Brief Announcement: Nested Active-Time Scheduling

Nairen Cao
nairen@ir.cs.georgetown.edu
Georgetown University
Washington D.C., USA

Jeremy T. Fineman
jfineman@cs.georgetown.edu
Georgetown University
Washington D.C., USA

Shi Li
shil@buffalo.edu
University at Buffalo
Buffalo, New York, USA

Julián Mestre
julian.mestre@sydney.edu.au
The University of Sydney
Sydney, Australia

Katina Russell
katina.russell@cs.georgetown.edu
Georgetown University
Washington D.C., USA

Seeun William Umboh
william.umboh@sydney.edu.au
The University of Sydney
Sydney, Australia

ABSTRACT

The ***active-time scheduling problem*** considers the problem of scheduling preemptible jobs with windows (release times and deadlines) on a parallel machine that can schedule up to g jobs during each timestep. The goal in the active-time problem is to minimize the number of active steps, i.e., timesteps in which at least one job is scheduled.

This paper presents a 9/5-approximation algorithm for a special case of the active-time scheduling problem in which job windows are laminar (nested). This result improves on the previous best 2-approximation for the general case.

CCS CONCEPTS

• **Theory of computation → Scheduling algorithms**.

KEYWORDS

Scheduling algorithms

ACM Reference Format:
Nairen Cao, Jeremy T. Fineman, Shi Li, Julián Mestre, Katina Russell, and Seeun William Umboh. 2022. Brief Announcement: Nested Active-Time Scheduling. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '22), July 11–14, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3490148.3538554

1 INTRODUCTION

In the active-time problem [1], we are given as input an integer g and a set J of n jobs, where each job $j \in J$ has an associated processing time p_j, release time r_j, and deadline $d_j \geq r_j + p_j$, all integers. The jobs are scheduled on a parallel machine that can execute up to g jobs during each step. Time is organized into discrete (integer) steps or slots, and preemption is allowed but only at slot boundaries. We call the time interval $[r_j, d_j)$ the job j's ***window***. Each job j must be fully scheduled within its window.

We say that a timestep t is ***active*** if the schedule assigns at least one job to step t. The goal in the ***active-time scheduling problem*** is to find a schedule with minimum number of active steps that schedule all jobs within their windows. Chang, Gabow, and Khuller [1] and Kumar and Khuller's [3] both achieve 2 approximation through LP rounding algorithm and greedy algorithm, respectively.

To push past the barriers for the general version of the problem, this paper instead considers a special case of the active-time problem in which the job windows are laminar (nested). That is, for each pair of jobs i, j, either the intervals $[r_i, d_i)$ and $[r_j, d_j)$ are disjoint (meaning either $d_i \leq r_j$ or $d_j \leq r_i$)), or one of the intervals is fully contained in the other (i.e., either $r_i \leq r_j < d_j \leq d_i$ or $r_j \leq r_i < d_i \leq d_j$).

Our main result is a 9/5-approximation algorithm for the active-time problem with laminar job windows. Since the simple example exhibiting the integrality gap [2] of 2 for the natural LP is a nested instance, a different LP formulation is needed. Our algorithm starts by solving a stronger linear program (LP) for the problem to produce a fractional solution, then performing a new rounding process over the tree of job windows. The algorithm itself is not overly complex, but the analysis is not at all straightforward.

2 PRELIMINARIES

For an integer p, We use $[p]$ to represent integers from $\{1, 2, ..., p\}$. Given an instance of the nested active time problem, we define its tree T as follows. Each tree node i is associated with an interval $K(i)$ such that $K(i) = [r_j, d_j)$ for some $j \in J$. If there are several jobs with the same interval, we only create a single tree node. A tree node i' is a child of i if $K(i') \subsetneq K(i)$ and no other node interval is strictly between $K(i)$ and $K(i')$, i.e, there is no node i'' such that $K(i') \subsetneq K(i'') \subsetneq K(i)$. The descendants and ancestors of a node i are denoted $Des(i)$ and $Anc(i)$, respectively. Note that both $Des(i)$ and $Anc(i)$ include i itself. We define $\text{par}(i)$ to be the parent node of i. W.l.o.g we can assume T is indeed a tree (instead of a forest) since otherwise the instance can be broken into several independent ones.

We assume that the tree contains m nodes and each node is associated with an unique id in $[m]$. Now, each job j's interval is associated with a node in the tree. For a job j, define $k(j)$ to be the tree node i with $K(k(j)) = [r_j, d_j)$; we say j belongs to the node i if $i = k(j)$. For jobs j_1 and j_2, if $r_{j_1} = r_{j_2}$ and $d_{j_1} = d_{j_2}$, then $k(i) = k(j)$. Given a node i and a job subset $J' \subseteq J$, $J'(i) = \{j \in J' \mid k(j) = i\}$ is the set of jobs in J' belonging to i. Note that at least one job belongs to each node. Define the length of a node i, which is denoted as $L(i)$, as the $|K(i)| - \sum_{i':\text{par}(i')=i} |K(i')|$, i.e, the number of time slots in the interval $K(i)$, but not in $K(i')$ for any

$$\textbf{min} \quad \sum_{i\in[m]} x(i) \quad \textbf{s.t.} \tag{1}$$

$$\sum_{i\in Des(K(j))} y(i,j) \geq p_j, \qquad \forall j \tag{2}$$

$$\sum_{j\in J(Anc(i))} y(i,j) \leq g\cdot x(i) \qquad \forall i \tag{3}$$

$$x(i) \leq L(i) \qquad \forall i \tag{4}$$

$$y(i,j) \leq x(i) \qquad \forall i,j \tag{5}$$

$$y(i,j) = 0 \qquad \forall i, j \notin J(Anc(i)) \tag{6}$$

$$\sum_{i'\in Des(i)} x(i') \geq 2 \qquad \forall i, \mathrm{OPT}_i \geq 2 \tag{7}$$

$$\sum_{i'\in Des(i)} x(i') \geq 3 \qquad \forall i, \mathrm{OPT}_i \geq 3 \tag{8}$$

Figure 1: Linear program for active time scheduling. By default we restrict $i \in [m]$ and $j \in J$.

child node i' of i. Given a function or vector f and a set S, we define $f(S) = \bigcup_{e\in S} f(e)$ or $f(S) = \bigcup_{e\in S} f_e$.

We say that a node i is ***rigid*** if a feasible solution must open the entire interval $K(i)$. For our rounding algorithm, we will make two extra transformations for the tree.

First, we transform an arbitrary tree to a binary tree. If a parent node i contains several children nodes $i_1, i_2, ..., i_t$, we will create several virtual nodes so that each node contains at most 2 children. Each virtual node's interval is the union of its children's intervals. There are no jobs associated with the virtual nodes and the length of a virtual node i' satisfying $L(i) = 0$. Notice that this transformation adds at most t virtual nodes for a node with t children. In total, this transformation only adds k virtual nodes to a tree that had k nodes originally. In the resulting tree, only internal nodes can be virtual so each leaf node must have at least one job associated with it.

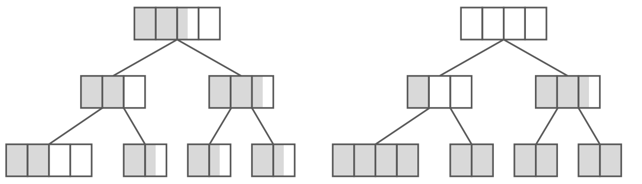
We perform one final transformation to make each leaf node rigid. For a leaf node i, let $j \in J(i)$ be a job in i with the longest processing time. If $p_j = L(i)$, then we leave i and the jobs therein unchanged. Otherwise, we can assume that j is scheduled in the first p_j steps of i because j is the longest job in the leaf node and all jobs in the leaf node could choose the leaf's interval to fit in. We transform the instance by creating a virtual child node i' of the leaf i with interval corresponding to the first p_j steps of $I(i)$, and we reduce j's window to match i''s. Notice this transformation does not change our solution for the original tree.

3 ALGORITHM

3.1 Linear Program

The linear program is LP (1), which is given in Figure 1. In the LP, $x(i)$ denotes the number of time slots opened in node i, and $y(i,j)$ denotes the amount of job j that is scheduled in node i. In the LP, OPT_i denotes the smallest number of slots to schedule the jobs in $J(Des(i))$.

The objective is to minimize $\sum_{i\in[m]} x(i)$. (2) ensures that every job j is scheduled in at least p_j time slots. (3) ensures that the total



(a) The open slots from an LP solution (b) The open slots after performing the LP transformation

Figure 2: An example of a tree before and after running the LP transformation in Lemma 1. The dark slots represent slots have jobs scheduled in them, and the white slots are closed.

number of jobs scheduled in $x(i)$ is at most $g\cdot x(i)$, for each node $i \in [m]$. (4) requires that the number of open time slots in a node $x(i)$ is at most the interval length $L(i)$ of node i. (5) says that we could give at most $x(i)$ time slots for a job j. (6) restricts that for each job $j \in J$, j can only be put into nodes in $Des(K(j))$. (7) and (8) are the key constraints that makes the LP stronger. They are clearly valid; moreover, checking if $\mathrm{OPT}_i \geq 2$ ($\mathrm{OPT}_i \geq 3$) can be done easily.

After running the LP and getting a solution (x, y), we will perform a transformation on the solution.

3.2 Transformation of LP Solution

Lemma 1. *Given a feasible LP solution, we can efficiently output another feasible LP solution such that for any pair of nodes i_1, i_2 such that $i_2 \in Des^+(i_1)$, if $x(i_2) < L(i_2)$, then $x(i_1) = 0$.*

The high level idea of Lemma 1 is to push down some part of the time slot if its descendant time slot is not full. We leave the proof to our full paper. An example of of the LP transformation is shown in Figure 2. Lemma 1 implies that for any i with $x(i) > 0$, all of its strict descendants are fully open. We let I be the set of topmost nodes i with $x(i) > 0$; those are the nodes i with $x(i) > 0$ but all its strict ancestors i' have $x(i') = 0$. Notice that after transformation, all nodes with fractional time slot will be in I.

3.3 Rounding Algorithm to Obtain an Integral Vector $\tilde{x} \in \{0,1\}^{[m]}$

The rounding algorithm that constructs our integral $\tilde{x}$ is given in Algorithm 1.

Algorithm 1 Rounding Algorithm

```
1: let $\tilde{x}(i) \leftarrow \lfloor x(i) \rfloor, \forall i \in I$ and $\tilde{x}(i) \leftarrow x(i), \forall i \in [m] \setminus I$.
2: for every node $i \in Anc(I)$ from bottom to top
3:     while $\frac{9x(Des(i))}{5} \geq \tilde{x}(Des(i)) + 1$
4:         if $\exists i' \in Des(i)$ with $\tilde{x}(i') < x(i')$ then
5:             choose such an $i'$ arbitrarily
6:             let $\tilde{x}(i') \leftarrow \lceil x(i') \rceil$
7:         else
8:             break
```

Clearly, the number of open slots is at most $\frac{9x([m])}{5}$.

Lemma 2. *After running the Algorithm 1,* $\tilde{x}([m]) \le \frac{9x([m])}{5}$.

Notice that we might round down some fractional time slots, which will make the new time slot infeasible. We leave the proof of feasibility to our full paper.

4 NP-COMPLETENESS

In this section, we show that the decision version of the nested active time problem is NP complete. Very recently, Sagnik and Manish [4] showed the general case is NP complete. Unfortunately, their proof uses crossing intervals. Our proof reduces the nested active time problem to a new problem that we call *prefix sum cover*, which is related to the classic set cover problem.

Prefix sum cover problem. For any pair of d-dimensional vectors $v = (v_1, v_2, ..., v_d), w = (w_1, w_2, ..., w_d) \in N^d$, we say $v \prec w$ if and only if for all $j \in [1, d]$, $\sum_{i \le j} v_i \ge \sum_{i \le j} w_i$. In the prefix sum cover problem, we are given n vectors $u_1, u_2, ...u_n \in N_+^d$, a target vector $v \in N^d$ and an integer number k, and we want to find k vectors $u_{l_1}, u_{l_2}, ..., u_{l_k}$ such that $\sum_{i \le k} u_{l_i} \prec v$.

Moreover, for the purposes of our reduction, we consider a restricted version of the problem. Let W be the maximum scalar that appears in any of the vectors $u_1, ..., u_n$ and v. First, we require that both d and W be bounded by some polynomial of n. For a vector $w \in N^d$, let $[w]_j$ be its j-th dimension value. Second, for each $i \in [1, n]$, we require that $[u_i]_1 \ge [u_i]_2 \ge ... \ge [u_i]_d$, and $[v]_1 \ge [v]_2 \ge ... \ge [v]_d$, i.e., all vectors are non-decreasing. We show in our full paper that prefix sum cover is NP complete.

Reduction. Now we will reduce the prefix sum cover problem to the active time problem. Let $(\{u_1, u_2, \dots, u_n\}, v, k)$ be the prefix sum cover instance. Our nested active time instance is defined by a set of jobs J and it uses $p = dW$ machines. Our instance is made up of three kinds of jobs:

- For each vector u_i, and each $w \in [2, W]$, we have $p - |\{j \in [1, d] \mid [u_i]_j \ge w\}|$ rigid unit length jobs, each with window consisting of a single slot $[(i-1)W + w - 1, (i-1)W + w]$.
- For each vector u_i, we also have $\sum_{j \le d} [u_i]_j - d$ flexible unit jobs with window $[(i-1)W, iW]$.
- Finally, we have jobs that depend on the target vector. For each $j \in [1, d]$, we have a job with length $[v]_j$ and window $[0, nW]$.

We denote each of these sets of job with S_1 (rigid jobs), S_2 (flexible jobs associate with each u_i vector), and S_3 (jobs associated with the target vector).

Let us try to schedule this instance, starting with S_1. Since the jobs in S_1 are rigid, we must open all slots in $[(i-1)W + 1, iW]$. Notice that each of these slots has at least $p - d$ jobs in S_1, so each of these time slots has at most d unused machines after scheduling S_1.

Next we will try to fit jobs from S_2 into $[(i-1)W, iW]$. Observe that the total capacity in the window $[(i-1)W+1, iW]$ is $p(W-1)$ and that the jobs from S_1 take up up $\sum_{w \in [2,W]} (p - |\{j \in [1, d] \mid [u_i]_j \ge w\}|)$ capacity. Further observe that

$$\sum_{w \in [2,W]} (p - |\{j \in [1, d] \mid [u_i]_j \ge w\}|) + \sum_{j \le d} [u_i]_j - d = p(W-1).$$

Therefore, if we do not open the slot $[(i-1)W, (i-1)W + 1]$, then the jobs from S_1 and S_2 will use up all of the available capacity in the time window $[(i-1)W, iW]$. This is important, as it means that we cannot schedule any job from S_3 in this window.

We say that the time slots $[(i-1)W, (i-1)W + 1]$ for $i \in [n]$ are *special*. Since all non-special slots in $[0, nW]$ must be open, the problem boils down to opening as few special slots as possible to accommodate the jobs in S_3.

Suppose we open the special time slot $[(i-1)W, (i-1)W + 1]$. We claim that all jobs in S_2 will be assigned to the special time slot. Indeed, even after all S_2 jobs are assigned to this slot, there are still $p - (\sum_{j \le d} [u_i]_j - d) \ge d$ unused machines in it, while we can only have at most d unused machines in each time slots in $[(i-1)W + 1, iW]$ after scheduling S_1.

A *configuration* is a sequence $(z_1, z_2, ..., z_M)$, where z_i is the number of unused machines at time $[i-1, i]$. Thus, once we have chosen which special slots to open, we get the configuration which tells us how many machines are left unused in each time slot.

Number the machines from 1 to p. For any given configuration, we can assume without loss of generality that if we have z_t unused capacity at time slot $[t-1, t]$ then machines 1 through z_t are unused; i.e, we always leaves smaller index machine unused. Let e_j be the number of empty time slots at machine j. We give an if-and-only-if condition for the feasibility based on the e_j values, whose proof is left to the full version.

Lemma 3. *Given a configuration, let e_j be the machine unused slot defined above and J' be a set of $q \le p$ jobs with no release time and due time constraint. Let $l_1 \ge l_2 \ge ... \ge l_q$ be the lengths of the jobs in J. The configuration can fit all jobs in J' if and only if $\sum_{i \le j} e_i \ge \sum_{i \le j} l_i$ for all $j \in [1, q]$.*

Now we show how to apply it to the active time instance. We will set $J = S_3$ and $q = d$. For any interval $[(i-1)W, iW]$, let $e_{1,i}, e_{2,i}, ..., e_{d,i}$ be the unused time slot for machine 1 to d in this interval. If we close the special time slot $[(i-1)W, (i-1)W + 1]$, then there is no capacity left so $e_{1,i} = e_{2,i} = ... = e_{d,i} = 0$. If we open it, then $e_{j,i} = [u_i]_j$, i.e. the j-th machine will hold exactly $[u_i]_j$ unused time slots in the interval. Now the problem becomes we want to open k special time slots such that the resulting configuration can fit all jobs from S_3. Lemma 3 implies that it is equivalent to choosing k vectors from $(e_{1,1}, \dots, e_{d,1}) = u_1, ..., (e_{1,n}, \dots, e_{d,n}) = u_n$ such that $\sum_{i \le j} e_i \ge \sum_{i \le j} [v]_i$ for every $j \in [1, d]$, which is exactly the definition of our prefix sum cover problem.

ACKNOWLEDGEMENTS

This research was supported in part by NSF grants CCF-1918989 and CCF-2106759.

REFERENCES

[1] Jessica Chang, Harold N. Gabow, and Samir Khuller. A model for minimizing active processor time. *Algorithmica*, 70(3):368–405, 2014.
[2] Jessica Chang, Samir Khuller, and Koyel Mukherjee. LP rounding and combinatorial algorithms for minimizing active and busy time. *J. of Scheduling*, 20(6):657–680, 2017.
[3] Saurabh Kumar and Samir Khuller. Brief announcement: A greedy 2 approximation for the active time problem. In *Proc. of the 30th on Symposium on Parallelism in Algorithms and Architectures*, page 347–349, 2018.
[4] Sagnik Saha and Manish Purohit. Np-completeness of the active time scheduling problem. *arXiv preprint arXiv:2112.03255*, 2021.