A Model for Minimizing Active Processor Time

Jessica Chang[⋆1], Harold N. Gabow[2], and Samir Khuller[⋆⋆3]

[1] University of Washington, Seattle WA 98195 `Email:jschang@cs.washington.edu`.
[2] University of Colorado, Boulder CO 80309 `Email:hal@cs.colorado.edu`.
[3] University of Maryland, College Park MD 20742 `Email:samir@cs.umd.edu`.

Abstract. We introduce the following elementary scheduling problem. We are given a collection of n jobs, where each job J_i has an integer length ℓ_i as well as a set T_i of time intervals in which it can be feasibly scheduled. Given a parameter B, the processor can schedule up to B jobs at a timeslot t so long as it is "active" at t. The goal is to schedule all the jobs in the fewest number of active timeslots. The machine consumes a fixed amount of energy per active timeslot, *regardless* of the number of jobs scheduled in that slot (as long as the number of jobs is non-zero). In other words, subject to ℓ_i units of each job i being scheduled in its feasible region and at each slot at most B jobs being scheduled, we are interested in minimizing the total time during which the machine is active. We present a linear time algorithm for the case where jobs are unit length and each T_i is a single interval. For general T_i, we show that the problem is NP-complete even for $B = 3$. However when $B = 2$, we show that it can be solved. In addition, we consider a version of the problem where jobs have arbitrary lengths and can be preempted at any point in time. For general B, the problem can be solved by linear programming. For $B = 2$, the problem amounts to finding a triangle-free 2-matching on a special graph. We extend the algorithm of Babenko et. al. [3] to handle our variant, and also to handle non-unit length jobs. This yields an $O(\sqrt{L}m)$ time algorithm to solve the preemptive scheduling problem for $B = 2$, where $L = \sum_i \ell_i$. We also show that for $B = 2$ and unit length jobs, the optimal non-preemptive schedule has at most 4/3 times the active time of the optimal preemptive schedule; this bound extends to several versions of the problem when jobs have arbitrary length.

1 Introduction

Power management strategies have been widely studied in the scheduling literature [1, 20, 4]. Many of the models are motivated by the energy consumption of the processor. Consider, alternatively, the energy consumed by the operation of large storage systems. Data is stored in memory which may be turned on and off [2], and each task or job needs to access a subset of data items to run. At each time step, the scheduler can perform at most B jobs. The only requirement is that the memory banks containing the required data from these jobs be turned

⋆ Supported by NSF CCF-1016509, and GRF.
⋆⋆ Supported by NSF CCF-0728839, CCF-0937865 and a Google Award.

on. The problem studied in this paper is the special case where all the data is in one memory bank. (For even special cases involving multiple memory banks, the problem often becomes NP-complete.)

We propose a simple model for measuring energy usage on a parallel machine. Instead of focusing on solution quality from the scheduler's perspective, we focus on the energy savings of "efficient" schedules.

In many applications, a job has many intervals of availability because, e.g., it interfaces with an external event like a satellite reading or a recurring broadcast. The real-time and period scheduling literatures address this. More broadly, tasks may be constrained by user availability, introducing irregularity in the feasible intervals. Our model is general enough to capture jobs of this nature.

More formally, we are given a collection of n jobs, each job J_i having integer length ℓ_i and a set T_i of time intervals with integer boundaries in which it can be feasibly scheduled. In particular, $T_i = \{I^i_k = [r^i_k, d^i_k]\}_{k=1}^{m_i}$ is a non-empty set of disjoint intervals, where $|I^i_k| = d^i_k - r^i_k$ is the size of I^i_k. Note that if $m_i = 1$, then we can think of job J_i as having a single release time and a single deadline.

For ease of notation, we may refer to J_i as job i. Additionally, time is divided into unit length timeslots and for a given parallelism parameter B, the system (or machine) can schedule up to B jobs in a single timeslot. If the machine schedules any jobs at timeslot t, we say that it is "active at t". The goal is to schedule all jobs within their respective feasible regions, while minimizing the number of slots during which the machine is active. The machine consumes a fixed amount of energy per active slot. In other words, subject to each job J_i being scheduled in its feasible region T_i, and subject to at most B jobs being scheduled at any time, we would like to minimize the total active time spent scheduling the jobs. There may be instances when there is no feasible schedule for all the jobs. However, this case is easy to check, as we show in Sec. 2. Note that for a timeslot significantly large (e.g. on the order of an hour), any overhead cost for starting a memory bank is negligible compared to the energy spent being "on" for that unit of time.

To illustrate this model in other domains, consider the following operational problem. Suppose that a ship can carry up to B cargo containers from one port to another. Jobs have delivery constraints, i.e. release times and deadlines. An optimal schedule yields the minimum number of times the ship must be sent to deliver all the packages on time. The motivating assumption is that it costs roughly the same to send the ship, regardless of load and that there is an upper bound on the load.

We could also consider this as a basic form of "batch" processing similar to the work of Ikura and Gimple [19], which presented an algorithm minimizing completion time for batch processing on a single machine in the case of *agreeable*[4] release times and deadlines. The natural extension to general release times and deadlines has an efficient algorithm as well [7]. Both of these works focus on

[4] When the ordering of jobs by release times is the same as the ordering of jobs by deadlines.

finding a feasible schedule (which can be used to minimize maximum lateness). However in our problem, in addition we wish to minimize the number of batches.

For the cases of unit length jobs or those in which jobs can be preempted at integral time points, our scheduling problem can be modeled as a bipartite matching problem in which each node on the left needs to be matched with a node on the right. Each node on the right can be matched to up to B nodes on the left, and we are interested in minimizing the number of nodes on the right that have non-zero degree in the matching selected. This problem can easily be shown to be NP-hard. For unit length jobs and single intervals T_i, we can develop a fast algorithm to obtain an optimal solution to this scheduling problem. [5]

In particular, even for $B = 2$, there is plenty of structure due to the connection with matchings in graphs. We anticipate that this structure will be useful in the design of improved approximation algorithms for $B > 2$.

Main Results:
For proof details, we refer the reader to the full version of this paper [5].

1. For the case where jobs have unit length and each T_i is one interval (i.e. $m_i = 1$), we develop an algorithm whose running time is linear. Our algorithm takes as input n jobs with release times and deadlines and outputs a non-preemptive schedule with the smallest number of active slots. The algorithm has the additional property that it schedules the maximum number of jobs. We also note that without loss of generality, time is slotted when job lengths, release times and deadlines are integral (Sec. 2). When the release times and deadlines are not integral, minimizing the number of batches can be solved optimally in polynomial time via dynamic programming. This objective is more restrictive than active time: a batch must start all its jobs at the same time and the system may work on at most one batch at any given point in time. Even so, scheduling unit length jobs with integer release times and deadlines to minimize active time is a special case of this. We extend this to the case with a budget on the number of active slots. The running time of the former algorithm was recently improved in [22].
2. We consider the generalization to arbitrary T_i, which is closely related to capacitated versions of vertex cover, k-center and facility location, all classic covering problems. In particular, for the special case where every job is feasible in exactly two timeslots, a 2-approximation is implied from the vertex cover result in [17]. The complexity of the problem depends on the value of B, since for any fixed $B \geq 3$, the problem is NP-hard. When $B = 2$ this problem can be solved optimally in $O(m\sqrt{n})$ time where m is the total number of timeslots which are feasible for some job (Sec. 3). We show that this problem is essentially equivalent to the maximum matching problem computationally. In addition, we show that this algorithm can be extended

[5] The problem can be solved in $O(n^2T^2(n+T))$ time using Dynamic Programming as was shown by Even et. al. [12], albeit the complexity of their solution is high. Their algorithm solves in the problem of stabbing a collection of horizontal intervals with the smallest number of vertical stabbers, each stabber having a bounded capacity.

to the case of non-unit length jobs where a job can be preempted at integer time points, i.e. scheduled in unit sized pieces.

3. We also consider the case when jobs have arbitrary lengths and can be preempted[6] at any time, i.e. not just at integer time points. For general B, the problem can be solved by linear programming. For $B = 2$, the problem amounts to finding a maximum triangle-free 2-matching on a special graph. Babenko et. al. [3] present an elegant algorithm that essentially shows a maximum cardinality triangle-free 2-matching can be found in the same time as a maximum cardinality matching. We extend it for our scheduling problem to show that when $B = 2$ and jobs can be scheduled in arbitrary given time slots, an optimal preemptive schedule can be found in $O(\sqrt{L}m)$ time, for L the total length of all jobs. The proof is sketched in Sec. 4.2.
4. We give a tight bound on the gain from arbitrary preemption: an optimal schedule allowing preemption only at integral times uses at most 4/3 the active time of the optimal preemptive schedule. This bound is the best possible. The proof draws on the Edmonds-Gallai decomposition of matching theory. A sketch is provided in Sec. 4.3.

Related Work:
Problems of scheduling unit jobs have a rich history [29, 18]. For scheduling unit length jobs with arbitrary release times and deadlines on B processors to minimize the sum of completion times, Simons and Warmuth [29] extended the work by Simons [28] giving an algorithm with running time $O(n^2B)$ to find a feasible solution. For constant B, the running time is improved in [25].

In the closely related busy time problem [13, 21], jobs of arbitrary length have release times and deadlines and are scheduled in batches, so that the job demand in a batch is never more than a given value. The goal is to minimize the total busy time. Unlike our problem, this model permits an unbounded number of machines, making every instance feasible. The "min gap" problem [4] is to find a schedule of unit length jobs minimizing the number of idle intervals. We refer the reader to [18] for results on unit job scheduling and to surveys [20, 1] for a more comprehensive overview of scheduling results for power management problems.

2 Unit Jobs and Single Execution Windows

If time is not slotted and release times and deadlines are integral, then w.l.o.g., the optimal solution schedules jobs in "slots" implied by integer time boundaries.

We only provide a high level description of the algorithm. We assume at first that the instance is feasible, since this is easy to check by an EDF computation. Denote the distinct deadlines by $d_{i_1} < d_{i_2} < \ldots < d_{i_k}$, and let S_p be the set of jobs with deadline d_{i_p}. Then $S_1 \cup S_2 \ldots \cup S_k$ is the entire set of jobs where the deadlines range from time 1 to $T = d_{i_k}$.

[6] For the non-preemptive case minimizing active time is strongly NP-complete, even for $B = 2$.

We process the jobs in two phases. In Phase I we scan the jobs in order of *decreasing* deadline. We do not schedule any jobs, but modify their deadlines to create a new instance which has an equivalent optimal solution. The desired property of the new instance is that at most B jobs have the same deadline. Process the timeslots from right to left. At slot D, let S be the set of jobs that currently have deadline D. From S, select $\max(0, |S| - B)$ jobs with earliest release times and decrement their deadlines by one. If $|S| \leq B$ then we do not modify the deadlines of jobs in S. (Note that a job may have its deadline reduced multiple times since it may be processed repeatedly.)

Assume for simplicity's sake that after Phase I, S_p refers to the jobs of (modified) deadline d_{i_p}. In Phase II, jobs are actually scheduled. Initially, all jobs are labeled *unscheduled*. As the algorithm assigns a job to an active timeslot, it status changes to *scheduled*. Once a job is scheduled, it remains scheduled. Once a slot is declared active, it remains active for the entire duration of the algorithm.

We schedule the slots d_{i_p} from left to right. To schedule d_{i_p}, if there remain unscheduled jobs with that deadline, we schedule them. If there are fewer than B such jobs, we schedule additional jobs that are available. Job j is available if it is currently unscheduled and has $d_{i_p} \in [r_j, d_j)$. We schedule these available jobs EDF, until the slot is full or no more jobs are available.

Analysis of the Algorithm:

It is easy to implement our algorithm in $O(n \log n)$ time using standard data structures. Suppose the initial instance I is transformed by Phase I to a modified instance I'. We prove the following properties about an optimal solution for I'.

Proposition 1. *An optimal solution for I' has the same number of active slots as an optimal solution for the original instance I.*

Proposition 2. *W.l.o.g., an optimal solution for I' uses a subset of slots that are deadlines.*

Theorem 1. *The algorithm computes an optimal solution for I' (i.e., with the fewest active slots).*

Proposition 3. *On infeasible instances, the algorithm maximizes the number of scheduled jobs and does so in the fewest number of active slots.*

Linear Time Implementation: We define Phase I$'$ to be equivalent to Phase I: initially each deadline value has no jobs assigned to it. Process jobs j in order of decreasing release time, assigning j to a new deadline equal to the largest value $D \leq d_j$ that has $< B$ jobs assigned to it. We now argue their equivalence.

Let D (D^*, respectively) be the deadline assigned to job j in Phase I$'$ (Phase I). Suppose for jobs i, $i < j$, i's Phase I and Phase I$'$ deadlines are equal. Then $D^* \leq D$ by Phase I$'$'s choice of D. Phase I first assigns D to the $< B$ jobs currently assigned D in Phase I$'$, since they have release time $\geq r_j$. Among all jobs which can still be assigned D, j has the latest release time. So $D^* = D$. We implement Phase I$'$ in $O(n+T)$ time using a disjoint set merging algorithm [16].

Let $D_1 < D_2 < \ldots < D_\ell$ be the distinct deadline values assigned to jobs in Phase I. Phase II compresses the active interval from $[0, T]$ to $[0, \ell]$, as follows: job j with Phase I deadline D_k gets deadline k; its release time r_j changes to the largest integer i with $D_i \leq r_j$ (let $D_0 = 0$). The algorithm returns the EDF schedule on this modified instance, with slot i changed back to D_i. Phase II assigns new release times in time $O(n + T)$ and constructs an EDF schedule via disjoint set merging, achieving time $O(n + T)$ [16].

3 Disjoint Collection of Windows

We discuss the generalization where each job's feasible timeslots are an arbitrary set of slots (as opposed to a single interval). For any fixed $B \geq 3$ the problem is NP-hard (see the full paper). There is an $O(\log n)$ approximation for this problem by an easy reduction to a submodular covering problem [30].

3.1 Connections to Other Problems

Capacitated Vertex Cover. Given graph $G = (V, E)$ and vertex capacities $k(v)$, the goal is to pick a subset $S \subseteq V$ and assign edges to vertices in S, so that each edge is assigned to an incident vertex and each vertex v is assigned up to $k(v)$ edges. Approximation algorithms are given in [17, 6].

In the special case of the activation problem where each job has exactly two feasible timeslots, there is an equivalence with VCHC with uniform capacities $k(v) = B$: the slots are the vertices and the jobs are the edges. One implication of our result is that we can solve VCHC optimally when $k(v) = 2$.

Capacitated K-center problem. Given a graph, pick K nodes (called centers) and assign each vertex to a center that is "close" to it [23]. No more than L nodes should be assigned to a center, minimize d such that each node is assigned to a center within distance d of it. Create an unweighted graph G^d induced by edges of weight $\leq d$. Create job J_i and timeslot t_i for each node i. Let J_i be feasible at t_j if and only if edge (i, j) is in G^d or $j = i$. Then assigning all vertices to K centers in G^d is equivalent to scheduling all jobs in K timeslots so that no slot is assigned more than $B = L$ jobs. Since we can solve the latter problem for $B = 2$, we can solve the K-center problem when $L = 2$.

Capacitated Facility Location. Given a set of facilities (with capacities) and clients, the goal is to open a subset of facilities and find a capacity-respecting assignment of clients to open facilities that minimizes the sum of facility opening costs and connection costs. When capacities are uniformly two, the problem can be solved exactly. Details are described at the end of Sec. 3.2.

3.2 Polynomial solution for B=2

We consider scheduling jobs in the fewest number of active slots, when there are two processors and each job can be scheduled in a specified subset of timeslots. We use the following graph G. The vertex set is $J \cup T$, where J is the set of all jobs and T the set of all timeslots. The edge set is $\{(j,t) : \text{job } j \text{ can be scheduled in timeslot } t\} \cup \{(t,t) : t \in T\}$. A degree-constrained subgraph (DCS) problem is defined on G by the degree constraints $d(j) \leq 1$ for every $j \in J$ and $d(t) \leq 2$ for every $t \in T$. By definition a loop (t,t) contributes two to the degree of t. Any DCS gives a schedule, and any schedule gives a DCS that contains a loop in every nonactive timeslot. Also any DCS D that covers ι jobs and contains λ loops has cardinality $|D| = \iota + \lambda$.

Lemma 1. *A maximum cardinality DCS (MC-DCS) D of G minimizes the number of active slots used by any schedule of $|V(D) \cap J|$ jobs.*

Let n be the number of jobs and m the number of given pairs (j,t) where job j can be scheduled at time t. Observe that G has $O(m)$ vertices and $O(m)$ edges, since we can assume every timeslot t is on some edge (j,t). In any graph, the majority of edges in a simple non-trivial path are not loops. So in the algorithms for MC-DCS and maximum matching, an augmenting path has length $O(n)$. An MC-DCS on G can be found in time $O(\sqrt{n}m)$. The approach is via non-bipartite matching. We describe two ways to handle the loops.

The first approach reduces the problem to maximum cardinality matching in a graph called the MG graph. To do this modify G, replacing each timeslot vertex t by two vertices t_1, t_2. Replace each edge (j,t) by two edges from j to the two replacement vertices for t. Finally replace each loop (t,t) by an edge joining the two corresponding replacement vertices.

A DCS D corresponds to a matching M in a natural way: if slot t is active in D then M matches corresponding edges of the form (j,t_i). If loop (t,t) is in D then M matches the replacement edge (t_1,t_2). Thus it is easy to see that an MC-DCS corresponds to a maximum cardinality matching of the same size.

A second approach is to use an algorithm for MC-DCS on general graphs. If it does not handle loops directly, modify G by replacing each loop (t,t) by a triangle (t, t_1, t_2, t), where each t_i is a new vertex with degree constraint $d(t_i) = 1$. A DCS in G corresponds to a DCS in the new graph that contains exactly $|T|$ more edges, one from each triangle. The cardinality algorithm of Gabow and Tarjan [15] gives the desired time bound. Let ι^* be the greatest number of jobs that can be scheduled. One can compute ι^* in time $O(\sqrt{n}m)$ by finding a MC-DCS on the bipartite graph formed by removing all loops from G [11].

Theorem 2. *A schedule for the greatest possible number of jobs (ι^*) that minimizes the number of active slots can be found in time $O(\sqrt{n}m)$.*

The idea is to show that the choice of ι^* jobs is irrelevant: the minimum number of active slots for a schedule of ι^* jobs can be achieved using *any* set of ι^* jobs that can be scheduled.

We note that matching is a natural tool for power minimization. Given a maximum cardinality matching problem, create a job for each vertex. If two vertices are adjacent, create a common slot in which they can be scheduled. A maximum cardinality matching corresponds to a schedule of minimum active time. Thus if $T(m)$ is the optimal busy time, a maximum cardinality matching on a graph of m edges can be found in $O(T(m))$ time. For bipartite graphs, the number of slots can be reduced from m to n. Thus a maximum cardinality matching on a bipartite graph of n vertices and m edges can be found in $O(T(n,m))$ time, where $T(n,m)$ is the optimal time to find a minimum active time schedule for $O(n)$ jobs, $O(n)$ timeslots, and m pairs (j,t).

Our algorithm can be extended to other versions of the power minimization problem. For example the following corollary models a situation where power is limited.

Corollary 1. *For any given integer α, a schedule for the greatest possible number of jobs using at most α active slots can be found in time $O(\sqrt{n}m)$.*

Now consider capacitated facility location with capacities being two. With clients as jobs and facilities as slots, denote by MG_w the weighted equivalent of the graph MG. Edge (j,t_i) has weight equal to the cost of connecting client j to facility t, for $i=1,2$. Edge (t_1,t_2) has weight $-C(t)$ where $C(t)$ is the cost of opening facility t. Then, solving the facility location problem amounts to determining a minimum weight matching in MG_w where each job is matched, which can be achieved by generalizing our algorithm to the weighted case in a natural way.

4 Active Time and Arbitrary Preemption

This section considers jobs j of arbitrary lengths ℓ_j. There are B processors that operate in a collection of unit length "timeslots" $s \in S$. Each job has a set of timeslots in which it may be scheduled. Preemptions are allowed at any time, i.e., job j must be scheduled in a total of ℓ_j units of time but it can be started and stopped arbitrarily many times, perhaps switching processors. The only constraint is that it cannot execute on more than one processor at a time. We seek a schedule minimizing the active time.

4.1 Linear program formulation

The problem can be written as a linear program. Form a bipartite graph where edge $js \in E$ if and only if job j can be scheduled in slot s. A variable x_{js} gives the amount of time job j is scheduled in slot s, and a variable i_s gives the amount of idletime in slot s. The problem is equivalent to the following LP:

$$\begin{array}{lllll}
\text{max.} & \sum_{s\in S} i_s & & & \\
\text{s.t.} & \sum_{js\in E} x_{js} & \geq \ell_j & j \in J & (1.a) \\
 & \sum_{js\in E} x_{js} + B i_s & \leq B & s \in S & (1.b) \\
 & x_{js} + i_s & \leq 1 & js \in E & (1.c) \\
 & i_s,\, x_{js} & \geq 0 & s \in S,\, js \in E &
\end{array} \qquad (1)$$

Observe that the inequalities (1.b)–(1.c) are necessary and sufficient for scheduling x_{js} units of job j, $js \in E$, in $1 - i_s$ units of time (on B processors). Necessity is clear. For sufficiency, order jobs and processors arbitrarily.

Repeatedly schedule the next $1 - i_s$ units of jobs on the next processor, until all jobs are scheduled. Even though a job may be split across processors, it would be scheduled last on one processor and first on the next, so (1.c) guarantees it is not executed simultaneously on two processors. A variant of this LP gives a polynomial-time solution to the problem where each job has a single interval $[r_j, d_j]$ in which it can be scheduled. The idea is to redefine S as the collection of minimal intervals formed by the $2n$ points $\{r_j,\, d_j : j \in J\}$.

4.2 The case $B = 2$ and 2-matchings

It is convenient to use a variant of the above LP based on the matching graph MG of Sec. 3. Each edge of MG has a linear programming variable. Specifically each edge $js \in E(G)$ gives rise to two variables x_e, $e = js_i$, $i \in \{1, 2\}$, that give the amount of time job j is scheduled on processor i in timeslot s. Also each timeslot $s \in S$, has a variable x_e, $e = s_1 s_2$ that gives the amount of inactive time in slot s. The problem is equivalent to the following LP:

$$\begin{array}{lllll}
\text{max.} & \sum\{x_e : e \in E(MG)\} & & & \\
\text{s.t.} & \sum\{x_e : e \text{ incident to } j\} & = \ell_j & j \in J & \text{(2.a)} \\
 & \sum\{x_e : e \text{ incident to } s_i\} & \leq 1 & s \in S,\ i \in \{1,2\} & \text{(2.b)} \\
 & \sum\{x_e : e \in \{js_1, js_2, s_1 s_2\}\} & \leq 1 & js \in E(G) & \text{(2.c)} \\
 & x_e & \geq 0 & e \in E(MG) &
\end{array} \qquad (2)$$

To see that this formulation is correct note that any schedule supplies feasible x_e values. Conversely any feasible x_e's give a feasible solution to LP (1) (e.g., set $x_{js} = x_{js_1} + x_{js_2}$ and $i_s = x_{s_1 s_2}$) and so corresponds to a schedule. Finally the objective of (2) is the sum over job lengths plus the total inactive time, so like (1) it maximizes the total inactive time.

Now consider an arbitrary graph $G = (V, E)$ and the polyhedron defined by the following system of linear inequalities:

$$\begin{array}{llll}
\sum\{x_e : e \text{ incident to } v\} & \leq 1 & v \in V & \text{(3.a)} \\
\sum\{x_e : e \text{ an edge of } T\} & \leq 1 & T \text{ a triangle of } G & \text{(3.b)} \\
x_e & \geq 0 & e \in E &
\end{array} \qquad (3)$$

Call $\sum\{x_e : e \in E\}$ the *size* of a solution to (3). Say vertex v is *covered* if equality holds in (3.a). Define a *2-matching* M to be an assignment of weight 0,1 or $\frac{1}{2}$ to each edge of G so that each vertex is incident to edges of total weight ≤ 1.[7] M is *basic* if its edges of weight $\frac{1}{2}$ form a collection of (vertex-disjoint) odd cycles. The basic 2-matchings are precisely the vertices of the polyhedron determined by the constraints (3) with (3.b) omitted. A basic 2-matching is *triangle-free* if no triangle has positive weight on all three of its edges. Cornuéjols and Pulleyblank [8] showed the triangle-free 2-matchings are precisely the vertices of the polyhedron determined by constraints (3).

[7] This definition of a 2-matching scales the usual definition by a factor $\frac{1}{2}$, i.e., the weights are usually 0,1 or 2.

When job lengths are one, the inequalities of (2) are system (3) for graph MG, with the requirement that every job vertex is covered. It is clear that the result of Cornuéjols and Pulleyblank implies our scheduling problem is solved by any maximum-size triangle-free 2-matching on MG subject to the constraint that it covers each job vertex. Interestingly, there is always a solution where each job is scheduled either completely in one slot or split into two pieces of size $1/2$.

Cornuéjols and Pulleyblank give two augmenting path algorithms for triangle-free 2-matching: they find such a matching that is perfect (i.e., every vertex is fully covered) in time $O(nm)$ [9], and such a matching that has minimum cost in time $O(n^2m)$ [8]. (The latter bound clearly applies to our scheduling problem, since it can model the constraint that every job vertex is covered.) Babenko et. al. [3] showed that a maximum cardinality triangle-free 2-matching can be found in time $O(\sqrt{n}m)$. This is done by reducing the problem to ordinary matching, with the help of the Edmonds-Gallai decomposition.

The full paper gives two easy extensions of [3]: a simple application of the Mendelsohn-Dulmage Theorem [24] shows that a maximum cardinality triangle-free 2-matching can be found in the same asymptotic time as a maximum cardinality matching (e.g., the algebraic algorithm of Mucha and Sankowski [27] can be used). This result extends to maximum cardinality triangle-free 2-matchings that are constrained to cover a given set of vertices (this is the variant needed for our scheduling problem). The development in the full paper is self-contained and based on standard data structures for blossoms. For instance, it is based on a simple definition of an *augmenting cycle* (analog of an augmenting path), leading to an *augmenting blossom* (which models the *triangle cluster*[8] of Cornuéjols and Pulleyblank [8, 9]); we show a maximum cardinality matching with the greatest number of augmenting blossoms gives a maximum cardinality cardinality triangle-free 2-matching. [9]

4.3 Applications to preemptive scheduling for $B = 2$

Our algorithm for finding a maximum triangle-free 2-matching and LP (2) solve the preemptive scheduling problem for unit jobs. We extend this to jobs of arbitrary length by reducing to unit lengths, using vertex substitutes from matching theory. A sketch follows; details are in the full paper.

A graph UG is defined wherein a job of length ℓ that can be preemptively scheduled in s given time slots is represented by a complete bipartite graph $K_{s-\ell,s}$. The $s-\ell$ vertices on the small side will be matched to $s-\ell$ vertices on the big side, leaving ℓ vertices to be matched to time slots where the job is scheduled. The triangles of UG are similar to those of MG. Lemma 4 (*ii*) in the full paper (or [9]) implies that a solution to the scheduling problem is given by a maximum cardinality triangle-free 2-matching on UG that covers all vertices in $K_{s-\ell,s}$ graphs. For efficiency we use the sparse vertex substitute technique of [14] which works on (various) graphs of $O(m)$ edges rather than UG itself.

[8] A triangle cluster is a connected subgraph whose edges can be partitioned into triangles, such that every vertex shared by two or more triangles is a cut vertex.

[9] Our algorithm was developed independently of the authoritative algorithm of [3].

Theorem 3. *Let J be a set of unit jobs that can be scheduled on $B = 2$ processors. A preemptive schedule for J minimizing the active time can be found in $O(\sqrt{n}m)$ time. The result extends to jobs of arbitrary integral length ℓ_j, where the time is $O(\sqrt{L}m)$ for L the sum of all job lengths.*

What can be gained by allowing arbitrary preemption? For example when $B = 2$, three unit jobs that may be scheduled in slots 1 or 2 require 2 units of active time under preemption at integral points, and only 3/2 units under arbitrary preemption. We now show this example gives the greatest disparity between the two preemption models for $B = 2$.

Recall that we construct an optimal preemptive schedule $\mathcal{P}$ by finding a maximum size triangle-free 2-matching (on MG for unit jobs, and UG for arbitrary length jobs) and converting it to $\mathcal{P}$ in the obvious way via LP (2). The following lemma is proved by examining the structure of blossoms in our special graphs, e.g., the triangles all have the form js_1s_2 for j a job vertex and s_1s_2 the edge representing a time slot; also vertices s_1 and s_2 are isomorphic. The matching terminology in the following lemma is from [10; or see 24].

Lemma 2. *(i) A blossom in MG (UG) is a triangle cluster iff it contains exactly one vertex of J (J_U).(ii) Let H be a Hungarian subgraph in MG (UG). Any slot s either has both its vertices inner, or both its vertices outer (and in the same blossom) or neither vertex in V_H. (iii) Let the optimal preemptive schedule $\mathcal{P}$ be constructed from $T(M)$, the 2-matching corresponding to matching M, and let $\mathcal{B}$ be an augmenting blossom of M. The slots with both vertices in $\mathcal{B}$ have $\geq 3/2$ units of active time in $\mathcal{P}$.*

Theorem 4. *For $B = 2$ and a set of jobs J, the minimum active time in a schedule allowing preemption at integral points is $\leq 4/3$ times that of a schedule permitting arbitrary preemption.*

Remark: The theorem holds for unit jobs, and for jobs of arbitrary length ℓ_j when time is slotted, i.e., a schedule is allowed to execute a length ℓ_j job in ℓ_j distinct timeslots, but no more.

References

1. S. Albers. Energy-efficient algorithms. in *CACM*, 53(5):86–96, 2010.
2. H. Amur, J. Cipra, V. Gupta, M. Kozuch, G. Ganger and K. Schwan. Robust and flexible power-proportional storage. *Proceedings of 1st SoCC*, pp. 217–218, 2010.
3. M. Babenko, A. Gusakov and I. Razenshteyn. Triangle-free 2-matchings revisited. *Computing and Combinatorics*, pp. 120–129, 2010.
4. P. Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. *SODA 2006*.
5. J. Chang, H. N. Gabow, and S. Khuller. A Model for Minimizing Active Processor Time. `http://www.cs.umd.edu/~samir/grant/active.pdf`, 2012.
6. J. Chuzhoy and S. Naor. Covering problems with hard capacities. *SIAM J. on Computing*, 36(2):498–515, 2006.

7. A. Condotta, S. Knust and N. Shakhlevich. Parallel batch scheduling of equal-length jobs with release and due dates. *J. of Scheduling*, 13(5):463–677, 2010.
8. G. Cornuéjols and W. Pulleyblank, A matching problem with side conditions. *Discrete Math.*, 29(2):135–159, 1980.
9. G. Cornuéjols and W. Pulleyblank, Perfect triangle-free 2-matchings. *Math. Programming Study*, 13:1–7, 1980.
10. J. Edmonds. Paths, trees and flowers. *Canadian J. Math.*, 17:447-467, 1965.
11. S. Even and R.E. Tarjan. Network flow and testing graph connectivity. *SIAM J. on Computing*, 4(4):507–581, 1975.
12. G. Even, R. Levi, D. Rawitz, B. Schieber, S. Shahar, M. Sviridenko. Algorithms for capacitated rectangle stabbing and lot sizing with joint set up costs. *ACM Trans. on Algorithms*, 4(3):34–51, 2008.
13. M. Flammini, G. Monaco, L. Moscardelli, H. Shachnai, M. Shalom, T. Tamir, S. Zaks. Minimizing total busy time in parallel scheduling with application to optical networks. *IPDPS Conference*, pp. 1–12, 2009.
14. H. N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. *Proc. 15th Annual STOC*, 448–456, 1983.
15. H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph matching problems. *J. ACM*, 38(4): 815–853, 1991.
16. H. N. Gabow and R.E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. Computer and System Sciences*, 30(2):209–221, 1985.
17. R. Gandhi, E. Halperin, S. Khuller, G. Kortsarz and A. Srinivasan. An improved approximation algorithm for vertex cover with hard capacities. *J. of Computer and System Sciences*, 72(1):16–33, 2006.
18. M. Garey, D. Johnson, B. Simons and R. Tarjan. Scheduling unit-time jobs with arbitrary release times and deadlines. *SIAM J. on Computing*, 10(2):256–269, 1981.
19. Y. Ikura and M. Gimple. Efficient scheduling algorithms for a single batch processing machine. *Operations Research Letters*, 5(2):61–65, 1986.
20. S. Irani and K. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2): 63–76, 2005.
21. R. Khandekar, B. Schieber, H. Shachnai and T. Tamir. Minimizing busy time in multiple machine real-time scheduling. *FST&TCS Conference*, pp. 169–180, 2010.
22. F. Koehler and S. Khuller. Quick and efficient: fast algorithms for completion time and batch minimization on multiple machines. *Manuscript.*
23. S. Khuller and Y. J. Sussman. The capacitated k-center problem. *SIAM J. on Discrete Mathematics*, 13(30):403–418, 2000.
24. E. Lawler. Combinatorial Optimization. *Holt, Rinehart and Winston*, 1976.
25. A. López-Ortiz and C.-G. Quimper. A fast algorithm for multi-machine scheduling problems with jobs of equal processing times. *Proc. of Symposium on Theoretical Aspects of Computer Science*, 2011.
26. L. Lovász and M.D. Plummer. *Matching Theory*, North-Holland, 1986.
27. M. Mucha and P. Sankowski. Maximum Matchings via Gaussian Elimination. *FOCS*, pp. 248-255, 2004.
28. B. Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM J. on Computing*, 12(2):294–299, 1983.
29. B. Simons and M. Warmuth. A fast algorithm for multiprocessor scheduling of unit length jobs. *SIAM J. on Computing*, 18(4):690–710, 1989.
30. L. A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.