Brief Announcement: A greedy 2 approximation for the active time problem

Saurabh Kumar
University of Maryland, College Park
kumarsa@cs.umd.edu

Samir Khuller[*]
University of Maryland, College Park
samir@cs.umd.edu

ABSTRACT

In this note, we give a very simple 2 approximation for the active time problem - we are given a set of pre-emptible jobs, each with an integral release time, deadline and required processing length. The jobs need to be scheduled on a machine that can process at most g distinct job units at any given integral time slot in such a way that we minimize the time the machine is on i.e the active time. Our algorithm matches the state of the art bound obtained by a significantly more involved LP rounding scheme.

CCS CONCEPTS

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

1 INTRODUCTION

In this paper, we consider the problem of scheduling jobs on a machine while minimizing the total time that the machine is on. This is captured by the active time model.

Active Time Model: We have a set of n jobs say $J = \{1, 2, ..., n\}$ where each job j has a processing time p_j and must be scheduled in a window defined by a release time r_j and deadline d_j (p_j, r_j, d_j are integers). Jobs are preemptible at integral points within their window. Time is divided into integral units. We are given a single machine that can process at most g distinct job units in parallel. The machine is considered on i.e *active* in a particular time unit when it is processing at least one job in that time unit. Our goal is to feasibly schedule the jobs in J while minimizing the *active time* (i.e the number of time units that the machine is on).

Chang et. al. [2] solve the problem exactly when jobs all have unit length. They show that the problem is NP hard when a job can have multiple disjoint windows but the complexity of the case where each job has a single contiguous window is unknown. The unit length version of this problem has been considered in other contexts such as in scheduling jobs with precedence constraints [6], finding a minimum b-clique cover in an interval graph [1], and rectangle stabbing [4].

The general problem with arbitrary integral job lengths was considered by Chang et. al. [3] where the authors show that a minimal feasible solution is a 3 approximation. The authors also describe a significantly more complicated 2 approximation based on LP rounding which is the current best known upper bound for the problem.

The main result in this paper is a simple combinatorial algorithm which achieves a 2 approximation for the active time problem, matching the upper bound obtained by the LP rounding scheme described by Chang et. al. [3].

2 PRELIMINARIES

A job j is said to be *live* at slot t if $t \in [r_j, d_j]$. A slot is *open* if a job can be scheduled in it. It is *closed* otherwise. An open slot is *full* if there are g jobs assigned to it. It is *non-full* otherwise.

A feasible solution is given by a set of open time slots into which the jobs can be feasibly scheduled. Given a set of slots, we can find a feasible assignment of jobs or determine that no schedule is possible by performing a simple flow computation (described in the appendix).

3 GREEDY ALGORITHM

All time slots are assumed to be open initially. Consider time slots from left to right. At a given time slot, close the slot and check if a feasible schedule exists in the open slots. If so, leave the slot closed, otherwise, open it again. Continue to the next slot.

THEOREM 3.1. *The greedy algorithm described above gives a 2 approximation to the active time problem.*

The remainder of this section is devoted to proving Theorem 3.1. We will bound the number of full and non-full slots separately. Let S and S^* denote the final greedy and optimal schedules respectively. Let $|S|$ and $|S^*|$ denote the number of open slots in S and S^* respectively. We first left shift the job units in S as much as possible while maintaining feasibility. This is captured by the following lemma.

LEMMA 3.1. *For any job j in time slot t, j must be present in every non-full slot in the window of j earlier than t i.e in the interval $[r_j, t]$.*

PROOF. The proof follows from left shifting. For any non-full slot t' earlier than t in the window of job j, a unit of j must be present in t' since otherwise we would have left

---

[*] Supported by NSF REU Award CNS 156019

SPAA'18, July 2018, Vienna, Austria

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

shifted the unit from t into t' (this would be feasible since t' is in j's window and non-full). ☐

For the proofs of the remaining lemmas and the definitions of a, b, a^* and b^*, we assume that all job units have been left shifted as much as possible in S. Let $b_t[j]$ and $b_t^*[j]$ denote the number of units of any job $j \in J$ scheduled by S and S^* respectively at or before t i.e in time interval $[r_j, t]$. Let $a_t[j]$ and $a_t^*[j]$ denote the amount of job j scheduled by S and S^* respectively in the time interval $[t, d_j]$. So, $b_t[j] + a_{t+1}[j] = b_t^*[j] + a_{t+1}^*[j] = p_j$. Let T be the latest deadline of all the jobs.

Lemma 3.2. *For any non-full slot t opened by S, there must exist at least one job j scheduled by S in t such that $b_t^*[j] \geq b_t[j]$.*

Proof. If possible, suppose $b_t^*[j] < b_t[j]$ for all j scheduled by S in t (as depicted in Figure 1). While moving left to right in our greedy algorithm, we would encounter t. At this point, by definition, we have already scheduled $b_t[j]$ of each job in $[1, t]$. We still need to schedule $a_{t+1}[j]$ of each job j in the interval $[t + 1, T]$.

Now, if we were to close t, then we would need to feasibly schedule the following in the interval $[t + 1, T]$:

(1) $a_{t+1}[j] + 1$ units[1] of each j scheduled by S in t.
 By our assumption, since $b_t^*[j] < b_t[j]$ we have $a_{t+1}^*[j] > a_{t+1}[j]$ and so $a_{t+1}[j] + 1 \leq a_{t+1}^*[j]$.

(2) $a_{t+1}[j]$ units of each j live at t but not scheduled by S in t.
 Since j is not scheduled in t, all units of j must have been scheduled by S earlier than t since otherwise we could have left shifted j into t as it is non-full[2]. Therefore, $b_t[j] = p_j$ and $a_{t+1}[j] = 0$. So $a_{t+1}[j] \leq a_{t+1}^*[j]$.

(3) $a_{t+1}[j]$ units of each j with $r_j > t$.
 Clearly $a_{t+1}[j] = p_j = a_{t+1}^*[j]$. So $a_{t+1}[j] \leq a_{t+1}^*[j]$.

It can be seen that the mass of each job j that ALG would need to schedule in $[t + 1, T]$ (either $a_{t+1}[j]$ or $a_{t+1}[j] + 1$ units) is less than or equal to the mass of that job that OPT feasibly schedules in that interval ($a_{t+1}^*[j]$ units). When moving from left to right in our algorithm, when we reached t, all the slots in $[t + 1, T]$ were open to schedule jobs. This means that, had we closed t in S, we would still have been able to find a feasible schedule of the remaining job units in $[t + 1, T]$, since OPT could find an optimal schedule for them in $[t + 1, T]$. Therefore, we would have closed t greedily while constructing S. Since we did not, our original assumption must have been incorrect. ☐

Lemma 3.3. *The number of non-full slots in S cannot exceed $|S^*|$.*

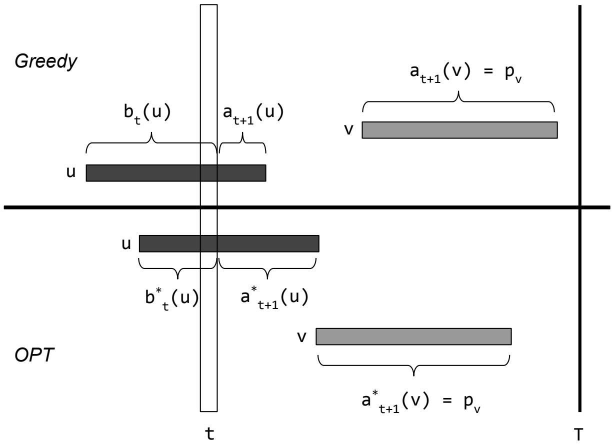

Figure 1: The top half depicts S and the bottom half S^*. Job u is scheduled by S in t such that $b_t^*[u] < b_t[u]$. If this was true for all such jobs u scheduled by S in t, then in $[t + 1, T]$, S^* would schedule as much as or more of every job that S would have scheduled there even after closing t.

Proof. Start at the right most non-full slot in S, say t. From Lemma 3.2, we can find one job j in t such that $b_t^*[j] \geq b_t[j]$. By Lemma 3.1, j must be present in every non-full slot in $[r_j, t]$. This means that the number of non-full slots in $[r_j, t]$ cannot exceed $b_t[j]$ ($\leq b_t^*[j]$). So we can charge every non-full slot of S in $[r_j, t]$ to a distinct slot in S^* in $[r_j, t]$. Now, move to the latest non-full slot opened by S strictly earlier than r_j and repeat this process. In this way, we can charge every non-full slot in S to distinct slots in S^*. ☐

Lemma 3.4. *The number of full slots in S cannot exceed $|S^*|$.*

Proof. Let the number of full slots in S be $|S_f|$. Since the maximum amount of job mass in any slot is g, the amount of job mass present in S_f is $g|S_f|$. Similarly, the total job mass OPT scheduled is at most $g|S^*|$. By the conservation of job mass, $g|S_f| \leq g|S^*|$ and the lemma follows. ☐

The total cost of our schedule is the sum of the full and non-full slots, and therefore, from Lemmas 3.3 and 3.4, this sum cannot exceed $2|S^*|$. This proves Theorem 3.1.

4 CONCLUSION

In this paper, we prove that a simple greedy algorithm matches the best known approximation ratio for the active time problem.

Crucially, the complexity status of this problem is still open as is breaking the 2 upper bound barrier. A possible avenue to achieving this is via a local search technique which we briefly sketch in the appendix.

---

[1] The extra unit comes from the slot t which we are attempting to close.

[2] Here, we crucially use the fact that t is non-full. If t was full, this point may not have been true since the left shifting argument would not hold, and the lemma breaks down.

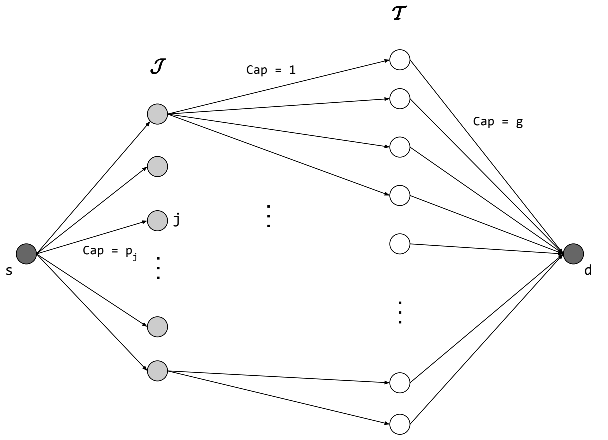

Figure 2: Flow network $G_{\mathbf{feas}}$. An integral flow of value $\sum_{j \in J} p_j$ corresponds to a feasible schedule.

REFERENCES

[1] Hans L Bodlaender and Klaus Jansen. 1995. Restrictions of graph partition problems. Part I. *Theoretical Computer Science* 148, 1 (1995), 93–109.

[2] Jessica Chang, Harold N Gabow, and Samir Khuller. 2012. A model for minimizing active processor time. In *European Symposium on Algorithms*. Springer, 289–300.

[3] Jessica Chang, Samir Khuller, and Koyel Mukherjee. 2014. LP rounding and combinatorial algorithms for minimizing active and busy time. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*. ACM, 118–127.

[4] Guy Even, Retsef Levi, Dror Rawitz, Baruch Schieber, Shimon Moni Shahar, and Maxim Sviridenko. 2008. Algorithms for capacitated rectangle stabbing and lot sizing with joint set-up costs. *ACM Transactions on Algorithms (TALG)* 4, 3 (2008), 34.

[5] Saurabh Kumar. 2016. *Combinatorial Algorithms for the Active Time and Busy Time Problems.* Master's thesis. University of Maryland, College Park.

[6] Christos H. Papadimitriou and Mihalis Yannakakis. 1979. Scheduling interval-ordered tasks. *SIAM J. Comput.* 8, 3 (1979), 405–409.

A APPENDIX

A.1 Verifying a feasible schedule exists

Define a graph G with vertex set consisting of one node for every job j, one node for every *open* time slot t and a source and destination node (s and d respectively). Add edges from s to each job node j with capacity p_j. Add edges from each open time slot node t to d with capacity g. For each job j, for any time slot t in its window, add an edge from job node j to time slot node t with unit capacity. The graph structure is shown in Figure 2. An active time instance has a feasible schedule on the set of open time slots iff the maximum flow from s to d has value $\sum_{j \in J} p_j$. Furthermore, if a feasible schedule is possible, the unit capacity edges with non-zero flow give the mapping of job units to time slots.

A.2 Tight Example

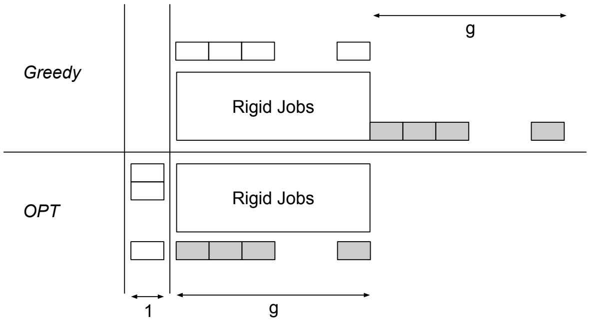

Figure 3: Tight Example for the Greedy Algorithm.

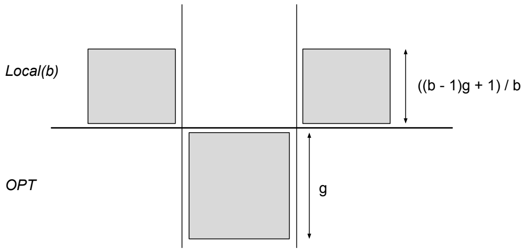

Figure 4: Lower bound for Local Search with parameter b.

The tight example consists of the following set of jobs - one job of length g with window $[1, 2g]$, g unit length jobs with window $[1, g+1]$ and $g-1$ rigid jobs of length g with window $[2, g+1]$. OPT would have opened time slot $t = 1$, scheduled all unit jobs there and therefore been able to schedule the g length job above the rigid jobs. This gives a total cost of $g+1$. However, our greedy algorithm closes time slot $t = 1$ since that is still feasible. Therefore, the unit jobs are forced to be scheduled above the rigid job, thereby pushing the long job out. This gives a total cost of $2g$. Thus, we get a lower bound of $\frac{2g}{g+1}$ which equals 2 as g becomes large. The two schedules are depicted in Figure 3 (reprinted from [5]).

A.3 Local Search

A possible approach to breaking the 2 barrier for this problem is local search. Local search parametrized by a constant b involves repeatedly performing local optimizations of the form - close b open slots and open at most $b-1$ new slots. We believe that this could provide a PTAS for this problem. Indeed, the best lower bound we currently have for local search is $1+1/(b-1)$. This is illustrated in Figure 4 (reprinted from [5]). Here, each column has $g-(g-1)/b$ job mass in it (where g is the capacity of the time slot) so that if we take any b columns, the total job mass amounts to $(b-1)g+1$ which clearly cannot be scheduled in at most $b-1$ slots. This gives a lower bound of $g/(g-(g-1)/b)$ which tends to $1+1/(b-1)$ as g becomes very large.