Algorithms for Hierarchical and Semi-Partitioned Parallel Scheduling

Vincenzo Bonifaci[a,*], Gianlorenzo D'Angelo[b], Alberto Marchetti-Spaccamela[c]

[a] *Università Roma Tre, Rome, Italy*
[b] *Gran Sasso Science Institute, L'Aquila, Italy*
[c] *Sapienza Università di Roma, Rome, Italy*

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 (ILP) formulation for the job assignment subproblem, and show how to turn arbitrary ILP solutions into valid schedules. We also derive a polynomial-time 2-approximation algorithm for the problem. Extensions that incorporate memory capacity constraints are also discussed.

Keywords: processor affinities, makespan minimization, unrelated machines, laminar family, wrap-around rule, clustered scheduling

1. Introduction

Multicore architectures have become the standard computing platform in many domains: multicore processors 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 (see Figure 1). In this architecture there are three levels of communication: the communication between two processors on the same chip (intra-CMP communication); the communication across

*Corresponding author. This work was carried out in association with IASI–Consiglio Nazionale delle Ricerche.

Email addresses: vincenzo.bonifaci@uniroma3.it (Vincenzo Bonifaci), gianlorenzo.dangelo@gssi.it (Gianlorenzo D'Angelo), alberto@diag.uniroma1.it (Alberto Marchetti-Spaccamela)

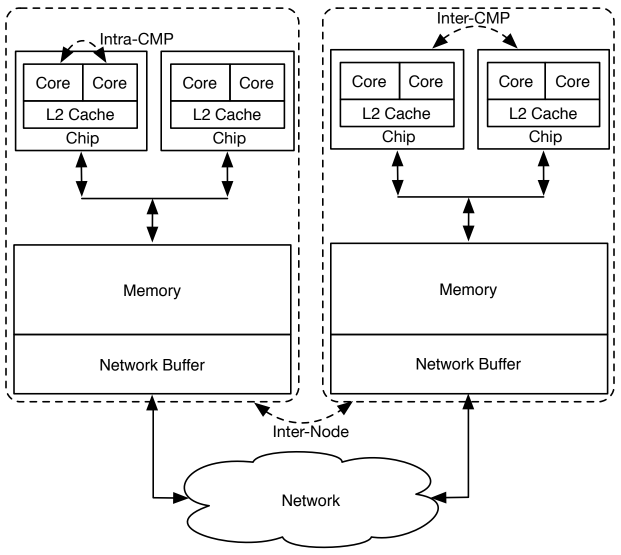


Figure 1: Example of a multicore cluster.

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 [29] 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 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: the cost of 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 also aware of the costs involved in migrations.

Additionally, 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 multi-core 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 multi-core 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 2 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 machines scheduling problem [17]; 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 [22].

Our work focuses for the most part on the case of families $\mathcal{A}$ that are hierarchical. In the case of general, non-hierarchical admissible families, we show how to derive a 2-approximation algorithm by connecting the problem to the unrelated machines scheduling problem. This result can also be seen as a specific way to round the ILPs we derive for the hierarchical case. Nevertheless, in practice the ILP solution found by a solver may have a substantially better quality than that guaranteed by the 2-approximation result, hence a substantial part of our work focuses on showing how, in the hierarchical setting, a *generic* ILP solution can be converted into a hierarchical schedule, which is not trivial.

While the model presented here does not account exactly for the number of migrations incurred, this number can be bounded (see Propositions 3.4 and 4.7), 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 machines. In fact, we also show how our basic model can be extended to incorporate memory capacity constraints.

A non-obvious property of our model is that, if one is satisfied with a 2-approximate solution, migrations can be avoided completely in the setting without memory constraints (see Section 6). But in general, migrations are helpful.

1.1. 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 machines scheduling problem [17]. Under global scheduling, on the other hand, task migration is allowed with no restrictions and with no additional costs [16, 22].

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 [28]). 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 statically assigned tasks. 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, 24].

Semi-partitioned and clustered scheduling are not the only two proposals; we briefly mention other proposals. *Federated scheduling* was introduced in [18] 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, 18] 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 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 [5, 21, 25].

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. In the rest of this section we highlight the differences with previous results on somewhat similar, but distinct, models.

We already observed that the problem of scheduling unrelated machines is a special case of our model; more recently, in [8, 13, 23] a non-preemptive scheduling problem is considered that is a special case of scheduling unrelated machines (and, thus, of our model). Namely, the *restricted assignement* problem consists in finding a schedule that minimizes the makespan when each job j a can be scheduled only on a set of machines $M(j)$ (i.e., the processing time of job j on machine i is p_j if $i \in M(j)$ or ∞ otherwise). For the case when machine sets have a laminar structure, Glass and Kellerer [8] give a $(2-1/m)$-approximation algorithm and Muratore et al. [23] improve this to a polynomial-time approximation scheme (PTAS). Jansen, Maack and Solis-Oba [13] derive the PTAS of Muratore et al. in a more general context for the restricted assignment problem, where the incidence structure between jobs and machines can be more general.

Bougeret et al. [6] (see also references therein) consider the non-preemptive scheduling of jobs on a clustered architecture. 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; they develop 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. [10] study a model of parallel non-preemptive scheduling on identical machines, where interprocessor communication times are explicitly given as part of the input. While potentially more accurate, 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.

1.2. Contributions and Organization of the Paper

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 2).

In Section 3 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 integer linear programming (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 4 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 5 we prove an upper bound on the approximation ratio of the problem for the general set-monotone scheduling problem. We show how to obtain a polynomial-time 2-approximation algorithm by building on existing algorithms for the unrelated machines scheduling problem.

In Section 6 we discuss the relative power of migratory versus non-migratory schedules in our model.

Finally, in Section 7 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 cluster of the hierarchy.

2. Notation and problem formulation

We are given a set of n jobs $J := \{1, \ldots, n\}$ and a set of m machines $M := \{1, \ldots, m\}$. Each job needs to be assigned to a *set* of machines on which the job is allowed to be 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 $\alpha \in \mathcal{A}$ is a set that minimizes the processing time of j among those that contain M' (if there is no such α, then j cannot be run on M'). Since P_j is monotone on $\mathcal{A}$, then α is an inclusion-wise minimal set in $\mathcal{A}$ that contains 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, for the most part 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 ($P|pmtn|C_{\max}$) [22]: 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 ($R||C_{\max}$) [17]: take $\mathcal{A}$ to be a family of m singletons, i.e., $\mathcal{A} = \{\{1\}, \{2\}, \dots, \{m\}\}$. 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 $\mathcal{A} = \{M, \{1\}, \{2\}, \dots, \{m\}\}$. 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 $\mathcal{A} = \{M, \{1\}, \dots, \{m\}, \{1, \dots, q\}, \{q+1, \dots, 2q\}, \dots, \{(k-1)q, \dots, kq\}\}$. Then each job can be run globally, or locally to a single machine, or locally to a *cluster* of q machines.

In a few cases, we drop the assumption that the instances are laminar, but maintain monotonicity of the processing times (i.e. $\alpha \subseteq \beta$ implies $P_j(\alpha) \leq P_j(\beta)$). We call this more general setting the *set-monotone scheduling problem.*

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 [17].

Proposition 2.1. *Set-monotone scheduling, hierarchical scheduling and semi-partitioned scheduling are* NP*-hard to approximate within any constant factor less than* $3/2$.

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

Example 2.1. (See also Figure 2.) 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.

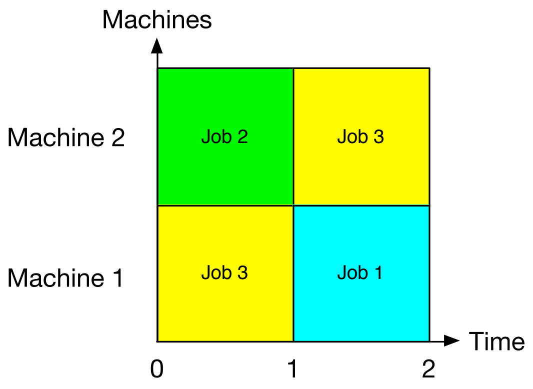


Figure 2: Optimal schedule for the instance of Example 2.1.

In the sequel, to more easily illustrate the ideas behind our approach, we first discuss an integer linear program (ILP) for the two-level case in Section 3. Section 4 is devoted to the more general hierarchical setting. We describe how to derive a polynomial-time 2-approximation algorithm for the set-monotone scheduling problem in Section 5. The relative power of migratory versus non-migratory schedules in our model is discussed in Section 6. Finally, in Section 7, we discuss how the model can be extended to incorporate memory capacity constraints.

3. Semi-partitioned scheduling

Let $j = 1, \dots, n$ be a job and $i = 1, \dots, m$ 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 $i = 0$).

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

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

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

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

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

$$x_{ij} \in \{0,1\} \qquad \text{for } j \in J \text{ and } i \in \{0\} \cup M. \tag{1e}$$

Note that constraint (1b) is not implied by the constraints (1c), due to the additional presence of the x_{0j} variables. 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 3.1. Consider the instance of Example 2.1. 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 $2x_{03} \le T$, $2x_{13} \le T - 1$, $2x_{23} \le T - 1$, $2x_{13} + 2x_{23} + 2x_{03} \le 2T - 2$. 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 over all the machines and assigns to each of them a suitable amount δ of global volume. While $V > 0$, the algorithm looks for an empty machine $i > 0$ (line 4) and schedules δ global volume in the interval $[t, t + \delta$

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

```
1  $t \leftarrow 0$;
2  $V \leftarrow \sum_{j=1}^{n} p_{0j} x_{0j}$;
3  while $V > 0$ do
4      $i \leftarrow$ an empty machine (in $1, \ldots, m$);
5      $\delta \leftarrow \min(V, T - \sum_{j=1}^{n} p_{ij} x_{ij})$;
6      Assign $\delta$ units of global work to $i$, in the interval $[t, t+\delta \pmod{T})$];
       /* (The remaining $T - \delta$ units on $i$ will be used by local jobs)    */
7      $t \leftarrow t + \delta \pmod{T}$;
8      $V \leftarrow V - \delta$;
9  end
10 foreach machine $i \in M$ and job $j \in J$ such that $x_{ij} = 1$ do
11     Schedule $j$ on machine $i$ in the free time of interval $[0, T]$;
12 end
```

$\pmod{T})]$ on i (line 6).[1] 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 11).

In order to show that Algorithm 1 produces a valid schedule, we need to prove the two next lemmas.

Lemma 3.1. *In the schedule produced by Algorithm 1, all jobs receive the required amount of processing time.*

Proof. We first show the statement for global jobs, in particular we show that $V = 0$ at the end of the while loop. By contradiction, assume that at the end of some iteration of the while loop $V > 0$ and there is no empty machine left to be selected at line 4 of the next iteration. This implies that for each machine i, the amount δ of global volume scheduled on i is $T - \sum_{j=1}^{n} p_{ij} x_{ij}$ (see lines 5 and 8). Therefore, the overall amount of scheduled global jobs is $\sum_{i=1}^{m}(T - \sum_{j=1}^{n} p_{ij} x_{ij})$

[1] Global jobs are assigned in an arbitrary order in the reserved interval. Note that $t + \delta \pmod{T}$ might be smaller than t, in this case global jobs assigned to machine i are first scheduled in the interval $[0, t+\delta \pmod{T})]$ and then in the interval $[t, T]$, possibly a global job is preempted at time $t + \delta \pmod{T}$.

and the amount of global jobs still to be scheduled is

$$\sum_{j=1}^{n} p_{0j}x_{0j} - \sum_{i=1}^{m}(T - \sum_{j=1}^{n} p_{ij}x_{ij}) = \sum_{i=0}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} - mT.$$

Since $V > 0$ at the end of the considered iteration, then the above quantity is strictly positive, and then $\sum_{i=0}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} > mT$, a contradiction to Constraint (1b).

Note also that, for each machine i, the global jobs leave a free time of at least $T - \sum_{j=1}^{n} p_{ij}x_{ij}$ in the interval $[0, T]$ (see line 5). Therefore, the local jobs that are assigned to machine i receive at least an overall amount $\sum_{j=1}^{n} p_{ij}x_{ij}$ of processing time at line 11. □

Lemma 3.2. *In the schedule produced by Algorithm 1, no job is scheduled in parallel with itself.*

Proof. Clearly, no local job will be scheduled in parallel with itself, since each such job is scheduled on a unique machine (and $p_{ij}x_{ij} \leq T$). Assume by contradiction that a global job j is scheduled on two different machines i and i' during the same time interval $[t_1, t_2]$, $t_2 > t_1$, and assume w.l.o.g. that the iteration related to machine i occurred before that of i'. This implies that in all the iterations from that of i to that of i', only job j is scheduled and that, since $t_2 > t_1$, $p_{0j} > T$. Since $x_{0j} = 1$ this is a contradiction to Constraint (1d) . □

Theorem 3.3. *Given a feasible solution* $(\mathbf{x}, T)$ *to* (IP-1)*, Algorithm 1 produces a valid schedule in the interval* $[0, T]$*.*

Proof. The statement follows from Lemmas 3.1 and 3.2 and by observing that all the jobs are scheduled in the interval $[0, T]$ (see lines 6–7 and 11 of Algorithm 1). □

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 3.4. *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* $2m - 2$*.*

One way to read Proposition 3.4, when combined with Theorem 3.3, is that beyond the first $m - 1$ migrations, additional migrations will substantially not help any algorithm for the semi-partitioned scheduling problem (since any such algorithm implicitly constructs feasible solutions to (IP-1)). The fact that migrations are sometimes necessary to reach optimality is testified by Example 2.1 (and Example 5.1, discussed later, exhibits an even larger gap between migratory and non-migratory schedules).

4. Hierarchical scheduling

To move from the semi-partitioned setting to the more general hierarchical setting, we generalize the integer program (IP-1). The following ILP expresses necessary conditions on the minimum makespan T and an optimal assignment $\mathbf{x}$ in the hierarchical scheduling problem.

$$
\begin{aligned}
&\min T && && \text{(IP-2)}\\
&\quad \sum_{\alpha \in \mathcal{A}} x_{\alpha j} = 1 && \text{for } j \in J && \text{(2a)}\\
&\quad \sum_{j \in J} \sum_{\beta \subseteq \alpha} p_{\beta j} x_{\beta j} \leq |\alpha| T && \text{for } \alpha \in \mathcal{A} && \text{(2b)}\\
&\quad p_{\alpha j} x_{\alpha j} \leq T && \text{for } \alpha \in \mathcal{A},\ j \in J && \text{(2c)}\\
&\quad x_{\alpha j} \in \{0, 1\} && \text{for } \alpha \in \mathcal{A},\ j \in J. && \text{(2d)}
\end{aligned}
$$

In (IP-2), there is a binary variable $x_{\alpha j}$ for every admissible set $\alpha \in \mathcal{A}$ and every job $j \in J$. This generalizes (IP-1) where we only had singleton ($\alpha = \{i\}$) and global ($\alpha = M$, denoted by the special index 0 in (IP-1)) admissible sets. Constraint (2a) generalizes (1a), enforcing the choice of a unique affinity mask for each job. The load constraints (2b) generalize constraints (1b)-(1c), which are obtained as special cases when $\alpha = M$ or $\alpha = \{i\}$ respectively. Finally, constraints (1d) are generalized by (2d), which exclude a job being assigned to a set of machines α if its length on α is excessive.

We observe that (IP-2) uses $|\mathcal{A}| \cdot n \leq 2mn$ binary variables and hence an optimal integer solution could be derived exhaustively in time $O^*(2^{2nm}) = O^*(4^{nm})$ (the O^* notation suppressing factors that are polynomial in the input size). A more computationally feasible approach, whose details are deferred to Section 5, is to solve a linear relaxation of the IP and round it. Yet another possibility is to leverage the block structure of the ILP and use a specialized algorithm for n-fold ILPs [12], which would yield a runtime of $O^*(4^{m^2 \log(mT)})$. For the remainder of this section, we assume that a feasible integer solution to (IP-2) has already been constructed.

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 $\sum_{i \in \alpha} \textsc{load}[i, \alpha] = \sum_{j=1}^{n} p_{\alpha j} x_{\alpha j}$.[2] The second phase (Algorithm 3) proceeds top-down (i.e., from the largest set down to the smallest ones) and, *for each set*

[2] An alternative approach could be to modify (IP-2) by adding additional fractional vari-

$\alpha \in \mathcal{A}$ and each job j such that $x_{\alpha j} = 1$ assigns job j to suitable time slots in machines in α.

The crucial observation is that the first phase computes LOAD in such a way that the second phase is able to 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 $\text{LOAD}[i, \beta] > 0$ and $\text{LOAD}[i, \alpha] > 0$, 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 $t_{i\alpha} = t + \text{LOAD}[i, \alpha] \pmod{T}$; the algorithm first schedules the jobs assigned to β in machine i, in the interval $[t_{i\alpha}, t_{i\beta}]$, where $t_{i\beta} = t_{i\alpha} + \text{LOAD}[i, \beta] \pmod{T}$. 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 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 job is scheduled in parallel with itself and that no machine has more than one job scheduled in the same time interval.

Before describing the algorithm more formally, we discuss an example.

Example 4.1. Let us consider an instance with seven jobs and four machines, and a solution of (IP-2) with $T = 4$, $x_{M1} = x_{\{1,2\}2} = x_{\{3,4\}3} = x_{14} = x_{25} = x_{63} = x_{74} = 1$, and any other variable equal to 0, that is

- job 1 can be scheduled and migrated on any machine in M,
- jobs 2 and 3 can be scheduled and migrated on two machines, namely machines $\{1, 2\}$ and $\{3, 4\}$, respectively, and
- jobs 4—7 can only be scheduled on a single machine, namely 1, 2, 3, and 4, respectively.

Processing time of jobs are equal to $p_{M1} = 4$, $p_{\{1,2\}2} = p_{\{3,4\}3} = 4$, $p_{14} = p_{25} = p_{63} = p_{74} = 1$ (we omit the processing time corresponding to variables equal to 0, as they are irrelevant).

The first phase proceeds bottom-up and first reserves one unit of time to each machine, corresponding to the processing time of jobs 4—7, that is

- $\text{LOAD}[1, \{1\}] = \text{LOAD}[2, \{2\}] = \text{LOAD}[3, \{3\}] = \text{LOAD}[4, \{4\}] = 1.$

ables 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 4.6); moreover, the method is combinatorial and avoids the complication of a larger number of variables.

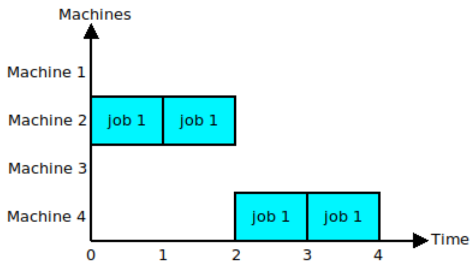
Then, it reserves four units of time for affinity masks of size two (jobs 2–3):

- $\text{LOAD}[1, \{1, 2\}] = 3$, $\text{LOAD}[2, \{1, 2\}] = 1$ (for job 2),
- $\text{LOAD}[3, \{3, 4\}] = 3$, $\text{LOAD}[4, \{3, 4\}] = 1$ (for job 3),

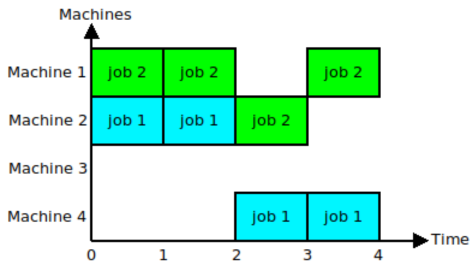
and finally it reserves four units of time for the only job with affinity mask M (job 1),

- $\text{LOAD}[1, M] = 0$,
- $\text{LOAD}[2, M] = 2$,
- $\text{LOAD}[3, M] = 0$,
- $\text{LOAD}[4, M] = 2$.

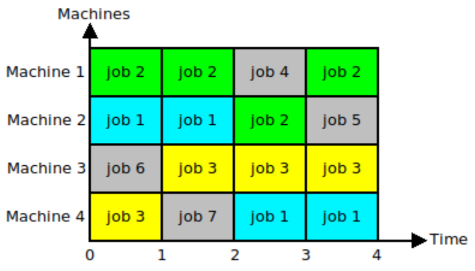
Note that, for each affinity mask $\beta \in \mathcal{A}$ there exists at most one machine $i \in \beta$ such that $\text{LOAD}[i, \beta] > 0$ and $\text{LOAD}[i, \alpha] > 0$, for some $\alpha \supset \beta$. For example, for $\beta = \{1, 2\}$, only machine 2 is loaded for $\alpha = M$.



(a) Schedule of jobs assigned to affinity mask M (i.e. job 1).



(b) Schedule of jobs assigned to affinity masks M and $\{1, 2\}$ (i.e. job 2).



(c) Final schedule.

Figure 3: Schedule of Example 4.1.

The second phase proceeds top-down and schedules:

- job 1 on machines 2 and 4,
- job 2 on machines 2 and 1,

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

```
1  LOAD$[i, \alpha] = 0$, for each $\alpha \in \mathcal{A}$ and $i \in \alpha$;
2  TOT-LOAD$[i, \alpha] = 0$, $\alpha \in \mathcal{A}$ and $i \in \alpha$;
3  MARK$[\alpha] =$ false for each $\alpha \in \mathcal{A}$;
4  while $\exists \alpha \in \mathcal{A}$ such that $\neg$MARK$[\alpha]$ do
5      Let $\alpha$ such that $\neg$MARK$[\alpha]$ and (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 $\beta$ be the maximal set $\beta \subset \alpha$ such that $i \in \beta$;
9          (if no such $\beta$ exists, set $\beta = \emptyset$ and TOT-LOAD$[i, \emptyset] = 0$);
10         LOAD$[i, \alpha] \leftarrow \min\{V, T -$ TOT-LOAD$[i, \beta]\}$;
11         TOT-LOAD$[i, \alpha] \leftarrow$ TOT-LOAD$[i, \beta] +$ LOAD$[i, \alpha]$;
12         $V \leftarrow V -$ LOAD$[i, \alpha]$;
13     end
14     MARK$[\alpha] \leftarrow$ true;
15 end
```

- job 3 on machines 4 and 3,
- jobs 4—7 on machines 1—4 respectively.

See Figure 3 for an illustration. As an example, let us consider machine 2.

- Jobs assigned to affinity mask M (i.e. job 1) are scheduled in the time interval $[0, 2]$, with $t_{2M} = 2$,
- jobs assigned to affinity mask $\{1, 2\}$ (i.e. job 2) are scheduled in the time interval $[t_{2M}, t_{2\{1,2\}}] = [t_{2M}, t_{2M} + \textsc{load}[2, \{1, 2\} \pmod T)] = [2, 3]$,
- and jobs assigned to affinity mask $\{2\}$ (i.e. job 5) are scheduled in interval $[t_{2\{1,2\}}, t_{2\{2\}}] = [t_{2\{1,2\}}, t_{2\{2\}} + \textsc{load}[2, \{2\} \pmod T)] = [3, 4]$.

The remaining processing time of job 2 is scheduled on machine 1, in the time interval $[t_{2\{1,2\}}, t_{2\{1,2\}} + \textsc{load}[1, \{1, 2\}] \pmod T)] = [3, 3 + 3 \pmod 4)]$, that is in intervals $[3, 4]$ and $[0, 2]$. This concludes the example.

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 TOT-LOAD$[i, \alpha]$, which stores the cumulative load of machine i due to all sets $\beta \subseteq \alpha$ (line 2). Variable 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 4–14 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 MARK$[\alpha] =$ `false` and, for each $\beta \subset \alpha$, MARK$[\beta] =$ `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 7–13 iterates for each machine $i \in \alpha$ in ascending order and, in order to compute 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 $\textsc{load}[i,\alpha]$ 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 $\textsc{tot-load}[i,\beta]$. Then, we can schedule at most $T - \textsc{tot-load}[i,\beta]$ 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 $T - \textsc{tot-load}[i,\beta]$, then we assign the entire volume to i and set $\textsc{load}[i,\alpha] = V$, otherwise, we exploit all the possible empty space and set $\textsc{load}[i,\alpha] = T - \textsc{tot-load}[i,\beta]$. Next, the algorithm computes the value of $\textsc{tot-load}[i,\alpha]$ by adding $\textsc{load}[i,\alpha]$ to $\textsc{tot-load}[i,\beta]$ (line 11). Note that, $\textsc{tot-load}[i,\alpha] = \sum_{\beta\subseteq\alpha : i\in\beta} \textsc{load}[i,\beta]$ and, eventually, $\sum_{i\in\alpha} \textsc{tot-load}[i,\alpha] = \sum_{\beta\subseteq\alpha}\sum_{i\in\beta} \textsc{load}[i,\beta]$. Finally, variables V and $\textsc{mark}[\alpha]$ are updated at lines 12 and 14. 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 $\textsc{load}[i,\alpha]$, for all $i \in \alpha$.

Lemma 4.1. i) *For every $\alpha \in \mathcal{A}$ and $i \in \alpha$, $\textsc{tot-load}[i,\alpha] \le T$ at the end of Algorithm 2.*
ii) *Whenever line 14 of Algorithm 2 is executed, $V = 0$.*

Proof. i) From lines 10 and 11 of the algorithm we get $\textsc{tot-load}[i,\alpha] \le \textsc{tot-load}[i,\beta] + T - \textsc{tot-load}[i,\beta] = T$. *ii)* Let α be the first set for which the statement does not hold. That is, $V > 0$ at the end of line 13 of Algorithm 2 of the iteration related to set α, while $V = 0$ at the end of the iteration related to each $\beta \subset \alpha$.

For each $\beta \subset \alpha$, we have that

$$\sum_{i\in\beta} \textsc{load}[i,\beta] = \sum_{j=1}^{n} p_{\beta j} x_{\beta j},$$

since $V = \sum_{j=1}^{n} p_{\beta j} x_{\beta j} - \sum_{i\in\beta} \textsc{load}[i,\beta] = 0$. By definition of $\textsc{tot-load}$, for each $\beta \subset \alpha$, $\sum_{i\in\beta} \textsc{tot-load}[i,\beta] = \sum_{\gamma\subseteq\beta}\sum_{i\in\gamma} \textsc{load}[i,\gamma]$ and then,

$$\sum_{i\in\beta} \textsc{tot-load}[i,\beta] = \sum_{\gamma\subseteq\beta}\sum_{j=1}^{n} p_{\gamma j} x_{\gamma j}. \tag{3}$$

For each $i \in \alpha$, let β_i be the maximal set $\beta_i \subset \alpha$ such that $i \in \beta_i$ (see line 8). Since for α the statement does not hold, then, for each $i \in \alpha$, $\textsc{load}[i,\alpha] = \min\{V, T - \textsc{tot-load}[i,\beta_i]\} = T - \textsc{tot-load}[i,\beta_i]$ (see line 10). It follows that at line 14 of the iteration related to set α:

$$\begin{aligned} V &= \sum_{j=1}^{n} p_{\alpha j} x_{\alpha j} - \sum_{i\in\alpha} \textsc{load}[i,\alpha] \\ &= \sum_{j=1}^{n} p_{\alpha j} x_{\alpha j} - \sum_{i\in\alpha} T + \sum_{i\in\alpha} \textsc{tot-load}[i,\beta_i]. \end{aligned}$$

The term $\sum_{i\in\alpha} T$ is equal to $|\alpha|T$. Since $\mathcal{A}$ is laminar, for each $i, i' \in \alpha$ either $\beta_i = \beta_{i'}$ or $\beta_i \cap \beta_{i'} = \emptyset$, and then

$$\begin{aligned}\sum_{i\in\alpha} \text{TOT-LOAD}[i,\beta_i] &= \sum_{\substack{\beta\subset\alpha \\ \beta \text{ is maximal}}} \sum_{i\in\beta} \text{TOT-LOAD}[i,\beta] \\ &= \sum_{\substack{\beta\subset\alpha \\ \beta \text{ is maximal}}} \sum_{\gamma\subseteq\beta} \sum_{j=1}^{n} p_{\gamma j} x_{\gamma j} \\ &= \sum_{\gamma\subset\alpha} \sum_{j=1}^{n} p_{\gamma j} x_{\gamma j},\end{aligned}$$

where the last two equalities follow from (3) and from the fact that $\mathcal{A}$ is laminar, respectively. Therefore,

$$\begin{aligned}V &= \sum_{j=1}^{n} p_{\alpha j} x_{\alpha j} - |\alpha|T + \sum_{\gamma\subset\alpha} \sum_{j=1}^{n} p_{\gamma j} x_{\gamma j} \\ &= \sum_{\gamma\subseteq\alpha} \sum_{j=1}^{n} p_{\gamma j} x_{\gamma j} - |\alpha|T.\end{aligned}$$

Since $V > 0$, then $\sum_{\gamma\subseteq\alpha} \sum_{j=1}^{n} p_{\gamma j} x_{\gamma j} > |\alpha|T$, a contradiction to Constraint (2b). □

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

Proof. By contradiction, let us assume that there exist two machines i and i', $i < i'$, such that LOAD$[i,\beta] > 0$, LOAD$[i,\alpha] > 0$, LOAD$[i',\beta] > 0$, and LOAD$[i',\alpha'] > 0$, for some $\alpha, \alpha' \in \mathcal{A}$ such that $\beta \subset \alpha, \alpha'$. Let us consider line 10 of Algorithm 2 at the iteration related to set β and machine i and let γ be the maximal set $\gamma \subset \beta$ such that $i \in \gamma$, that is LOAD$[i,\beta] = \min\{V, T - \text{TOT-LOAD}[i,\gamma]\}$.

- If $\min\{V, T - \text{TOT-LOAD}[i,\gamma]\} = V$, then LOAD$[i,\beta] = V$ and at the end of the iteration the algorithm sets $V = 0$ (line 12). Therefore, for each machine $i'' \in \beta$, $i'' > i$, the instruction at line 10 sets LOAD$[i'',\beta] = V = 0$. Since $i' > i$, then LOAD$[i',\beta] = 0$, a contradiction to LOAD$[i',\beta] > 0$.

- If $\min\{V, T-\text{TOT-LOAD}[i,\gamma]\} = T-\text{TOT-LOAD}[i,\gamma]$, then LOAD$[i,\beta] = T-$ TOT-LOAD$[i,\gamma]$ and the algorithm sets TOT-LOAD$[i,\beta] =$ TOT-LOAD$[i,\gamma]+$ LOAD$[i,\beta] = T$ at line 11. Therefore, for each α such that $\beta \subset \alpha$, TOT-LOAD$[i,\alpha] =$ TOT-LOAD$[i,\beta] = T$ and LOAD$[i,\alpha] = \min\{V, 0\} = 0$, a contradiction to LOAD$[i,\alpha] > 0$. □

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

```
1  MARK$[\alpha]$ $\leftarrow$ false for each $\alpha \in \mathcal{A}$;
2  while $\exists \beta \in \mathcal{A}$ such that $\neg$MARK$[\beta]$ do
3      Let $\beta$ such that $\neg$MARK$[\beta]$ and (MARK$[\alpha]$, for each $\alpha$ such that $\beta \subset \alpha$);
4      if $\exists i \in \beta$ such that LOAD$[i, \beta] > 0$ and LOAD$[i, \alpha] > 0$, for some set $\alpha \in \mathcal{A}$
        such that $\beta \subset \alpha$ then
5          Let $\alpha$ be the minimal set such that $\beta \subset \alpha$ and 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     end
12     foreach $k \in \beta$ in any order starting from $\ell$ do
13         Assign 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 + $LOAD$[k, \beta] \pmod{T}]$;
14         $t_\beta \leftarrow t_\beta + $LOAD$[k, \beta] \pmod{T}$;
15         $t_{k\beta} \leftarrow t_\beta$;
16     end
17     MARK$[\beta]$ $\leftarrow$ true;
18 end
```

The pseudo-code of the second phase is given in Algorithm 3. As in Algorithm 2, variable 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 2–18). 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 4.2, there exists at most one machine $i \in \beta$ such that LOAD$[i, \beta] > 0$ and 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 LOAD$[k, \beta]$ units of time of jobs assigned to β to machine k, in the interval $[t_\beta, t_\beta + $LOAD$[k, \beta] \pmod{T}]$ (line 13);[3] and we update t_β and $t_{k\beta}$ by adding LOAD$[k, \beta] \pmod{T}$ (lines 14–15). 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

[3]As for global jobs in the semi-partitioned case, the jobs here are assigned in an arbitrary order in the reserved interval. Again, we can have that $t_\beta + $LOAD$[k, \beta] \pmod{T} < t_\beta$, in this case the jobs assigned to machine k are first scheduled in the interval $[0, t_\beta + $LOAD$[k, \beta] \pmod{T}]$ and then in the interval $[t, t_\beta]$ and possibly a job is preempted at time $t_\beta + $LOAD$[k, \beta] \pmod{T}$.

chosen as the smallest machine in β and t_β is initialized to 0 (lines 9–10).

Lemma 4.3. *In the schedule produced by Algorithm 3, all jobs receive the required amount of processing time.*

Proof. We show that for each α all the jobs j such that $x_{\alpha j} = 1$ receive the required amount of processing time, i.e., $\sum_{j=1}^{n} p_{\alpha j} x_{\alpha j}$. For each $i \in \alpha$, Algorithm 3 assigns $\textsc{load}[i, \alpha]$ units of time to machine i and therefore assigns $\sum_{i \in \alpha} \textsc{load}[i, \alpha]$ overall time to jobs j such that $x_{\alpha j} = 1$. By Lemma 4.1.ii, $\sum_{i \in \alpha} \textsc{load}[i, \alpha] = \sum_{j=1}^{n} p_{\alpha j} x_{\alpha j}$. □

Lemma 4.4. *In the schedule produced by Algorithm 3, no job is scheduled in parallel with itself.*

Proof. The proof is similar to that of Lemma 3.2. Assume by contradiction that a job j such that $x_{\beta j} = 1$ is scheduled on two different machines $i, i' \in \beta$ during the same time interval $[t_1, t_2]$, $t_2 > t_1$, and assume w.l.o.g. that j is scheduled first on machine i and then on machine i'. Then, in all the iterations of the loop at lines 12–16 of Algorithm 3, from that of i to that of i', only job j is scheduled and thus, since $t_2 > t_1$, $p_{\beta j} > T$, a contradiction. □

Lemma 4.5. *In the schedule produced by Algorithm 3, no machine processes more than one job in the same time interval.*

Proof. By contradiction, let us consider the first iteration of the loop at lines 12–16 in which a machine i that is already scheduling a job j in some time interval is assigned another job j' in the same interval.

Let us consider the set β such that $x_{\beta j'} = 1$. We have that all the jobs j'' such that $x_{\gamma j''} = 1$ for any $\gamma \subset \beta$ are not yet scheduled in the considered iteration, therefore $x_{\alpha j} = 1$ and $\textsc{load}[i, \alpha] > 0$, for some α such that $\beta \subset \alpha$.

By Lemma 4.2, i is the only machine such that $\textsc{load}[i, \beta] > 0$ and $\textsc{load}[i, \alpha] > 0$, then, i is the machine ℓ selected at line 7 of Algorithm 3. Let $\alpha = \alpha_0 \supset \alpha_1 \supset \alpha_2 \ldots \supset \alpha_L = \beta$ be all the sets in $\mathcal{A}$ such that $\beta \subseteq \alpha_p \subseteq \alpha$. We recall that $t_{i\alpha_p}$ is the last time instant in which the algorithm schedules a job assigned to α_p on machine i and that the value of t_β after line 6 of the algorithm is executed, but before line 14 is executed, is $\bar{t}_\beta = t_{\alpha_{L-1}} = t_{i\alpha} + \sum_{l=1}^{L-1} \textsc{load}[i, \alpha_l] \pmod{T}$. Let t_x be the first time instant in which the algorithm schedules a job assigned to α on machine i, that is, jobs assigned to α are scheduled either in the interval $[t_x, t_{i\alpha}]$, if $t_x < t_{i\alpha}$, or in the intervals $[t_x, T]$ and $[0, t_{i\alpha}]$, otherwise. In the former case, to have that machine i processes a job in α and a job in β in the same time interval, we must have that $\bar{t}_\beta + \textsc{load}[i, \beta] > t_x + T$, that is $t_{i\alpha} + \sum_{l=1}^{L} \textsc{load}[i, \alpha_l] > t_x + T$. Since $t_{i\alpha} - t_x = \textsc{load}[i, \alpha]$, this implies that $\sum_{l=0}^{L} \textsc{load}[i, \alpha_l] > T$, a contradiction to Lemma 4.1.i. In the latter case, we must have that $\bar{t}_\beta + \textsc{load}[i, \beta] > t_x$, where $\bar{t}_\beta = t_{i\alpha} + \sum_{l=1}^{L-1} \textsc{load}[i, \alpha_l]$, that is $t_{i\alpha} + \sum_{l=1}^{L} \textsc{load}[i, \alpha_l] > t_x$. Since $\textsc{load}[i, \alpha] = T - t_x + t_{i\alpha}$, we obtain again the contradiction $\sum_{l=0}^{L} \textsc{load}[i, \alpha_l] > T$. □

Theorem 4.6. *Given a feasible solution* $(\mathbf{x}, T)$ *to* (IP-2)*, Algorithms* 2 *and* 3 *produce a valid schedule in the interval* $[0, T]$.

Proof. The statement follows from Lemmas 4.3–4.5. □

Before discussing how to round the ILP, we state a bound on the number of migrations introduced by the scheduling algorithm.

Proposition 4.7. *The number of job migrations in the schedule produced by Algorithm* 3 *is at most* $O(m^2)$.

Proof. Recall that in the top-down scheduling phase, we schedule the work for a set β on some machine $k \in \beta$ until we have consumed the allowed budget $\textsc{load}[k, \beta]$. After that budget is consumed, machine k is never used again for work with affinity mask β (of course, it could still be used by work with other affinity masks containing machine k). Thus, for each set $\beta \in \mathcal{A}$ the algorithm introduces at most $|\beta|$ migrations and the total number is bounded by $\sum_{\beta \in \mathcal{A}} |\beta| \le m \cdot \sum_{\beta \in \mathcal{A}} 1 \le 2m^2$, where the last inequality follows from $\mathcal{A}$ being a laminar family (see for example [26, Theorem 3.5]). □

The results of the next section imply that if we are satisfied with a 2-approximate solution, then migrations can be avoided completely. We come back to this point in Section 6.

5. A 2-approximation for the set-monotone scheduling problem

In this section, we describe a 2-approximation algorithm for the general set-monotone scheduling problem (which includes the hierarchical and semi-partitioned scheduling problems as special cases). The idea is to relate an instance $I = (J, M, \mathcal{A}, p)$ of the set-monotone scheduling problem to an instance $I_{pu} = (J, M, p')$ of the *preemptive* unrelated machines problem $R|\text{preempt}|C_{\max}$ [7, 16], and then to round I_{pu} to an nonpreemptive, nonmigratory solution which is valid for I. Namely, given an instance $I = (J, M, \mathcal{A}, p)$, we define

$$p'_{ij} := \min_{\alpha \in \mathcal{A} \colon \alpha \ni i} p_{\alpha j} \qquad \text{for all } i \in M, j \in J,$$

and consider the preemptive unrelated machines instance $I_{pu} = (J, M, p')$.

Lemma 5.1. *If* I *has a valid schedule of makespan* T^**, then* I_{pu} *has a valid preemptive schedule of makespan* T^* *such that each job* $j \in J$ *is only processed on machines* i *such that* $p'_{ij} \le T^*$.

Proof. Let S be a valid schedule for I with makespan T^*. In S, no job is assigned to an affinity mask $\alpha \in \mathcal{A}$ such that $p_{\alpha j} > T^*$, otherwise the makespan of S could not be T^*. Hence, in S, if a job j is assigned an affinity mask α, then $p_{\alpha j} \le T^*$. Now let $\alpha(j)$ denote the unique mask which a job j is assigned in S, and construct a fractional assignment $\mathbf{x}'$ as follows:

$$x'_{ij} = \begin{cases} 0 & \text{if } i \notin \alpha(j) \\ \frac{t^S_{ij}}{p_{\alpha j}} & \text{if } i \in \alpha(j), \end{cases}$$

where t_{ij}^S is the overall time job j spends on machine i in schedule S.

We claim that $\mathbf{x}'$ certifies the existence of a preemptive schedule for I_{pu} with makespan at most T^*. Notice that $p'_{ij} \leq p_{\alpha(j)j}$, by definition of p'_{ij}. The following three inequalities hold:

$$\sum_{i \in M} x'_{ij} = \frac{\sum_{i \in \alpha(j)} t_{ij}^S}{p_{\alpha(j)j}} = \frac{p_{\alpha(j)j}}{p_{\alpha(j)j}} = 1, \text{ for all } j \in J \tag{4}$$

$$\sum_{i \in M} p'_{ij} x'_{ij} = \sum_{i \in \alpha(j)} \frac{p'_{ij}}{p_{\alpha(j)j}} \cdot t_{ij}^S \leq \sum_{i \in \alpha(j)} t_{ij}^S \leq T^*, \text{ for all } j \in J \tag{5}$$

$$\sum_{j \in J} p'_{ij} x'_{ij} = \sum_{j \in J} \frac{p'_{ij}}{p_{\alpha(j)j}} \cdot t_{ij}^S \leq \sum_{j \in J} t_{ij}^S \leq T^*, \text{ for all } i \in M. \tag{6}$$

Together, these inequalities imply that the standard linear program for the preemptive unrelated machine instance I_{pu} is satisfied with makespan T^*, hence a preemptive schedule exists for I_{pu} with makespan T^* [7, 16]. Moreover, by construction, $x'_{ij} = 0$ whenever $p'_{ij} > T^*$, that is, in this schedule no job j is processed on a machine i for which $p'_{ij} > T^*$. □

Theorem 5.2. *The set-monotone scheduling problem admits a polynomial-time 2-approximation algorithm.*

Proof. Consider an instance $I = (J, M, \mathcal{A}, p)$ of the set-monotone scheduling problem and let T^* be its minimum makespan. By Lemma 5.1, the preemptive unrelated machines instance I_{pu} has a valid preemptive schedule of makespan T^* where each job $j \in J$ is only processed on machines i such that $p'_{ij} \leq T^*$.

The idea is now to invoke an existing LP-based algorithm for the unrelated machine scheduling problem and run it on I_{pu}. The classic rounding algorithm by Lenstra, Shmoys and Tardos [17, 27] (see also [7, Theorem 2.1]) constructs an integral assignment $\bar{\mathbf{x}}$ for I_{pu} with makespan

$$T' \leq \sum_{j \in J} p'_{ij} x'_{ij} + \max\{p'_{ij} : x'_{ij} > 0\} \leq 2T^*.$$

Such an assignment $\bar{\mathbf{x}}$ can be interpreted as an affinity mask assignment for the set-monotone scheduling instance I by extending it with 0 values on all sets α with $|\alpha| > 1$. By construction, the assignment $\bar{\mathbf{x}}$ assigns each job j to a singleton machine set, i.e. to a set of the form $\{i\}$, for some i depending on j. From this assignment, a schedule of makespan $T \leq 2T^*$ can be trivially obtained by scheduling the jobs assigned to each machine $i \in M$ in any order. □

We remark that the lower bound technique used in the above proof, which lower bounds the cost of the set-monotone scheduling I in terms of that of the preemptive unrelated instance I_{pu}, would *not* be valid for the nonpreemptive unrelated problem: indeed, in Example 2.1, the original instance I of the semi-partitioned problem has an optimal makespan of 2, while the associated nonpreemptive unrelated machines 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 5.1. (See also Figure 4.) 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 $\mathrm{opt}(I) = n - 1$: 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 $2n - 3$.

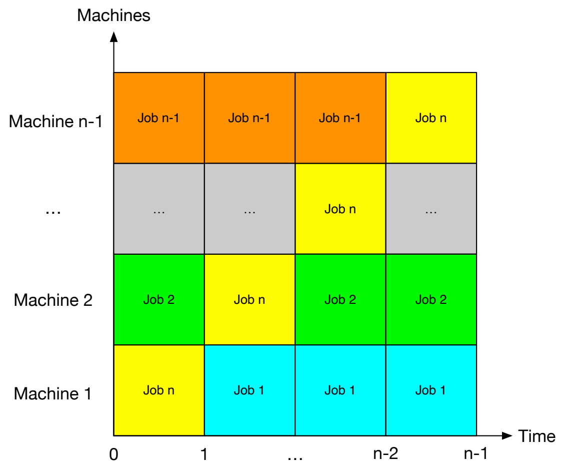


Figure 4: Optimal schedule for the instance of Example 5.1.

6. Migratory vs. non-migratory schedules

The proof of Theorem 5.2 allows us to conclude something about the relative power of job migrations in our model. Note that by disallowing migrations in our model, we obtain exactly the unrelated parallel machines scheduling problem, $R||C_{\max}$. We have already seen with Example 2.1 that migrations are useful in our model: they allow us to construct shorter schedules. Nevertheless, if we are satisfied with a 2-approximation of the makespan, then migrations can be eliminated.

Corollary 6.1. *Let S be a (possibly migratory) schedule of length T^* for the hierarchical scheduling problem. Then there is a non-migratory schedule S' of length at most $2T^*$.*

Proof. Let $(\mathbf{x}^*, T^*)$ be an integral solution to (IP-2). Such a solution exists by hypothesis, as otherwise there would be no schedule of length T^*. Exactly as in the proof of Theorem 5.2, we can therefore construct an assignment $\bar{\mathbf{x}}$ that is nonzero only on singleton machine sets and has makespan at most $2T^*$. This yields the desired non-migratory schedule S'. □

Note that the above gap of 2 is tight, due to Example 5.1. Note also that while resorting to migrations might conceivably reduce the approximation factor of an algorithm below 2, it is unlikely to reduce it below $3/2$ due to the hardness of approximation of the hierarchical scheduling problem (recall Proposition 2.1).

7. 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. In the first model, each machine has a separate memory capacity. In the second model, each cluster of the hierarchical architecture has a certain memory capacity, which is shared among all machines of the cluster.

7.1. First memory model (Model 1)

In this first model, 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 will constrain ILP (IP-2) by adding the memory capacity constraints

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

Let us slightly reformulate the original constraints of the ILP (IP-2). By fixing the value of the target makespan T, which can be found by a standard binary search technique [9], they can be rewritten as:

(IP-3)

$$\sum_{\alpha\in\mathcal{A}} x_{\alpha j} = 1 \qquad \text{for } j\in J \tag{8a}$$

$$\sum_{j\in J}\sum_{\beta\subseteq\alpha} p_{\beta j}x_{\beta j} \le |\alpha|T \qquad \text{for } \alpha\in\mathcal{A} \tag{8b}$$

$$x_{\alpha j}\in\{0,1\} \qquad \text{for } \alpha\in\mathcal{A}, j\in J, \tag{8c}$$

$$x_{\alpha j} = 0 \qquad \text{for } (\alpha,j)\notin R, \tag{8d}$$

where $R = \{(\alpha, j) \in \mathcal{A} \times J : p_{\alpha j} \leq T\}$. That is, we can implicitly deal with Constraints (2c) by observing that they are satisfied by a 0-1 solution if and only if $p_{\alpha j} \leq T$ whenever $x_{\alpha j} = 1$. Therefore, we can simply set to zero all variables $x_{\alpha j}$ such that $p_{\alpha j} > T$. Finally, 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 the revised ILP, we show that a 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 immediately by repeated application of the next lemma, which allows to "push down" the fractional weights towards the singleton sets of the laminar family.

Lemma 7.1. *Let $\eta \in \mathcal{A}$ be a non-singleton set. If $\mathbf{x}$ is a fractional solution to the LP relaxation of* (IP-3)*, then there exists another fractional 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

$$\begin{aligned}
&\sum_{\alpha \in \mathcal{A}} x_{\alpha j} = 1 && \text{for } j \in J && \text{(9a)}\\
&\text{slack}(\alpha, \mathbf{x}) \geq 0 && \text{for } \alpha \in \mathcal{A} && \text{(9b)}\\
&x_{\alpha j} \geq 0 && \text{for } \alpha \in \mathcal{A}, j \in J && \text{(9c)}\\
&x_{\alpha j} = 0 && \text{for } (\alpha, j) \,:\, p_{\alpha j} > T. && \text{(9d)}
\end{aligned}$$

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 nonnegative slack in $\mathbf{x}$, we have

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

which is equivalent to

$$\sum_{j \in J} p_{\eta j} x_{\eta j} \leq \text{slack}(\beta_1, \mathbf{x}) + \ldots + \text{slack}(\beta_q, \mathbf{x}). \tag{10}$$

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

$$x'_{\beta j} = x_{\beta j} + \frac{\text{slack}(\beta, \mathbf{x})}{\text{slack}(\beta_1, \mathbf{x}) + \ldots + \text{slack}(\beta_q, \mathbf{x})} \cdot x_{\eta j} \tag{11}$$

for $\beta = \beta_1, \ldots, \beta_q$. We claim that $\mathbf{x}'$ is valid for the LP. To see that (9a) is satisfied by the new solution, note that

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

To see that (9b) 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 (11) across jobs,

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

where for the first inequality we used the monotonicity of the processing times, and for the second inequality we used (10). Therefore,

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

Now, to round the fractional relaxation of the revised ILP, we apply the *iterative rounding* approach [11, 15, 20], allowing us to prove the following theorem.

Theorem 7.2. *Whenever ILP (IP-3) has a fractional solution $(\mathbf{x}, T)$ that also satisfies constraints (7), it is possible to construct, in polynomial time, a valid schedule with makespan at most $3T$ such that*

$$\sum_{j \in J} \left(s_{ij} \cdot \sum_{\alpha \in \mathcal{A} \,:\, i \in \alpha} x_{\alpha j} \right) \leq 3 \cdot B_i \qquad \textit{for each } i \in M. \tag{12}$$

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

Proof. Consider a fractional solution $\mathbf{x}$ to (9a)–(9d) that also satisfies the capacity constraints (7). By applying Lemma 7.1, we can obtain a feasible fractional vector $\mathbf{z}$ that satisfies constraints (9a)–(9d) and such that $z_{\alpha j} > 0$ only if $|\alpha| = 1$. Moreover, note that

$$z_{ij} \leq \sum_{\alpha \in \mathcal{A} \,:\, i \in \alpha} x_{\alpha j} \qquad \text{for } i \in M,\ j \in J \tag{13}$$

because the fractional weight in $\mathbf{z}$ associated to a pair (i,j) can only be due to the fractional weight in $\mathbf{x}$ of pairs (α, j) such that $i \in \alpha$. Therefore,

$$\sum_{j\in J} s_{ij} z_{ij} \le \sum_{j \in J} s_{ij} \left(\sum_{\alpha \in \mathcal{A}\,:\, i \in \alpha} x_{\alpha j} \right) \le B_i, \tag{14}$$

where the second inequality follows by (7). This means that the fractional solution $\mathbf{z}$ is feasible for the following LP:

$$\begin{aligned}
&\sum_{i\in M} z_{ij} = 1 && \text{for } j \in J && \text{(15a)}\\
&\sum_{j\in J} p_{ij} z_{ij} \le T && \text{for } i \in M && \text{(15b)}\\
&\sum_{j\in J} s_{ij} z_{ij} \le B_i && \text{for } i \in M && \text{(15c)}\\
&z_{ij} \ge 0 && \text{for } i \in M, j \in J, && \text{(15d)}\\
&z_{ij} = 0 && \text{for } (i,j) \notin R, && \text{(15e)}
\end{aligned}$$

where $R = \{(i,j) \in M \times J \,:\, p_{ij} \le T \wedge s_{ij} \le b_i\}$. This linear program can be rounded using an existing iterated rounding technique, described in [20]. The algorithm of [20] constructs an integral assignment $\bar{\mathbf{x}}$ satisfying

$$\begin{aligned}
&\sum_{j\in J} p_{ij} \bar{x}_{ij} \le T + 2 \max_{j \in J:\,(i,j)\in R} p_{ij} && \text{for } i \in M && \text{(16a)}\\
&\sum_{j\in J} s_{ij} \bar{x}_{ij} \le B_i + 2 \max_{j \in J:\,(i,j)\in R} s_{ij} && \text{for } i \in M, && \text{(16b)}
\end{aligned}$$

where the factor of 2 is due to the fact that each variable z_{ij} appears with at most two nonzero coefficients in the packing Constraints (15b)-(15c) (one is p_{ij}, the other s_{ij}). Since $\max_{j:(i,j)\in R} p_{ij} \le T$ and $\max_{j:(i,j)\in R} s_{ij} \le B_i$, this proves the claim. □

7.2. Second memory model (Model 2)

In this model, each cluster of the hierarchical architecture has a memory capacity that is shared among all machines of the cluster, and larger clusters have larger capacity. The exact memory scaling is captured by some monotone function $\mu : \mathcal{A} \to \mathbb{Q}_+$.

We can assume that some "root" cluster contains all machines; if not, we can add to $\mathcal{A}$ the set M containing all machines, and set a very high value of p_{Mj} for each job j (i.e., $p_{Mj} = \infty$) to ensure that the set of feasible solutions is not affected.

Formally, our model assumes that job j requires space $s_j \le 1$ (with $s_j \in \mathbb{Q}_+$), that each cluster α except the root has memory capacity $\mu(\alpha) \ge 1$, and that the root has unbounded capacity. We require the jobs assigned to a cluster to fit the memory capacity of that cluster. Thus, this time we revise ILP (IP-2) by adding the capacity constraints

$$\sum_{j\in J} s_j x_{\alpha j} \le \mu(\alpha) \text{ for each } \alpha \in \mathcal{A}\setminus\{M\}. \tag{17}$$

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

The additional constraints affect the applicability of Lemma 7.1 to "push down" the values of the fractional variables towards the leaves of the laminar family. Moreover, the approximability bounds ensured by known rounding techniques (for example, [14, 20]) are not suitable in this case. Indeed, one cannot apply the result of Karp et al. [14] because it does not ensure that the resulting integral vector satisfies the assignment constraints exactly; while the guarantee of Marchetti-Spaccamela et al. [20] yields a large approximation factor, equal to $1 + k$, where k is the number of levels of the hierarchy.

We improve the analysis of the iterative rounding scheme from Marchetti-Spaccamela et al. [20]. We show, in Lemma 7.3 below, that the scheme satisfies the assignment constraints exactly 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 7.3. *Let I, J be nonempty finite sets, and $R \subseteq I \times J$. Consider a linear program of the form:*

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

$$\sum_{i:(i,j)\in R} z_{ij} = 1 \qquad \forall j \in J \tag{18a}$$

$$\sum_{q=(i,j)\in R} a_{lq} z_{ij} \le b_l \qquad l = 1,\dots,\theta \tag{18b}$$

$$0 \le z_{ij} \le 1 \qquad \forall (i,j) \in R, \tag{18c}$$

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 $\bar{z}_{ij} \in \{0,1\}$, for each $(i,j) \in R$, such that

$$\sum_{(i,j)\in R} c_{ij}\bar{z}_{ij} \le \sum_{(i,j)\in R} c_{ij} z^0_{ij} \tag{19a}$$

$$\sum_{i:(i,j)\in R} \bar{z}_{ij} = 1 \qquad \forall j \in J \tag{19b}$$

$$\sum_{q=(i,j)\in R} a_{lq}\bar{z}_{ij} \le (1+\rho) b_l \qquad l = 1,\dots,\theta \tag{19c}$$

ALGORITHM 4: Iterative ILP rounding

```
1  while LP has $s \geq 1$ variables do
2      Solve LP to find extreme-point solution $\mathbf{z}^h$;
3      if $\mathbf{z}^h$ has at least one integral entry then
4          fix all integral variables at their value in $\mathbf{z}^h$, remove them from LP and
             update right-hand side coefficients;
5          remove from LP all constraints that no longer involve any variable;
6      else
7          find a constraint index $l \in \{1, \dots, \theta\}$ such that
             $\sum_{q=(i,j)\in R} a_{lq}(1 - z^h_{ij}) \leq \rho \cdot b_l$, where $S = \{0,1\}^s$;
8          remove from LP the constraint corresponding to $l$;
9      end
10 end
```

Proof. Before formally proving the Lemma we present the iterative rounding procedure detailed in Algorithm 4. As in Marchetti-Spaccamela et al. [20], the idea of the algorithm is to compute, at iteration $h = 0, 1, 2, \dots$, an optimal extreme-point solution $\mathbf{z}^h$ of a linear program LP^h, with LP^0 being the initial program (LP) assumed in the Lemma. Each subsequent LP is obtained by either freezing some variables at their integer value in the current LP solution (and updating the corresponding right-hand side coefficients – Lines 4–5), or, if no variable is integral, by discarding a packing constraint $l \in \{1, \dots, \theta\}$ among those in (18b) that satisfy

$$\sum_{q=(i,j)\in R} a_{lq}(1 - z^h_{ij}) \leq \rho \cdot b_l,$$

with $S = \{0,1\}^s$. Such a packing constraint can be "safely dropped", in the sense that even if the partial integer solution is completed by setting all remaining variables to their "worst" possible value, which is 1, the constraint will eventually be violated by at most ρ times its original right-hand side b_l (Lines 7–8). Note that the assignment Constraints (18a) are never dropped, and that if z^h_{ij} is fixed at value 1, all remaining variables $z^h_{i'j}$ with $i' \neq i$ are fixed at value 0, due to the structure of the assignment constraints.

A crucial point, of course, is to guarantee that whenever the solution of LP^h has only fractional entries (and thus the **else** branch is taken in Line 6 of the algorithm), then there always exists some constraint of LP^h that can be safely dropped, i.e., the appropriate index l can always be found in Line 7 of the algorithm, ensuring progress of the iterative procedure. This fact is proven in the following auxiliary lemma.

Lemma 7.4. *Let LP^h be the linear program that is solved in iteration h of the rounding procedure, having s variables and r constraints. Let $\mathbf{z}^h$ be an extreme-point solution to this LP. If $s > r$, then $\mathbf{z}^h$ has at least one integral entry. If $s \leq r$, then there is some $l \in \{1, \dots, \theta\}$ such that* $\sum_{q=(i,j)\in R} a_{lq}(1 - z^h_{ij}) \leq$

$\rho \cdot b_l$, where S is the integer solution space for all remaining variables, i.e., $S = \{0,1\}^s$.

Proof. Assume that the subproblem in iteration h (described by LP^h) is defined as $\mathbf{A}\mathbf{z} \leq \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{r \times s}$. Note that $r = r_a + r_b$, where r_a is the number of constraints of type (18a) and r_b is the number of constraints of type (18b). We can ignore the constraints $0 \leq z_{ij} \leq 1$ because if any of them is active, the iteration immediately completes with one variable being fixed to 0 or 1 and removed from the problem.

First assume that $s > r$. Then the null space of $\mathbf{A}$ is nontrivial, so let $\mathbf{z}_0$ be a nonzero vector in the null space of $\mathbf{A}$. Since $\mathbf{z}^h$ is an extreme-point solution to LP^h, it cannot be expressed as the convex combination of two (or more) solutions to LP^h. If $\mathbf{z}^h$ does not have any integral entry, then we can find a value $\delta > 0$ such that $\mathbf{z}^h + \delta\mathbf{z}_0$ and $\mathbf{z}^h - \delta\mathbf{z}_0$ are both solutions to LP^h and, in particular, $\mathbf{z}^h$ is a convex combination of these two solutions. Therefore $\mathbf{z}^h$ must have at least one integral entry.

Now assume that $s \leq r$. For this case, we show that there always exists a constraint l of type (18b) such that $\{(\mathbf{A}\mathbf{1})_l - (\mathbf{A}\mathbf{z})_l\} \leq \rho \cdot b_l$. We show the statement by contradiction. Assume that the statement is not true; that is, for each constraint l of type (18b) it holds that

$$(\mathbf{A}\mathbf{1})_l - (\mathbf{A}\mathbf{z})_l > \rho \cdot b_l. \tag{20}$$

In each previous round, if variables were removed from the program, also constraints that had become redundant, were removed. Therefore, for all variables present in the linear program in this round, the corresponding constraint of type (18a) is also still present in the linear program (and this constraint is not present if all of its variables have been removed from the program). It follows that

$$\sum_{(i,j)\in R^h} z_{ij} = r_a, \tag{21}$$

where R^h is the index set of the variables in LP^h. Define Θ^h as the set of constraints of type (18b) present in the current linear program LP^h. Then,

$$\begin{aligned}
\rho\,(r - r_a) &= \rho\, r_b \\
&< \sum_{l \in \Theta^h} \frac{1}{b_l}((\mathbf{A1})_l - (\mathbf{Az})_l) \\
&= \sum_{l \in \Theta^h} \frac{1}{b_l} \sum_{q \in R^h} a_{lq}(1 - z_q) \\
&= \sum_{q \in R^h} (1 - z_q) \sum_{l \in \Theta^h} \frac{a_{lq}}{b_l} \\
&\le \sum_{q \in R^h} \rho\,(1 - z_q) \\
&= \rho\, s - \sum_{q \in R^h} \rho\, z_q \\
&= \rho(s - r_a).
\end{aligned}$$

The second inequality follows by the assumption on the normalized column sums of $\mathbf{A}$

The chain of inequalities implies that $\rho\,(r - r_a) < \rho\,(s - r_a)$, that is, $r < s$, which is a contradiction to being in the case that $s \le r$. Hence, we conclude that if $s \le r$, there must be a constraint l of type (18b), for which $\{(\mathbf{A1})_l - (\mathbf{Az})_l\} \le \rho \cdot b_l$. □

We now complete the proof of Lemma 7.3 showing that the obtained solution verifies the claim. Lemma 7.4 guarantees that when the extreme-point solution $\mathbf{z}^h$ has no integral entries, one of the Constraints (18b) can be dropped without violating the corresponding guarantee in (19c) in subsequent steps: even if all remaining variables are set to 1, (19c) will be satisfied for that value of l. Therefore, Algorithm 4 always finds either an integral variable – which is fixed and removed – or an appropriate constraint that is discarded (i.e., Line 7 always finds an appropriate l).

By induction, each LP^h is feasible: LP^0 has feasible solution $\mathbf{z}^0$ by assumption, and each subsequent LP is obtained by discarding constraints or by fixing some variables at their current LP value; both operations preserve feasibility, and do not increase the cost of the optimal solution; indeed, $\mathbf{z}^h$ induces a solution to LP^{h+1} with the same objective value as $\mathbf{z}^h$ in LP^h. Thus, the final vector $\bar{\mathbf{z}}$ fixed by the iterative rounding process has 0/1 components, satisfies the assignment constraints (i.e., (19b) holds), and has a cost bounded by the cost of the initial feasible solution $\mathbf{z}^0$ (i.e., (19a) holds). In general, $\bar{\mathbf{z}}$ may violate some of the packing constraints, but by the choice of the dropped constraints in Algorithm 4, no packing constraint l on $\bar{\mathbf{z}}$ will be violated by more than an additional factor $\rho \cdot b_l$, i.e., (19c) holds. This concludes the proof of Lemma 7.3. □

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

Theorem 7.5. *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*

$$\sum_{j \in J} s_j x_{\alpha j} \leq \sigma \cdot \mu(\alpha) \text{ for each } \alpha \in \mathcal{A} \setminus \{M\}, \tag{22}$$

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 7.3. In particular, we take I in Lemma 7.3 to be the admissible sets family $\mathcal{A}$, and we use the generic packing Constraints (19c) to encode Constraints (8b) and (17). Indeed, Constraints (8b) 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 (17) can be encoded as $|\mathcal{A}| - 1$ constraints with coefficients of the form

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

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 7.3 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, $|\mathcal{A}| \leq 2m$ (see, e.g., [26, 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 7.3, note that $(\beta, j) \in R$ only when $p_{\beta j} \leq T$ by construction, so if $q = (\beta, j)$,

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

where for the first inequality we also used $s_j \leq 1 \leq \mu(\beta)$, 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 7.3 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 $s_j/\mu(\beta)$, which is at most 1. Therefore, $\rho = 2 + 1/m$ is sufficient.

Finally, the integer solution $\mathbf{x}$ to (19), which can be found by applying Lemma 7.3, can be used to construct a schedule with makespan at most $(1 + \rho)T$ and satisfying (22) by feeding $\mathbf{x}$ to the algorithms presented in Sections 3 and 4. □

We remark that the $(2 + H_k)$-approximate solution guaranteed by Theorem 7.5 is (clearly) a migratory schedule, since each job gets assigned to a certain set of machines. However, if desired, one can still transform it into a nonmigratory schedule with approximation ratio $2(2 + H_k)$, by using a reasoning similar to that used in the proof of Theorem 5.2.

Acknowledgment

We thank an anonymous reviewer for pointing out references [7, 19] and their connection with the non-hierarchical setting. We also thank an anonymous reviewer for suggesting a simpler approach to the proof of Theorem 5.2. Work of the first author was carried out in association with IASI-Consiglio Nazionale delle Ricerche, Italy.

References

[1] Sanjoy Baruah. 2015. Federated Scheduling of Sporadic DAG Task Systems. In *Proc. 2015 IEEE International Parallel and Distributed Processing Symposium*. 179–186. https://doi.org/10.1109/IPDPS.2015.33

[2] Andrea Bastoni, Björn B. Brandenburg, and James H. Anderson. 2010. An Empirical Comparison of Global, Partitioned, and Clustered Multiprocessor EDF Schedulers. In *Proc. of the 31st IEEE Real-Time Systems Symposium*. IEEE, 14–24. https://doi.org/10.1109/RTSS.2010.23

[3] Andrea Bastoni, Björn B. Brandenburg, and James H. Anderson. 2011. Is Semi-Partitioned Scheduling Practical?. In *Proc. of the 23rd Euromicro Conf. on Real-Time Systems*. IEEE, 125–135. https://doi.org/10.1109/ECRTS.2011.20

[4] Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, and Harsha Vardhan Simhadri. 2011. Scheduling irregular parallel computations on hierarchical caches. In *Proc. of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures*. ACM, 355–366. https://doi.org/10.1145/1989493.1989553

[5] Vincenzo Bonifaci, Björn B. Brandenburg, Gianlorenzo D'Angelo, and Alberto Marchetti-Spaccamela. 2016. Multiprocessor Real-Time Scheduling with Hierarchical Processor Affinities. In *Proc. 28th Euromicro Conf. on Real-Time Systems*. 237–247. https://doi.org/10.1109/ECRTS.2016.24

[6] Marin Bougeret, Pierre-Francois Dutot, Denis Trystram, Klaus Jansen, and Christina Robenek. 2015. Improved approximation algorithms for scheduling parallel jobs on identical clusters. *Theoretical Computer Science* 600 (2015), 70 – 85. https://doi.org/10.1016/j.tcs.2015.07.003

[7] José R. Correa, Martin Skutella, and José Verschae. 2012. The Power of Preemption on Unrelated Machines and Applications to Scheduling Orders. *Math. Oper. Res.* 37, 2 (2012), 379–398. https://doi.org/10.1287/moor.1110.0520

[8] Celia A. Glass and Hans Kellerer. 2007. Parallel machine scheduling with job assignment restrictions. *Naval Research Logistics* 54, 3 (2007), 250–257. https://doi.org/10.1002/nav.20202

[9] Dorit S. Hochbaum and David B. Shmoys. 1987. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *J. ACM* 34, 1 (1987), 144–162. https://doi.org/10.1145/7531.7535

[10] Jing-Jang Hwang, Yuan-Chieh Chow, Frank D. Anger, and Chung-Yee Lee. 1989. Scheduling Precedence Graphs in Systems with Interprocessor Communication Times. *SIAM J. Comput.* 18, 2 (1989), 244–257. https://doi.org/10.1137/0218016

[11] Kamal Jain. 2001. A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem. *Combinatorica* 21, 1 (2001), 39–60. https://doi.org/10.1007/s004930170004

[12] Klaus Jansen, Alexandra Lassota, and Lars Rohwedder. 2020. Near-Linear Time Algorithm for n-Fold ILPs via Color Coding. *SIAM J. Discret. Math.* 34, 4 (2020), 2282–2299. https://doi.org/10.1137/19M1303873

[13] Klaus Jansen, Marten Maack, and Roberto Solis-Oba. 2020. Structural parameters for scheduling with assignment restrictions. *Theor. Comput. Sci.* 844 (2020), 154–170. https://doi.org/10.1016/j.tcs.2020.08.015

[14] Richard M. Karp, Frank Thomson Leighton, Ronald L. Rivest, Clark D. Thompson, Umesh V. Vazirani, and Vijay V. Vazirani. 1987. Global Wire Routing in Two-Dimensional Arrays. *Algorithmica* 2 (1987), 113–129. https://doi.org/10.1007/BF01840353

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

[16] Eugene L. Lawler and Jacques Labetoulle. 1978. On Preemptive Scheduling of Unrelated Parallel Processors by Linear Programming. *J. ACM* 25, 4 (1978), 612–619. https://doi.org/10.1145/322092.322101

[17] Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. 1990. Approximation Algorithms for Scheduling Unrelated Parallel Machines. *Math. Program.* 46 (1990), 259–271. https://doi.org/10.1007/BF01585745

[18] Jing Li, Jian-Jia Chen, Kunal Agrawal, Chenyang Lu, Christopher D. Gill, and Abusayeed Saifullah. 2014. Analysis of Federated and Global Scheduling for Parallel Real-Time Tasks. In *Proc. 26th Euromicro Conf. on Real-Time Systems*. 85–96. https://doi.org/10.1109/ECRTS.2014.23

[19] Jyh-Han Lin and Jeffrey Scott Vitter. 1992. ϵ-Approximations with Minimum Packing Constraint Violation (Extended Abstract). In *Proc. 24th Annual ACM Symposium on Theory of Computing*. ACM, 771–782.

[20] Alberto Marchetti-Spaccamela, Cyriel Rutten, Suzanne van der Ster, and Andreas Wiese. 2015. Assigning sporadic tasks to unrelated machines. *Math. Program.* 152, 1-2 (2015), 247–274. https://doi.org/10.1007/s10107-014-0786-9

[21] Evangelos P. Markatos and Thomas J. LeBlanc. 1994. Using Processor Affinity in Loop Scheduling on Shared-Memory Multiprocessors. *IEEE Trans. Parallel Distrib. Syst.* 5, 4 (1994), 379–400. https://doi.org/10.1109/71.273046

[22] Robert McNaughton. 1959. Scheduling with Deadlines and Loss Functions. *Management Science* 6, 1 (1959), 1–12. https://doi.org/10.1287/mnsc.6.1.1

[23] Gabriella Muratore, Ulrich M. Schwarz, and Gerhard J. Woeginger. 2010. Parallel machine scheduling with nested job assignment restrictions. *Operations Research Letters* 38, 1 (2010), 47 – 50. https://doi.org/10.1016/j.orl.2009.09.010

[24] Rolf Rabenseifner, Georg Hager, and Gabriele Jost. 2009. 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*. IEEE, 427–436. https://doi.org/10.1109/PDP.2009.43

[25] James D. Salehi, James F. Kurose, and Donald F. Towsley. 1996. The effectiveness of affinity-based scheduling in multiprocessor network protocol processing (extended version). *IEEE/ACM Trans. Netw.* 4, 4 (1996), 516–530. https://doi.org/10.1109/90.532862

[26] Alexander Schrijver. 2003. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer.

[27] David B. Shmoys and Éva Tardos. 1993. An approximation algorithm for the generalized assignment problem. *Math. Program.* 62 (1993), 461–474. https://doi.org/10.1007/BF01585178

[28] Yanyong Zhang, Hubertus Franke, José E. Moreira, and Anand Sivasubramaniam. 2003. An Integrated Approach to Parallel Scheduling Using Gang-Scheduling, Backfilling, and Migration. *IEEE Trans. Parallel Distrib. Syst.* 14, 3 (2003), 236–247. https://doi.org/10.1109/TPDS.2003.1189582

[29] Sergey Zhuravlev, Juan Carlos Saez, Sergey Blagodurov, Alexandra Fedorova, and Manuel Prieto. 2012. Survey of scheduling techniques for addressing shared resources in multicore processors. *ACM Comput. Surv.* 45, 1 (2012), 4. https://doi.org/10.1145/2379776.2379780