpha$.

Lemma IV.1. i) *For every $\alpha \in \mathcal{A}$ and $i \in \alpha$,* $\text{TOT-LOAD}[i,\alpha] \leq T$ *at the end of Algorithm 2.*
ii) *Whenever line 13 of Algorithm 2 is executed, $V = 0$.*

Algorithm 2 guarantees that for any set β there exists at most one machine $i \in \beta$ whose load is due to jobs assigned to α and to β, where α is some set such that $\beta \subset \alpha$. This is proven in the next lemma and will be exploited by the second phase of the algorithm.

Lemma IV.2. *For each set $\beta \in \mathcal{A}$ there exists at most one machine $i \in \beta$ such that, for some set $\alpha \in \mathcal{A}$ such that $\beta \subset \alpha$, it holds that* $\text{LOAD}[i,\beta] > 0$ *and* $\text{LOAD}[i,\alpha] > 0$.

Theorem IV.3. *Given a feasible solution $(\mathbf{x}, T)$ to* (IP-2)*, Algorithms 2 and 3 produce a valid schedule in the interval $[0, T]$.*

The pseudo-code of the second phase is given in Algorithm 3. As in Algorithm 2, variable $\text{MARK}[\alpha]$ is used to determine whether a set α has been visited or not. In this case, the algorithm visits all the sets in $\mathcal{A}$ in top-down order (see the while loop at lines 2–15). Variable $t_{i\alpha}$ stores the latest time instant in which a job assigned to set α is scheduled on machine $i \in \alpha$. Let β be a maximal set that has not been visited yet (line 3). By Lemma IV.2, there exists at most one machine $i \in \beta$ such that $\text{LOAD}[i,\beta] > 0$ and $\text{LOAD}[i,\alpha] > 0$, for some set $\alpha \in \mathcal{A}$ such that $\beta \subset \alpha$. If such a machine exists, then let α be the minimal set satisfying the previous condition (line 5) and let ℓ be the unique machine where both sets have some load (line 7). We first schedule jobs assigned to β from time $t_{i\alpha}$ on machine ℓ and then we proceed by scheduling the remaining volume on the empty machines in β as done for global jobs in Algorithm 1. In detail, we initialize t_β to $t_{i\alpha}$ (line 6); for each machine

$k \in \beta$, starting from ℓ, we assign $\text{LOAD}[k, \beta]$ units of time of jobs assigned to β to machine k, in the interval $[t_\beta, t_\beta + \text{LOAD}[k, \beta] \pmod T]$ (line 12); and we update t_β and $t_{k\beta}$ by adding $\text{LOAD}[k, \beta] \pmod T$ (lines 13–14). In the case that there is no machine in β with some loads due to two different sets, the only difference is that ℓ is chosen as the smallest machine in β and t_β is initialized to 0 (lines 9–10).

V. Rounding the ILP

To round (IP-2), we first transform it into a decision form by applying a standard pruning technique [8]. It suffices to decide, for an arbitrary but fixed value of T, the feasibility of the following system:

$$\sum_{\alpha \in \mathcal{A}} x_{\alpha j} = 1 \qquad \text{for } j \in J \qquad \text{(IP-3)}$$

$$\sum_{j \in J} \sum_{\beta \subseteq \alpha} p_{\beta j} x_{\beta j} \le |\alpha| T \qquad \text{for } \alpha \in \mathcal{A} \qquad \text{(3a)}$$

$$x_{\alpha j} \in \{0, 1\} \qquad \text{for } \alpha \in \mathcal{A}, j \in J, \qquad \text{(3b)}$$

$$x_{\alpha j} = 0 \qquad \text{for } (\alpha, j) \notin R, \qquad \text{(3c)}$$

where $R = \{(\alpha, j) \in \mathcal{A} \times J : p_{\alpha j} \le T\}$. We have eliminated constraints (2c) by observing that they are satisfied by a 0-1 solution if and only if $p_{\alpha j} \le T$ whenever $x_{\alpha j} = 1$. Therefore, one can simply set to zero all variables $x_{\alpha j}$ such that $p_{\alpha j} > T$, i.e., the variables with indices not in R. The binary search process for the minimal T for which (IP-3) is feasible requires a number of iterations logarithmic in the range of T, and therefore multiplies the overall running time by only a polynomial factor.

Without loss of generality, we can assume that the family $\mathcal{A}$ always contains the singleton machine sets $\{1\}, \{2\}, \ldots, \{m\}$; if not, these sets can be added to $\mathcal{A}$ by setting the processing time of a job $j \in J$ on machine $i \in M$ as the processing time of j on the (inclusion-wise) minimal set in $\mathcal{A}$ that contains i. Before discussing how to round the fractional relaxation of (IP-3), we show that a feasible fractional solution can always be modified so that, for every $\alpha \in \mathcal{A}$, $x_{\alpha j} = 0$ unless α is a singleton set. This follows easily by repeated application of the next lemma, which allows to "push down" the fractional weights towards the singleton sets of the laminar family.

Lemma V.1. *Let $\eta \in \mathcal{A}$ be a non-singleton set. If $\mathbf{x}$ is a feasible solution to the LP relaxation of* (IP-3)*, then there exists another feasible solution $\mathbf{x}'$ to the same LP relaxation such that $x'_{\eta j} = 0$ and $x'_{\alpha j} = x_{\alpha j}$ whenever $\alpha \not\subseteq \eta$.*

Proof: For any $\alpha \in \mathcal{A}$, we define the *slack* of α in $\mathbf{x}$ to be

$$\text{slack}(\alpha, \mathbf{x}) := |\alpha| T - \sum_{j \in J} \sum_{\beta \subseteq \alpha} p_{\beta j} x_{\beta j}.$$

Note that the LP relaxation of (IP-3) can be written as

$$\sum_{\alpha \in \mathcal{A}} x_{\alpha j} = 1 \qquad \text{for } j \in J \qquad \text{(4a)}$$

$$\text{slack}(\alpha, \mathbf{x}) \ge 0 \qquad \text{for } \alpha \in \mathcal{A} \qquad \text{(4b)}$$

$$x_{\alpha j} \ge 0 \qquad \text{for } \alpha \in \mathcal{A}, j \in J \qquad \text{(4c)}$$

$$x_{\alpha j} = 0 \qquad \text{for } (\alpha, j) \notin R. \qquad \text{(4d)}$$

Without loss of generality, assume that $\eta = \beta_1 \cup \ldots \cup \beta_q$ with $\beta_1, \ldots, \beta_q \in \mathcal{A}$, $\beta_1, \ldots, \beta_q \subset \eta$, the sets $\beta_1, \ldots, \beta_q$ being maximal and pairwise disjoint. Because η has non-negative slack in $\mathbf{x}$, we have

$$\sum_{j \in J} \sum_{\gamma \subseteq \beta_1} p_{\gamma j} x_{\gamma j} + \ldots + \sum_{j \in J} \sum_{\gamma \subseteq \beta_q} p_{\gamma j} x_{\gamma j} + \sum_{j \in J} p_{\eta j} x_{\eta j} \le |\beta_1| T + \ldots + |\beta_q| T,$$

which is equivalent to

$$\sum_{j \in J} p_{\eta j} x_{\eta j} \le \text{slack}(\beta_1, \mathbf{x}) + \ldots + \text{slack}(\beta_q, \mathbf{x}). \qquad (5)$$

We now define a new solution $\mathbf{x}'$ by setting $x'_{\eta j} = 0$, $x'_{\alpha j} = x_{\alpha j}$ for $\alpha \ne \beta_1, \ldots, \beta_q$, and

$$x'_{\beta j} = x_{\beta j} + \frac{\text{slack}(\beta, \mathbf{x})}{\text{slack}(\beta_1, \mathbf{x}) + \ldots + \text{slack}(\beta_q, \mathbf{x})} \cdot x_{\eta j} \qquad (6)$$

for $\beta = \beta_1, \ldots, \beta_q$. We claim that $\mathbf{x}'$ is valid for the LP. To see that (4a) is satisfied by the new solution, note that

$$\begin{aligned} \sum_{\alpha \in \mathcal{A}} x'_{\alpha j} &= \\ &= \sum_{\substack{\alpha \in \mathcal{A} \\ \alpha \ne \eta}} x_{\alpha j} + \sum_{i=1}^{q} \frac{\text{slack}(\beta_i, \mathbf{x})}{\text{slack}(\beta_1, \mathbf{x}) + \ldots + \text{slack}(\beta_q, \mathbf{x})} \cdot x_{\eta j} \\ &= \sum_{\alpha \in \mathcal{A}} x_{\alpha j} = 1. \end{aligned}$$

To see that (4b) is satisfied, it suffices to show that the new slack of $\beta_1, \ldots, \beta_q$ is nonnegative, since the slack of any other set does not decrease. Consider, say, β_i. By summing (6) across jobs,

$$\begin{aligned} &\sum_{j \in J} \sum_{\gamma \subseteq \beta_i} p_{\gamma j} x'_{\gamma j} = \\ &\sum_{j \in J} \sum_{\gamma \subseteq \beta_i} p_{\gamma j} x_{\gamma j} + \\ &\quad + \frac{\text{slack}(\beta_i, \mathbf{x})}{\text{slack}(\beta_1, \mathbf{x}) + \ldots + \text{slack}(\beta_q, \mathbf{x})} \sum_{j \in J} p_{\beta_i j} x_{\eta j} \le \\ &\sum_{j \in J} \sum_{\gamma \subseteq \beta_i} p_{\gamma j} x_{\gamma j} + \\ &\quad + \frac{\text{slack}(\beta_i, \mathbf{x})}{\text{slack}(\beta_1, \mathbf{x}) + \ldots + \text{slack}(\beta_q, \mathbf{x})} \sum_{j \in J} p_{\eta j} x_{\eta j} \le \\ &\sum_{j \in J} \sum_{\gamma \subseteq \beta_i} p_{\gamma j} x_{\gamma j} + \text{slack}(\beta_i, \mathbf{x}), \end{aligned}$$

where for the first inequality we used the monotonicity of the processing times, and for the second inequality we used (5). Therefore,

$$\begin{aligned}&\text{slack}(\beta_i, \mathbf{x}') = \\ &= \text{slack}(\beta_i, \mathbf{x}) - \sum_{j\in J}\sum_{\gamma\subseteq\beta_i} p_{\gamma j}x'_{\gamma j} + \sum_{j\in J}\sum_{\gamma\subseteq\beta_i} p_{\gamma j}x_{\gamma j} \\ &\geq \text{slack}(\beta_i, \mathbf{x}) - \text{slack}(\beta_i, \mathbf{x}) \geq 0.\end{aligned}$$

■

Equipped with the above lemma, we can proceed to prove an upper bound on the approximability of the hierarchical scheduling problem.

Theorem V.2. *The hierarchical scheduling problem admits a polynomial-time 2-approximation algorithm.*

Proof: Consider an instance $I = (J, M, \mathcal{A}, p)$ of the hierarchical scheduling problem.

Let T^* be the minimum value of T for which the LP relaxation of (IP-3) is feasible. Clearly, $T^* \leq \text{opt}(I)$ where $\text{opt}(I)$ is the optimal makespan of the hierarchical scheduling instance I. By applying repeatedly Lemma V.1, we can ensure that there exists a feasible fractional solution $\mathbf{x}$ with makespan T^* and such that $x_{\alpha j} > 0$ only for α such that $|\alpha| = 1$. Because of this, observe that $\mathbf{x}$ can also be seen as a fractional solution to an unrelated machines scheduling instance with makespan T^*: the instance $I_u = (J, M, p')$ obtained by defining $p'_{ij} := p_{\{i\}j}$.

The idea is now to invoke an existing LP-based algorithm for the unrelated machine scheduling problem and run it on I_u. The classic rounding algorithm by Lenstra, Shmoys and Tardos [13] constructs an integral assignment $\bar{\mathbf{x}}$ for I_u with makespan $T_u \leq 2T^*$. The assignment $\bar{\mathbf{x}}$, extended with 0 values on all sets with $|\alpha| > 1$, is also valid for (IP-3) if we take $T = T_u \leq 2T^* \leq 2 \cdot \text{opt}(I)$. Therefore, such an extended assignment yields a 2-approximate solution for the hierarchical scheduling problem. ■

We note that the reduction used in the above proof, from fractional hierarchical scheduling to fractional unrelated machines, is *not* valid for the original (integral) formulations of the two problems: indeed, in Example II.1, the original instance I of the semi-partitioned problem has an optimal makespan of 2, while the unrelated machine instance I_u has an optimal makespan of 3. In general, the gap between the makespan of I_u and the makespan of I can be arbitrarily close to a factor 2, as the next example shows.

Example V.1. Consider a semi-partitioned instance I with n jobs and $m := n-1$ machines. Recall that we use machine index 0 to denote global processing. Job j, $j = 1, \ldots, n-1$, has $p_{ij} = n-2$ if $i = j$, and $p_{ij} = \infty$ otherwise. Job n has $p_{ij} = n-1$ for each $i = 0, 1, \ldots, n-1$. An optimal solution has makespan $\text{opt}(I) = n-1$: assign job j, $j = 1, \ldots, n-1$ to machine j and assign job n globally; schedule each job j, $j = 1, \ldots, n-1$, on machine j in time intervals $[0, j-1)$ and $[j, n-1)$; schedule job n on machine i, $i = 1, \ldots, n-1$ during $[i-1, i)$. On the other hand, in the corresponding unrelated machine instance I_u, jobs cannot be migrated and therefore the minimum value of the makespan is $2n-3$.

VI. Memory constraints

The basic model as described in the previous sections focuses on makespan minimization, without additional constraints. However, machines often also have limited *memory capacity*. In this section we show how, in the hierarchical case, our model can be extended to incorporate memory capacities and discuss how to obtain efficient algorithms with a guaranteed bicriteria approximation ratio. We consider two distinct extensions, which we call Model 1 and Model 2.

Model 1: We assume that each machine $i \in M$ has some memory budget $B_i \in \mathbb{Z}_+$ and that each job $j \in J$ requires memory space $s_{ij} \in \mathbb{Z}_+$ when run on machine i. We require the jobs assigned to sets that include machine i to fit the memory bound B_i; i.e., if j is assigned to a set of machines α, then its space requirement is counted towards each machine in α. Thus, we revise ILP (IP-3) by adding the capacity constraints

$$\sum_{j\in J}\left(s_{ij}\cdot\sum_{\alpha\in\mathcal{A}\,:\,i\in\alpha} x_{\alpha j}\right) \leq B_i \qquad \text{for each } i \in M. \quad (7)$$

To round the revised ILP, we apply the *iterative rounding* approach [10], [12], [16], allowing us to prove the following theorem.

Theorem VI.1. *Whenever ILP* (IP-3) *has a solution satisfying constraints* (7)*, it is possible to construct, in polynomial time, a valid schedule with makespan at most* $3T$ *such that*

$$\sum_{j\in J}\left(s_{ij}\cdot\sum_{\alpha\in\mathcal{A}\,:\,i\in\alpha} x_{\alpha j}\right) \leq 3\cdot B_i \qquad \textit{for each } i \in M. \quad (8)$$

where $\mathbf{x}$ *represents the schedule's assignment.*

Model 2: Assume for simplicity that the forest associated to the laminar family $\mathcal{A}$ is a tree such that every leaf has the same level; it can be shown that this assumption is not a limitation to the model, but we defer the details to the full version of the paper. For a node α of the tree, define $\text{h}(\alpha)$ (the *height* of α) to be the shortest distance in the tree from α to a leaf. Thus $\text{h}(\alpha) = 0$ iff α is a leaf of the tree, and the height of the root is $k-1$ in a k-level instance. We assume that job j requires space $s_j \leq 1$ (with $s_j \in \mathbb{Q}_+$), that each node of height h except the root has memory capacity μ^h (where $\mu > 1$ is a parameter that captures how the memory hierarchy scales) and that the root has unbounded capacity. We require the jobs assigned to a set (node of the tree) to fit the memory capacity of that set. Thus, we revise ILP (IP-3) by adding the capacity constraints

$$\sum_{j\in J} s_j x_{\alpha j} \leq \mu^{\text{h}(\alpha)} \text{ for each } \alpha \in \mathcal{A}\setminus\{M\}. \quad (9)$$

Call (IP-4) the revised ILP obtained in this way.

The additional constraints affect the applicability of Theorem V.2, since one cannot use Lemma V.1 to "push down" the values of the fractional variables towards the leaves of the laminar family. Moreover, known rounding techniques such as [11], [16] are not suitable in this case. Indeed, one cannot apply the result of Karp et al. [11] because it does not ensure that the resulting integral vector satisfies the assignment constraints exactly; while the technique of [16] would yield a possibly large approximation factor, equal to $1 + k$, where k is the number of levels of the hierarchy.

Instead, we introduce a modified iterative rounding scheme (Lemma VI.2 below), which satisfies exactly the assignment constraints and yields a bound in terms of the *sum* of the column's entries of the (normalized) coefficient matrix; when applied to (IP-4), this guarantees $O(\log k)$ approximation of the packing constraints. Such a rounding scheme applies to general assignment and packing constraints, and thus may find applications beyond the hierarchical scheduling problem.

Lemma VI.2. *Let I, J be nonempty finite sets, and $R \subseteq I \times J$. Consider a linear program of the form:*

$$\min \sum_{(i,j)\in R} c_{ij} z_{ij} \tag{LP}$$

$$\sum_{i:(i,j)\in R} z_{ij} = 1 \qquad \forall j \in J \tag{10a}$$

$$\sum_{q=(i,j)\in R} a_{lq} z_{ij} \le b_l \qquad l = 1, \dots, \theta \tag{10b}$$

$$0 \le z_{ij} \le 1 \qquad \forall (i,j) \in R, \tag{10c}$$

where z_{ij} are variables, $\theta \in \mathbb{N}$, and $a_{lq} \ge 0$, $b_l > 0$, $c_{ij} \ge 0$ for all $l = 1, \dots, \theta$, $q = (i,j) \in R$. Assume that the LP has a feasible solution $\mathbf{z}^0$ and that the bound $\sum_{l=1}^{\theta} a_{lq}/b_l \le \rho$ holds for each $q \in R$. Then there are values $\bar{z}_{ij}$ such that

$$\sum_{(i,j)\in R} c_{ij} \bar{z}_{ij} \le \sum_{(i,j)\in R} c_{ij} z^0_{ij} \tag{11a}$$

$$\sum_{i:(i,j)\in R} \bar{z}_{ij} = 1 \qquad \forall j \in J \tag{11b}$$

$$\sum_{q=(i,j)\in R} a_{lq} \bar{z}_{ij} \le (1+\rho) b_l \qquad l = 1, \dots, \theta \tag{11c}$$

$$\bar{z}_{ij} \in \{0, 1\} \qquad \forall (i,j) \in R. \tag{11d}$$

Equipped with the rounding lemma, we can proceed to prove our result for Model 2.

Theorem VI.3. *Let k be the number of levels of the laminar family $\mathcal{A}$ and let H_k be the kth harmonic number. Whenever ILP (IP-4) has a solution, it is possible to construct, in polynomial time, a valid schedule with makespan at most $\sigma \cdot T$ such that*

$$\sum_{j\in J} s_j x_{\alpha j} \le \sigma \cdot \mu^{\mathrm{h}(\alpha)} \text{ for each } \alpha \in \mathcal{A} \setminus \{M\}, \tag{12}$$

where $\mathbf{x}$ represents the schedule's assignment and $\sigma = 2 + H_k$. When $k = 2$, the same holds with $\sigma = 3 + 1/m$.

Proof: We observe that (IP-4) is in a form suitable for applying Lemma VI.2. In particular, we take I in Lemma VI.2 to be the admissible sets family $\mathcal{A}$, and we use the generic packing constraints (11c) to encode constraints (3a) and (9). Indeed, constraints (3a) can be encoded as $|\mathcal{A}|$ constraints with coefficients of the form

$$a_{\alpha,(\beta,j)} := \begin{cases} p_{\beta j} & \text{if } \beta \subseteq \alpha, \\ 0 & \text{otherwise,} \end{cases}, \quad b_\alpha := |\alpha| T, \quad (\alpha \in \mathcal{A}),$$

while constraints (9) can be encoded as $|\mathcal{A}| - 1$ constraints with coefficients of the form

$$a_{\alpha,(\beta,j)} := \begin{cases} s_j & \text{if } \beta = \alpha, \\ 0 & \text{otherwise,} \end{cases}, \; b_\alpha := \mu^{\mathrm{h}(\alpha)}, \; (\alpha \in \mathcal{A} \setminus \{M\}).$$

The cost coefficients c_{ij} in (LP) can be set to zero and (LP) becomes the linear relaxation of (IP-4). By our hypothesis, such LP relaxation must be feasible, and the hypothesis of Lemma VI.2 is satisfied. In particular, we can take $\mathbf{z}^0$ to be an optimal solution of the LP relaxation. Note that the LP relaxation can be solved in polynomial time; in particular, since $\mathcal{A}$ is laminar, $|\mathcal{A}| \le 2m$ (see, e.g., [22, Theorem 3.5]) and the number of constraints in (IP-4) is polynomial in n and m.

To choose an appropriate value of ρ in Lemma VI.2, note that $(\beta, j) \in R$ only when $p_{\beta j} \le T$ by construction, so if $q = (\beta, j)$,

$$\sum_{l=1}^{\theta} \frac{a_{lq}}{b_l} = \sum_{\alpha\in\mathcal{A}:\beta\subseteq\alpha} \frac{p_{\beta j}}{|\alpha|\, T} + \frac{s_j}{\mu^{\mathrm{h}(\beta)}} \le \sum_{\alpha\in\mathcal{A}:\beta\subseteq\alpha} \frac{1}{|\alpha|} + 1$$
$$\le 1 + \sum_{1\le i\le k} \frac{1}{i} = 1 + H_k,$$

where for the first inequality we also used $s_j \le 1 < \mu$, and for the second the fact that the laminar family $\mathcal{A}$ has k levels and the fact that all sets in $\mathcal{A}$ are distinct and nonempty. Thus, we can apply Lemma VI.2 with $\rho = 1 + H_k$.

In the semi-partitioned case (i.e., when $\mathcal{A}$ has $k = 2$ levels), the summation $\sum_{l=1}^{\theta} a_{lq}/b_l$ involves at most three nonzero terms. The first term is due to the local scheduling constraints and has the form p_{ij}/T, which is at most 1. The second term is due to the global scheduling constraint and has the form p_{ij}/mT, which is at most $1/m$. The third term is due to the memory constraints and has the form $s_j/\mu^{\mathrm{h}(\beta)}$, which is at most 1. Therefore, $\rho = 2 + 1/m$ is sufficient.

Finally, the integer solution $\mathbf{x}$ to (11), which can be found by applying Lemma VI.2, can be used to construct a schedule with makespan at most $(1 + \rho)T$ and satisfying

(12) by feeding $\mathbf{x}$ to the algorithms presented in Sections III and IV. ■

ACKNOWLEDGMENT

We would like to thank an anonymous reviewer for suggesting the 8-approximation algorithm for the non-hierarchical case.

REFERENCES

[1] S. Baruah. Federated scheduling of sporadic DAG task systems. In *Proc. 2015 IEEE International Parallel and Distributed Processing Symposium*, pages 179–186, 2015.

[2] A. Bastoni, B. B. Brandenburg, and J. H. Anderson. An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers. In *Proc. of the 31st IEEE Real-Time Systems Symposium*, pages 14–24. IEEE, 2010.

[3] A. Bastoni, B. B. Brandenburg, and J. H. Anderson. Is semi-partitioned scheduling practical? In *Proc. of the 23rd Euromicro Conf. on Real-Time Systems*, pages 125–135. IEEE, 2011.

[4] G. E. Blelloch, J. T. Fineman, P. B. Gibbons, and H. V. Simhadri. Scheduling irregular parallel computations on hierarchical caches. In *Proc. of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 355–366. ACM, 2011.

[5] V. Bonifaci, B. B. Brandenburg, G. D'Angelo, and A. Marchetti-Spaccamela. Multiprocessor real-time scheduling with hierarchical processor affinities. In *Proc. 28th Euromicro Conf. on Real-Time Systems*, pages 237–247, 2016.

[6] M. Bougeret, P.-F. Dutot, D. Trystram, K. Jansen, and C. Robenek. Improved approximation algorithms for scheduling parallel jobs on identical clusters. *Theoretical Computer Science*, 600:70 – 85, 2015.

[7] C. A. Glass and H. Kellerer. Parallel machine scheduling with job assignment restrictions. *Naval Research Logistics*, 54(3):250–257, 2007.

[8] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *J. ACM*, 34(1):144–162, 1987.

[9] J. Hwang, Y. Chow, F. D. Anger, and C. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM J. Comput.*, 18(2):244–257, 1989.

[10] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[11] R. M. Karp, F. T. Leighton, R. L. Rivest, C. D. Thompson, U. V. Vazirani, and V. V. Vazirani. Global wire routing in two-dimensional arrays. *Algorithmica*, 2:113–129, 1987.

[12] L. Lau, R. Ravi, and M. Singh. *Iterative Methods in Combinatorial Optimization*. Cambridge University Press, 2011.

[13] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990.

[14] J. Li, J. Chen, K. Agrawal, C. Lu, C. D. Gill, and A. Saifullah. Analysis of federated and global scheduling for parallel real-time tasks. In *Proc. 26th Euromicro Conf. on Real-Time Systems*, pages 85–96, 2014.

[15] J. Lin and J. S. Vitter. ϵ-approximations with minimum packing constraint violation (extended abstract). In *Proc. 24th Annual ACM Symposium on Theory of Computing*, pages 771–782. ACM, 1992.

[16] A. Marchetti-Spaccamela, C. Rutten, S. van der Ster, and A. Wiese. Assigning sporadic tasks to unrelated machines. *Math. Program.*, 152(1-2):247–274, 2015.

[17] E. P. Markatos and T. J. LeBlanc. Using processor affinity in loop scheduling on shared-memory multiprocessors. *IEEE Trans. Parallel Distrib. Syst.*, 5(4):379–400, 1994.

[18] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.

[19] G. Muratore, U. M. Schwarz, and G. J. Woeginger. Parallel machine scheduling with nested job assignment restrictions. *Operations Research Letters*, 38(1):47 – 50, 2010.

[20] R. Rabenseifner, G. Hager, and G. Jost. Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. In *Proc. of the 17th Euromicro International Conf. on Parallel, Distributed and Network-Based Processing*, pages 427–436. IEEE, 2009.

[21] J. D. Salehi, J. F. Kurose, and D. F. Towsley. The effectiveness of affinity-based scheduling in multiprocessor network protocol processing (extended version). *IEEE/ACM Trans. Netw.*, 4(4):516–530, 1996.

[22] A. Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, 2003.

[23] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam. An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration. *IEEE Trans. Parallel Distrib. Syst.*, 14(3):236–247, 2003.

[24] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto. Survey of scheduling techniques for addressing shared resources in multicore processors. *ACM Comput. Surv.*, 45(1):4, 2012.