The Price of Bounded Preemption

Noga Alon*
Tel-Aviv University
Tel-Aviv, Israel
Princeton University
Princeton, New Jersey, USA
nogaa@post.tau.ac.il

Yossi Azar†
Tel-Aviv University
Tel-Aviv, Israel
azar@tau.ac.il

Mark Berlin
Tel-Aviv University
Tel-Aviv, Israel
markberlin@mail.tau.ac.il

ABSTRACT

In this paper we provide a tight bound for the *price of preemption* for scheduling jobs on a single machine (or multiple machines). The input consists of a set of jobs to be scheduled and of an integer parameter $k \geq 1$. Each job has a release time, deadline, length (also called processing time) and value associated with it. The goal is to feasibly schedule a subset of the jobs so that their total value is maximal; while preemption of a job is permitted, a job may be preempted no more than k times. The price of preemption is the worst possible (i.e., largest) ratio of the optimal non-bounded-preemptive scheduling to the optimal k-bounded-preemptive scheduling.

Our results show that allowing at most k preemptions suffices to guarantee a $\Theta(\min\{\log_{k+1} n, \log_{k+1} P\})$ fraction of the total value achieved when the number of preemptions is unrestricted (where n is the number of the jobs and P the ratio of the maximal length to the minimal length), giving us an upper bound for the price; a specific scenario serves to prove the tightness of this bound. We further show that when no preemptions are permitted at all (i.e., $k = 0$), the price is $\Theta(\min\{n, \log P\})$.

As part of the proof, we introduce the notion of the *Bounded-Degree Ancestor-Independent Sub-Forest (BAS)*. We investigate the problem of computing the maximal-value BAS of a given forest and give a tight bound for the loss factor, which is $\Theta(\log_{k+1} n)$ as well, where n is the size of the original forest and k is the bound on the degree of the sub-forest.

KEYWORDS

scheduling jobs; multiple machines; bounded preemptions; bounded-degree sub-forest

ACM Reference Format:
Noga Alon, Yossi Azar, and Mark Berlin. 2018. The Price of Bounded Preemption. In *Proceedings of 30th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'18).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3210377.3210407

*Supported by the ISF and the GIF
†Supported by: Israel Science Foundation Award 1506/16, ICRC Blavatnik Fund

SPAA'18, July 16–18, 2018, Vienna, Austria

ACM ISBN 978-1-4503-5799-9/18/07…$15.00
https://doi.org/10.1145/3210377.3210407

1 INTRODUCTION

1.1 Background

We study the bounded preemption real-time scheduling problem in an off-line setting, modelled as follows. There is a set J of n jobs available for scheduling, each of the jobs is associated with a triplet $\langle r_j, d_j, p_j \rangle$, denoting its release time, deadline and length (also called processing time), respectively, and with a value val(j). In addition, there is an integer $k > 0$. The jobs can be scheduled on a single machine as well as on multiple machines (for which we consider the migrative and non-migrative cases alike).

A *feasible k-bounded-preemptive schedule* is a scheduling of the jobs for execution in a way that at most one job is executed in every moment, the time constraints are not violated and that no job is preempted (i.e., its execution is stopped) more than k times. The objective is to provide a feasible (in the sense defined in the last sentence) schedule for a subset of the jobs of maximum total value.

We want to investigate the *price* of bounding preemption to k, which is defined as PoBP $\triangleq \sup_J \frac{\mathrm{OPT}_\infty(J)}{\mathrm{OPT}_k(J)}$, i.e. the amount of value we lose by limiting the preemptions to k with respect to the optimal ∞-preemptive scheduling. Also note that Lawler [21] has already found an optimal algorithm for the unbounded case on a single machine, which in turn can be extended to yield a constant factor approximation for the multiple identical machines setting; therefore, the price of bounded preemption and the approximation factor for the problem can be related to each other.

1.2 Motivation

The (NP-hard) problem of unbounded preemptive scheduling on multiple identical machines (denoted $P \,|\, pmtn, r_j \,|\, \sum_j (1 - U_j) W_j$, using the notation of [17]) was first analyzed by Lawler in [21], who found for the single-machine case an optimal solution in pseudo-polynomial time, which becomes polynomial for unit-value jobs. This (single-machine) solution can be transformed into an FPTAS using a technique developed by Pruhs and Woeginger [25].

This result can in turn be extended for multiple (identical) machines in the following way. First, one can use the same FPTAS iteratively on the residual jobs set [2], thus obtaining a $(2 + \epsilon)$-approximation for the non-migrative case. Then, this result can be further extended to the migrative case using the result of [18], thus obtaining a $(6 + \epsilon)$-approximation.

However, in a real-world setting, preemption comes with a certain price tag (e.g., the sequence of operations required for a context switch), and so one would want to somehow limit its usage: hence the motivation to study the bounded preemption setting.

The *bounded* version of the problem was addressed using several different approaches (e.g., [11, 12, 27]; surveyed in [13]). The approach consisting of bounding the number of preemptions per job, which is also the approach studied in this paper, was first addressed by Albagli-Kim et al. [1], who showed an $O(1)$-approximation for the special cases of unit-value and unit-density.

Other approaches to the notion of the power of preemption (albeit without regard to specific k) were suggested in [26] (defining it in terms of makespan) and [16] (defining it in terms of cost).

1.3 Our Results

Our paper provides both lower and upper bounds for the price of k-bounded preemption in terms of $n \triangleq |J|$ (i.e., the number of jobs) and of $P \triangleq \frac{\max_j p_j}{\min_j p_j}$.

- First, we explore the problem of the k-Bounded Degree Ancestor-Independent Sub-Forest (k-BAS) and find:
  - an optimal solution using Dynamic Programming;
  - a $\Theta(\log_{k+1} n)$ bound for the loss factor yielded by the optimal solution.
- We then proceed to bound the price of bounded preemption and achieve the following:
  - an upper bound of $PoBP_k = O(\log_{k+1} n)$, for single and multiple machines alike;
  - an upper bound of $PoBP_k = O(\log_{k+1} P)$, for single and multiple machines alike;
  - these above bounds are proven to be tight.
- We also show that for the special case of $k = 0$, i.e. when no preemptions are allowed for the machine, but an unbounded number of preemptions is allowed for a hypothetical competitor. We establish $PoBP_k = \Theta(\min\{n, \log P\})$ for this case.

1.4 Other Related Work

Recall that Lawler's solution for the non-bounded preemption problem on a single machine [21] can be transformed into an FPTAS, which in turn can be extended to a $(2+\epsilon)$-approximation for multiple non-migrative machines and to a $(6+\epsilon)$-approximation for multiple migrative machines (as described in subsection 1.2). Also, Baptiste et al. [5, 6] gave a polynomial algorithm for the special case of non-bounded preemption on a single machine with uniform processing time.

Recall also that Albagli-Kim et al. [1] established an $O(1)$-approximation (and price) for both the unit-value and the unit-density sub-problems. Their results can be easily extended, by Classify-and-Select, to yield $O(\log \rho)$ and $O(\log \sigma)$ approximations (and price) for the general bounded problem, respectively, where $\rho \triangleq \frac{\max_j \text{val}(j)}{\min_j \text{val}(j)}$, $\sigma \triangleq \frac{\max_j(\text{val}(j)/p_j)}{\min_j(\text{val}(j)/p_j)}$. However, their work gave no result with respect to either n or P, a gap which our work attends to.

The on-line version of the problem with no bound on the preemption was analyzed by [14], whose upper bound for the competitiveness ratio was later proven to be tight by [3].

The non-preemptive real-time scheduling problem has been posited and studied already several decades ago, and has various flavours, several of which have also been solved since. Moore devised an $O(n \log n)$-time algorithm [24] solving the simplest version of the problem, maximizing the number of jobs meeting their deadline (equivalent to minimizing the number of late jobs), where all the jobs are unit-value and have the same release time. Later, Lawler [20] extended this technique for special sub-cases of the problem of weighted jobs with the same release time (known to be NP-hard in general by [19]), and again [22] – for special sub-cases of the problem of unit-value jobs with different release times (known to be NP-hard in general by [21]). Also, Lawler and Moore devised [23] an $O(n \sum p_j)$-time (pseudopolynomial) algorithm for maximizing the value of jobs meeting the deadline with the same release time.

Bar-Noy et al. [7, 8] showed a constant factor approximation for maximizing the value of jobs meeting their deadline with different release times; other approximations in this setting have been suggested prior to that, particularly [15].

A different approach was used by Bar-Yehuda et al. in [10], who consider the scheduling of jobs already divided into k intervals which can be scheduled only at specific points in time. Using the *local ratio* technique first presented in [9], they achieve an $O(k)$ approximation for the problem itself and the APX-hard *Max-Weight Independent Set* (MWIS) problem to which it can be reduced.

2 PRELIMINARIES

2.1 Statement of Main Problem

Let J be a set of *n jobs* that should be scheduled for processing on a single machine (multiple non-migrating machines), and $k > 0$ an integer. Each of the jobs $j \in J$ has a *value* $\text{val}(j) > 0$, a *release time* r_j, a *deadline* d_j and a *length* p_j.

Definition 2.1.

(a) A *feasible schedule of a job on a single machine* $j \in J$ is a set G_j of pairwise-disjoint segments such that $\forall g \in G_j : g \subseteq [r_j, d_j]$ and $\sum_{g \in G_j} |g| = p_j$.

(b) A *feasible schedule of a job set* J *on a single machine* is a set of feasible schedules of jobs $\mathcal{G}_J = \{G_j | j \in J\}$, s.t. $\forall j_1, j_2 \in J, j_1 \neq j_2 : \forall g_1 \in G_{j_1}, g_2 \in G_{j_2} : g_1 \cap g_2 = \emptyset$.

(c) A *k-preemptive feasible schedule of a job set* J *on a single machine* is a feasible schedule $\mathcal{G}_J$ s.t. $\forall j \in J, |G_j| \leq k+1$, i.e. no job is preempted more than k times.

The above definitions can trivially be extended to multiple **non-migrating** machines by making the feasible schedule for multiple machines a union of feasible schedules on single machine, s.t. no job is scheduled twice on different machines.

Our objective is, given a set of jobs J, to find $J' \subseteq J$ and a k-preemptive feasible schedule $\mathcal{G}_{J'}$ s.t. $\text{val}(J') \triangleq \sum_{j \in J'} \text{val}(j)$ is maximal.

2.2 Useful Shorthands

We will denote the i-th segment of a job j as $g_j^i = [s_j^i, t_j^i]$. Also, we will define the relation of *precedence* between any two segments $g_{j_1}^{i_1} = [s_{j_1}^{i_1}, t_{j_1}^{i_1}], g_{j_2}^{i_2} = [s_{j_2}^{i_2}, t_{j_2}^{i_2}]$ as follows: $g_{j_1}^{i_1} \prec g_{j_2}^{i_2} \iff t_{j_1}^{i_1} \leq s_{j_2}^{i_2}$. Note, that since all the segments are disjoint, this relation induces a total order on the segments.

3 BOUNDED-DEGREE ANCESTOR-INDEPENDENT SUB-FOREST

3.1 Statement of Problem

In order to solve the k-bounded preemption problem we first reduce it to the following problem of the k-Bounded Degree Ancestor-Independent Sub-Forest (k-BAS). In this section, we denote by $T(u)$ the sub-tree of a tree T which is rooted in the node $u \in T$ and by $C_T(u)$ the set of the children nodes of a tree node u. Also, the *degree* of a node u in T is defined to be the number of u's children in T, i.e. $\deg_T(u) \triangleq |C_T(u)|$.

Definition 3.1. An *Ancestor-Independent Sub-Forest* (AISF) of a given forest (tree) $T(V, E)$ is a sub-forest T' of T s.t. for any two different connected components $C_1 \neq C_2$ of T', for any $v_1 \in C_1, v_2 \in C_2$, v_1 is not a descendant of v_2 with respect to E.

Definition 3.2. A *k-Bounded-Degree Ancestor-Independent Sub-Forest* (k-BAS) of a given forest (tree) $T(V, E)$ is an Ancestor-Independent Sub-Forest T' of T, s.t. $\forall v \in V', \deg_{T'}(v) \leq k$.

Definition 3.3. An *optimal k-BAS* of a given forest (tree) $T(V, E)$ with values $\text{val} : V \to \mathbb{R}^+$ is a k-BAS T' of T s.t. $\text{val}(V') \triangleq \sum_{v \in V'} \text{val}(v)$ is maximum.

Naturally, a k-BAS of a tree does not equal the whole tree in the general case, but has less nodes and hence less total value. We will formalize this through the notion of the *loss factor*.

Definition 3.4. The *loss factor* α_{ALG} of an algorithm ALG for the computation of a k-BAS of a tree (forest) is the ratio $\alpha_{\text{ALG}} \triangleq \sup_T \frac{\text{val}(T)}{\text{val}(\text{ALG}(T))}$.

Observation 3.5. *A max-value k-BAS of a forest T is the union of the max-value k-BASs of the trees constituting T.*

Corollary 3.6. *A loss factor α for an algorithm computing a k-BAS of a tree implies the same loss factor for the same algorithm applied to a forest.*

In the following sub-sections we will present an optimal algorithm for the computation of a k-BAS of a tree and bound its loss factor. By the above reasoning, this loss factor will hold for forests as well.

3.2 Dynamic Programming Solution

Trivially, in order to turn a given tree (of arbitrary degree) into a k-BAS, several of the nodes need to be deleted.

Lemma 3.7. *Let $T'(V', E')$ be an ancestor-independent sub-forest of a forest (tree) $T(V, E)$. Then $v \in V \setminus V'$ iff all of its ancestors or all of its descendants are in $V \setminus V'$.*

Proof. Assume on the contrary that there is a node $v \in V \setminus V'$ s.t. both an ancestor of it u and a descendant of it w are V'. Since T is a forest, and v is deleted, it follows that u, w belong to two different connected components of T'. But w is a descendant of u, contrary to the definition of an AISF. ∎

We therefore can divide the nodes of the tree into three groups as follows: (1) *retained* nodes, i.e. nodes that are retained in the k-BAS (however, some of their descendants might be deleted); (2) *pruned-up* nodes, i.e. nodes that are deleted with all their ancestors up to the root (in order to fulfil Ancestor Independence); (3) *pruned-down* nodes, i.e. nodes that are deleted with all their descendants.

Observation 3.8. *For any k-BAS:*

(a) A retained *node cannot have* pruned-up *descendants.*
(b) A pruned-up *node can have descendants of all types.*
(c) A pruned-down *node has only* pruned-down *descendants.*

For a node v, let us denote its *aggregate value* (i.e., the maximal possible value that can be salvaged from $T(v)$ after the pruning) as $t(v)$ in case v is *retained*, and as $m(v)$ in case it is *pruned-up* (there is no need to introduce a notation for the aggregate value of v in case it is *pruned-down*, since in that case it is discarded). Also, let us denote as $C_k(v)$ the nodes in $C(v)$ with k highest $t(v)$. Observation 3.8 leads us to the following formula:

$$\begin{cases} t(u) = \text{val}(u) + \sum_{v_i \in C_k(u)} t(v_i) \\ m(u) = \sum_{v_i \in C(u)} \max(t(v_i), m(v_i)) \end{cases} \tag{3.1}$$

In order to compute the k-BAS of a tree rooted in u, we need to decide for each node in this tree whether it is better for it to be *retained* or *pruned-up* (it may, of course, so happen that a node is better *pruned-down* with all its value in order to allow for its parent node to be *retained*). This is done by computing $\langle t(u), m(u) \rangle$ using bottom-up traversal and then deciding which of them is better and applying the decision in reverse order for all its descendants. This scheme is, essentially, Dynamic Programming. For simplicity, the algorithm will be presented for a tree as an input, but it can be trivially extended to a forest by applying it separately to each of the trees of which the forest consists.

Procedure TM

Input: A tree $T(V, E)$ rooted in u.
Output: Two numbers, $t(u), m(u)$, being the maximal values attained for u being designated *retained* or *removed*, respectively.

```
1  if u is leaf then
2      t(u) ← val(u) ;
3      m(u) ← 0 ;
4  else
5      foreach v_i ∈ C(u) do
6          (t(v_i), m(v_i)) ← TM(v_i) ;
7      end foreach
8      t(u) ← val(u) + Σ_{v_i ∈ C_k(u)} t(v_i) ;
9      m(u) ← Σ_{v_i ∈ C(u)} max(t(v_i), m(v_i)) ;
10 end if
11 return t(u), m(u);
```

Analysis. The runtime of the algorithm is trivially $O(|V|)$. Since the algorithm implements exactly the formulae presented in equation 3.1, it is optimal, i.e. gives the maximal value k-BAS possible for a given tree (forest).

3.3 Determining the Loss Factor

The analysis of the quality of the loss factor directly from the TM algorithm is difficult. We will therefore look at a different algorithm, which is easier to analyze and which we will call the algorithm of *levelled contraction*. Using it, we will prove the following theorem:

Theorem 3.9. *The loss factor of the algorithm* TM *is at most* $\log_{k+1} n$.

We will make use of the simple observation, that a sub-tree with a degree k can be viewed as a single large node, since from the point of view of the choice which node to retain there is no need to look at each of them individually, but all of them can be accommodated *en bloc*. Formally:

Definition 3.10. A node u of a tree T is *k-contractible* iff: **either** (i) it is a leaf; **or** (ii) it has no more than k children, all which are *contractible* as well.

Definition 3.11. The *k-contraction* of a k-contractible node u in a tree T with values $\text{val} : T \to \mathbb{R}^+$ is deleting $T(u)$ from the tree and setting $\text{val}(u) \leftarrow \text{val}(T(u))$.

Observation 3.12. *Let u be a k-contractible node in a forest $T(V, E)$ with values assigned to the nodes. Then the sub-tree $T(u)$ can be contracted to a single (leaf) job without loss of value.*

The contraction itself is done by a bottom-up marking of the nodes as *contractible*. The algorithm is described in the procedure MaxContract, which has the following properties:

Observation 3.13. *After the application of* MaxContract *to a tree, all non-leaf nodes have more than k descendants; meaning that the contraction is* maximal *(i.e., there are no more k-contractible nodes).*

Observation 3.14. *A contraction (in particular, the application of* MaxContract*) cannot increase the number of leafs.*

If a node u is not contractible, the number of leafs of $T(u)$ remains the same. If u is contractible, then all the leafs of $T(u)$ are replaced by u as a new leaf; but $T(u)$ trivially has at least one leaf. Thus, in either case the number of leafs does not increase.

The algorithm LevelledContraction uses MaxContract as a primitive in an iterative fashion: in each iteration it performs a maximal contraction, takes the set of leafs aside and continues to the next one. In the end, the algorithm returns the set of leafs with the maximal value. Each of such sets represents a k-BAS of the original tree, as shown in the following lemma.

Observation 3.15. *Let S_i be the set of leafs S after the contraction at the i-th iteration. Then at the beginning of the i-th iteration all the nodes $w_i^j \in S_i$ are k-contractible.*

Lemma 3.16. *Let $S_i = \left\{w_i^1, w_i^2, \dots w_i^{|S_i|}\right\}$ be the set of leafs S after the contraction at the i-th iteration, and let T_i^j be the sub-forest of T rooted in s_i^j in the beginning of that iteration (which is contracted during it). Then the union $T_i = \bigcup_{j=1}^{|S_i|} T_i^j$ is a k-BAS.*

Proof. T_i is trivially a sub-forest of T. Thus, in order to prove it is a k-BAS of T, we need to prove that its degree is bounded by k and that it is ancestor-independent.

Algorithm 1: Levelled Contraction

```
1  Procedure MaxContract()
       Input: A tree T.
       Output: The tree T after maximal k-contraction.
2      foreach node u of T in bottom-up order do
3          if u is a leaf then mark u as true ;
4          else if deg(u) ≤ k and all children of u are marked
             true then
5              contract children of u ;
6              set val(u) ← Σ_{v_i ∈ CN(u)} val(v_i) ;
7              mark u as true ;
8          else mark u as false ;
9      end foreach
10 end
11 Procedure LevelledContraction()
       Input: A tree T.
       Output: A k-BAS S of T.
12     S ← ∅ ;
13     while T ≠ ∅ do
14         MaxContract(T) ;
15         S ← all leafs of T ;
16         S ← S ∪ {S} ;
17         T ← T \ S ;
18     end while
19     return argmax_{S∈S} val(S) ;
20 end
```

Bounded Degree. From observation 3.15 immediately follows that the degree of each T_i^j is at most k, and therefore the same holds for their union T_i.

Ancestor Independence. Assume on the contrary that there exist $u_1 \in T_i^{j_1}, u_2 \in T_i^{j_2}$ s.t. u_1 is ancestor of u_2 in the original T. It therefore means that $w_i^{j_1}$ is ancestor of u_2 as well. However, u_2 was not contracted into $w_i^{j_1}$; therefore, on the (single) path from u_2 to $w_i^{j_1}$ there is a node with a degree higher than $k+1$. This contradicts the fact that $w_i^{j_1}$ is k-contractible, by observation 3.15. ∎

Lemma 3.17. *The value returned by* LevelledContraction *is not less than $\frac{\text{val}(T)}{L}$, where L is the number of iterations.*

Proof. Let S_i be the set of leafs S at the i-th iteration, and S_i' the set of their parents. Also, let T_i be the k-BAS of T corresponding to S_i, as explained in lemma 3.16. Since the algorithm only performs contractions, by observation 3.12, no value is lost, i.e., $\text{val}(T) = \sum_{i=1}^{L} \text{val}(T_i) = \sum_{i=1}^{L} \text{val}(S_i)$. Therefore, the value returned by the algorithm is

$$\text{ALG} \geq \frac{\text{val}(T)}{L} \quad . \tag{3.2}$$

∎

Lemma 3.18. *The number of iterations of* LevelledContraction *is less than* $\log_{k+1} n$.

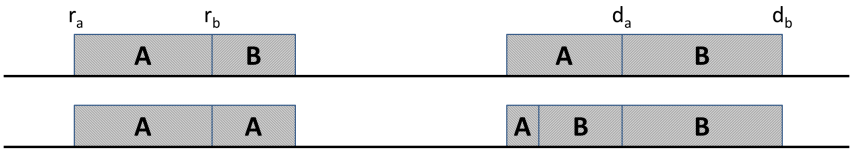

Figure 1: Rearranging a Lawler Schedule

Proof. Due to observation 3.13, $|S_i| \geq (k+1)\cdot|S'_i|$. After the removal of S_i from the tree in the iteration's end, S'_i becomes the set of the tree's leafs. By observation 3.14, a contraction cannot increase the number of leafs in a tree, and therefore $|S_{i+1}| \leq |S'_i|$. Combining these, we get that for every iteration i,

$$|S_{i+1}| \leq |S'_i| \leq \frac{1}{k+1} \cdot |S_i| \Rightarrow 1 = |S_L| \leq \frac{1}{(k+1)^{L-i}} \cdot |S_i|$$

where L is the (maximum) number of iterations. Hence follows:

$$n \geq \sum_{i=1}^{L} |S_i| \geq \sum_{i=1}^{L} (k+1)^{L-i} = \frac{(k+1)^L - 1}{k} \geq (k+1)^{L-1} - 1$$

which in turn implies

$$L \leq \log_{k+1} n$$

Combining this result with equation 3.2 yields us:

$$\text{ALG} \geq \frac{\text{val}(T)}{L} \geq \frac{\text{val}(T)}{\log_{k+1} n}$$

■

Corollary 3.19. *The loss factor of* `LevelledContraction` *is not larger than* $\log_{k+1} n$.

Proof of Theorem 3.9. Since algorithm `TM` is optimal, its loss factor cannot be worse than the loss factor of `LevelledContraction`. Therefore, it is at most $\log_{k+1} n$, i.e. the algorithm `TM` (T) yields a k-BAS T' such that $\text{val}(T') \geq \frac{1}{\log_{k+1} n} \cdot \text{val}(T)$. ■

Remark. The runtime of the `LevelledContraction` algorithm is trivially $O(|V|)$.

3.4 Tightness of Bound

Theorem 3.20. *The loss factor of the* `TM` *algorithm for the k-BAS problem in the general case is at least* $\log_{k+1} n$*, i.e. the loss factor established previously is tight.*

The proof of this lemma is presented in appendix A. ■

4 THE MAIN PROBLEM

4.1 Reduction to k-BAS

Assume we are given an optimal solution for the case $k = \infty$. Observe, that if in a feasible schedule for a single machine segments of two jobs A and B interleave, i.e. $\exists g_A^{a_1}, g_A^{a_2} \in G_A, g_B^{b_1}, g_B^{b_2} \in G_B$ s.t. $g_A^{a_1} \prec g_B^{b_1} \prec g_A^{a_2} \prec g_B^{b_2}$, they can be re-arranged into $g_A^{a_1} \prec g_A^{a_2} \prec g_B^{b_1} \prec g_B^{b_2}$ without affecting the feasibility. This of course can be done independently for each machine in the multiple machine setting (if the machines are non-migrative). It therefore follows that any schedule (in particular, an optimal one) can be re-arranged (with no loss of value) into a form where a segment of B is between two segments of A iff there are no segments of A between any two of the segments of B.

Note, that the described re-arrangement makes the *preempts* relation *laminar*. If we now present such a re-arranged schedule $\mathcal{G}_J$ as a graph $T(V, E)$, where the set of nodes $V = J$ is the set of jobs in the schedule, and the set of edges $E = \{(u, v) \in V \times V \mid v \text{ preempts } u\}$, the resulting (directed) graph is therefore a forest, which be will call a *Schedule Forest*. Let $T'(V', E')$ be a max-value k-BAS of T, computed using the algorithm `TM` described in subsection 3.2.

We will now describe the way of obtaining a new schedule $\mathcal{G}'_{J'}$ from G'; also, for the sake of simplicity, we will call the child nodes of a node (job) j in T the *sub-jobs* of j. The set of jobs J' in the new schedule corresponds directly to V'. For each retained job $j \in V'$, its segments between two consecutive sub-jobs remain as is. If, however, a sub-job of j is removed (pruned-down – see observation 3.8), the segments of j surrounding it are merged to the left. This left-merge may affect several segments at once if several consecutive sub-jobs are removed.

Lemma 4.1. *The schedule $\mathcal{G}'_{J'}$, obtained as described from an original unbounded-preemption schedule $\mathcal{G}_J$ is a feasible k-bounded schedule as defined in definition 2.1.*

Proof.
Feasibility of Individual Job Schedules. The feasibility of the individual job schedules $G'_j \in \mathcal{G}_{J'}$ follows from the feasibility of the original $G_j \in \mathcal{G}_J$, as the segments of the jobs are either retained as is or merged to the left, i.e., earlier than the deadline, but nevertheless not earlier than the release time. Also, since segments are removed only as part of the removal of a job as a whole, the length constraint is kept as well.
Feasibility of the Schedule Overall. The unchanged segments do not intersect each other by feasibility of the original $\mathcal{G}_J$. The merged segments replace sub-jobs which are pruned-down, i.e. removed with all their descendants and therefore occupy a place which would otherwise be empty.
Preemption Bound. The segments of any job j are scheduled so that between any two of them there is a sub-job of j (because of the left-merge). Since by the definition of a k-BAS each $j \in J'$ has no more than k sub-jobs, the number of segments of each job j is not more than $k + 1$. ■

Remark. In the multiple non-migrative machines setting the reduction can be applied to each machine separately. The forests obtained per machine are simply merged to a single forest. Since migration can be eliminated by using 6 times more machines [18], if we keep the number of machines constant, we retain at least the $\frac{1}{6}$-th part of the optimal solution after this elimination and prior to applying the

reduction. Thus all the prices achieved below using this reduction stay the same in terms of O notation.

Remark. The described reduction is **not** bidirectional, i.e. the optimal solution for the k-BAS problem is not necessarily optimal for the corresponding k-Bounded Preemptive Schedule problem. This due to the problem of adequately translating a case where a single job A is preempted by a string of successive jobs. In this case A is preempted only once, however if each of these other jobs is in turn preempted, say, $k+1$ times, we are forced to represent it in the tree as if A were preempted more than once (modelling it as an infinitely small segment of A between each of the jobs preempting it in succession), which may force us to forgo some of these jobs, which of course is not necessary in the schedule problem and thus the reverse reduction might yield a non-optimal solution. However, the reduction does provide an upper bound for the loss factor.

4.2 The Price of Bounded Preemption as Function of the Number of the Jobs

Theorem 4.2. *The price of limiting preemption to k is at most* $\log_{k+1} n$.

Proof. Lemma 4.1 describes the transformation of an original optimal unbounded preemptive schedule $\mathcal{G}_J$ into a k-bounded preemption schedule $\mathcal{G}'_{J'}$. Trivially, the total value of the new set of jobs J' is equal to the total value of the V' of the max-value k-BAS T', which in turn is optimal (see subsection 3.2) and obeys theorem 3.9. Thus,

$$PoBP_k \triangleq \sup_J \frac{\mathrm{OPT}_\infty(J)}{\mathrm{OPT}_k(J)} \leq \frac{\mathrm{val}(J)}{\mathrm{val}(J')} = \frac{\mathrm{val}(V)}{\mathrm{val}(V')} \leq \log_{k+1} n$$

■

The following theorem establishes the tightness of this bound; the proof is presented in appendix B.

Theorem 4.3. $PoBP_k = \Theta(\log_{k+1} n)$.

4.3 The Price of Bounded Preemption as Function of the Length-Ratio

In order to analyze the Price of Bounded Preemption as a function of the length ratio P, we will first introduce the notion of *relative laxity* of jobs.

Definition 4.4. The *relative laxity* of a job $j \in J$ is the ratio $\lambda_j \triangleq \frac{d_j - r_j}{p_j}$. The maximal relative laxity in an instance of the problem is denoted $\lambda_{\max}$.

We will divide the set of jobs J into two by their relative laxity (i.e., $J_1 = \{j \in J | \lambda_j \leq k+1\}$, $J_2 = \{j \in J | \lambda_j > k+1\}$) and solve each case separately. We show that in both cases the price bound is $\log_{k+1} P$, and therefore it is $O(\log_{k+1} P)$ for the overall problem. In other words, we prove the following:

Theorem 4.5. *The price of limiting preemption to k is $O(\log_{k+1} P)$.*

4.3.1 Strict Jobs. Let us define the *window* of a scheduled job $j \in J_{in}$, denoted $w(j)$, as the time difference between its release time and its deadline. Trivially, $\forall j \in J, w(j) = p_j \cdot \lambda_j$.

Note that the window of a job necessarily includes the window of all its descendants in the original tree. Thus, if we look back at the proof of lemma 3.18 and define $W_i \triangleq \min_{j \in S_i} w(j)$, we have, by observation 3.13, that for every iteration i,

$$W_{i+1} \geq (k+1) \cdot W_i \Rightarrow p_{\max} \cdot \lambda_{\max} \geq W_L \geq (k+1)^{L-i} \cdot W_i$$

where L is the (maximum) number of iterations. Hence follows:

$$p_{\max} \cdot \lambda_{\max} \geq (k+1)^L \cdot W_0 \geq (k+1)^L \cdot p_{\min}$$
$$L \leq \log_{k+1}(P \cdot \lambda_{\max})$$

By lemma 3.17, this in turn implies:

Lemma 4.6. *For a tree T created from a schedule satisfying $\lambda_{\max} \leq k+1$, the k-BAS T' returned by algorithm* `Levelled-Contraction` *satisfies* $\mathrm{val}(T') \geq \frac{1}{\log_{k+1} P} \cdot \mathrm{val}(T)$. ■

4.3.2 Lax Jobs. The treatment of the lax jobs is slightly different and requires a different algorithm. It also requires the establishment of several auxiliary lemmas presented below.

Lemma 4.7. *Let S be a set of intervals. There exists a subset $S' \subseteq S$ such that each point in $\bigcup_{I \in S} I$ is covered by at least one and at most two different intervals in S'.*

Proof. Assume WLOG $\bigcup_{I \in S} I$ is a single segment (otherwise apply the proof separately for each segment and all intervals contained in that segment). Let I_0 be the interval whose left endpoint is the leftmost among all intervals in S. We start with $S' \leftarrow I_0$ and iteratively add intervals to S'. Let I_i be the interval whose right endpoint is the rightmost among all intervals in S that intersect an interval in S'. Add I_i to S', that is $S' \leftarrow S' \cup \{I_i\}$. Repeat this procedure until $\bigcup_{I \in S'} I = \bigcup_{I \in S} I$. ■

Corollary 4.8. *If we number the intervals in S' by the order of their leftmost coordinate and divide them into two disjoint subsets S'_E, S'_O by the parity of this numbering, then each of these subsets is a set of pairwise-disjoint intervals.*

Proof. Assume on the contrary WLOG that S'_E does not satisfy the above. Then $\exists k$ s.t. $I_{2k} \cap I_{2k+2} \neq \emptyset$. By construction, $I_{2k} \cap I_{2k+1} \neq \emptyset$ and $I_{2k+2} \cap I_{2k+1} \neq \emptyset$. However, since I_{2k+1} is contiguous, this necessary implies the existence of a point belonging to all three at once, in contradiction of the previous lemma. ■

The following lemma was proven in [4]:

Lemma 4.9. *Given a sequence $\{a_1, ..., a_n\}$, a non-increasing non-negative sequence $\{b_1, ..., b_n\}$ and two sets $X, Y \subseteq \{1, ..., n\}$, let $X^i = X \cap \{1, ..., i\}$ and $Y^i = Y \cap \{1, ..., i\}$. Then*

$$\forall 1 \leq i \leq n : \sum_{j \in X^i} a_j > \alpha \sum_{j \in Y^i} a_j \Longrightarrow \sum_{j \in X} a_j b_j > \alpha \sum_{j \in Y} a_j b_j$$

Let us now return to the main problem. Recall that we assume all the jobs are lax, specifically $\forall j \in J, \lambda_j > k+1$. We will prove the following:

Lemma 4.10. *For a given set of jobs J exclusively of jobs for which $\lambda_j \geq k+1$, the algorithm* `LSA_CS` *yields a feasible k-preemptive schedule $\mathcal{G}_{J_{in}}$ s.t.* $\mathrm{val}(J_{in}) \geq \frac{1}{6 \cdot \log_{k+1} P} \cdot \mathrm{val}(J)$.

In order to prove this theorem, we will present the Leftmost-Schedule Algorithm (LSA) suggested by [1], with the difference that the jobs will be sorted by their *density* $\sigma_j \triangleq \frac{\mathrm{val}(j)}{p_j}$ rather than by value. In the following analysis we will use the same terminology

Algorithm 2: The Leftmost Schedule Algorithm

Procedure LSA_CS()
  Input: A set of lax jobs J (i.e., satisfying $\forall j \in J, \lambda_j \geq k+1$).
  Output: A feasible schedule for jobs $J_{in} \subseteq J$, s.t. $\text{val}(J_{in}) \geq \frac{\text{val}(J_{\text{OPT}})}{6 \cdot \log_{k+1} P}$.
  Separate all jobs in J into classes $J_c, 1 \leq c \leq log_{k+1}P$ by length, s.t. $\forall\ 1 \leq c \leq log_{k+1}P$, $\forall\ j \in J_c,\ (k+1)^{c-1} \leq p_j \leq (k+1)^c$;
  foreach $1 \leq c \leq \log_{k+1} P$ **do**
    $\langle J_{c,in}, \mathcal{G}_{J_{c,in}} \rangle \leftarrow \text{LSA}(J_c)$;
  end foreach
  return $\langle J_{c^*,in}, \mathcal{G}_{J_{c^*,in}} \rangle$, s.t. $\text{val}(J_{c^*,in})$ is maximal among the $J_{c,in}$;
end

Procedure LSA()
  Input: A set of lax jobs J (satisfying $P \leq k+1$).
  Output: A feasible schedule for jobs $J_{in} \subseteq J$ (s.t. $\text{val}(J_{in}) \geq \frac{1}{6} \cdot \text{val}(J_{\text{OPT}})$).
  Sort J in descending order of the jobs *density* ;
  foreach $j \in J$ **do**
    Let S be the set of the leftmost $k+1$ *idle segments* in $[r_j, d_j]$;
    repeat
      if *j fits into the segments in S* **then**
        Schedule j in members of S in the leftmost possible way ;
        break;
      else
        Remove shortest segment from S and replace it with the next *idle segment* in $[r_j, d_j]$;
      end if
    until *all* idle segments *are exhausted*;
  end foreach
end

of *busy-segment* (a maximal subset g^* of the timeline s.t. $\exists j \in J : \exists g \in G_j : g^* \cap g \neq \emptyset$) and *idle-segment* (a maximal subset g^* of the timeline s.t. $\forall\ j \in\ J : \forall\ g \in\ G_j : g^*\ \cap\ g = \emptyset$). Also, for the sake of brevity, we introduce the notation $J_{out} \triangleq J \setminus J_{in}$ for the set of the jobs not scheduled.

Lemma 4.11. *The length of each busy-segment in the schedule created by* LSA *is at least the length of the shortest job.*

Proof is omitted and will be presented in the full paper. ■

Now let j be a discarded job. If it was discarded, it means that already at the moment of its discarding in the span of time $[r_j, d_j]$ which is at least $p_j \cdot (k+1)$ long there could be found no $k+1$ idle segments of total cumulative length p_j.

Let us denote the the number of the idle segments in $[r_j, d_j]$ as f. We will look at two distinct cases. Either (i) $f \leq k+1$; or (ii) $f > k+1$.

In case (i), since the total length of the busy segments and idle segments in $[r_j, d_j]$ is equal to $d_j - r_j \geq p_j \cdot (k+1)$, and the total length of (all) the idle segments is less than p_j, it necessarily means that the total length of the busy segments is more than $p_j \cdot k$, and so in the worst case we have that $\text{val}(J') \geq \frac{k}{k+1} \cdot \text{val}(J)$.

In case (ii), $f > k+1$, yet j could not be scheduled; this implies that the sum of the $k+1$ longest idle segments in $[r_j, d_j]$ is less than p_j. Therefore, if we denote the total cumulative length of all idle segments in $[r_j, d_j]$ as L_{idle}, it holds that $L_{idle} \leq \frac{f}{k+1} \cdot p_j$. On the other hand, the number of busy segments in $[r_j, d_j]$ is trivially at least $f-1$, making their total length $L_{busy} \geq f-1$ (by lemma 4.11). It holds that:

$$\left.\begin{aligned} L_{idle} &\leq \frac{f}{k+1} \cdot p_j \\ L_{busy} &\geq f-1 \end{aligned}\right\} \Rightarrow \frac{L_{idle}}{L_{busy}} \leq \frac{f}{f-1} \cdot \frac{p_j}{k+1} \leq \frac{2P}{k+1}$$

$$\frac{L_{busy}}{L_{busy} + L_{idle}} = \frac{1}{1 + \frac{L_{idle}}{L_{busy}}} \geq \frac{1}{1 + \frac{2P}{k+1}} = \frac{k+1}{2P+k+1} \ ,$$

meaning that the busy segments constitute at least the $\frac{k+1}{2P+k+1}$-th part of the span $[r_j, d_j]$. We have thus proved the following lemma:

Lemma 4.12. *For a given set of jobs J exclusively of jobs for which $\lambda_j \geq k+1$, algorithm* LSA *yields a feasible k-preemptive schedule $\mathcal{G}_{J_{in}}$ s.t. $\forall j \in J_{out}$, the timespan $[r_j, d_j]$ is at least b_0-loaded, where $b_0 = \frac{k+1}{2P+k+1}$.* ■

Remark. Since in LSA we operate under classify and select, in each class $P(J_c) \leq k+1$, and therefore $b_0 \geq \frac{1}{3}$.

Now let us assume WLOG that all the union of the rejected jobs' timespans $U = \bigcup_{j \in J_{out}} [r_j, d_j]$ is a single contiguous time segment.

Applying to U the construction described in lemmas 4.7, 4.8, we get two disjoint sets of pairwise disjoint timespans U'_E, U'_O, in which in turn correspond to two disjoint sets of jobs J'_E, J'_O. Let us denote the timespans set with the larger total length U^*, and the corresponding job set J^*. By lemma 4.12, the timespan of every job in J_{out}, and in particular of every job j in $J^* \subseteq J_{out}$, is at least b_0-loaded by jobs from J_{in}. Since the timespans of jobs in J^* are disjoint, we can safely claim that

$$\sum_{j \in J_{\text{OPT}}} p_j \leq |U| \leq 2 \cdot |U^*| = 2 \cdot \sum_{j \in J^*} \lambda_j p_j \leq \frac{2}{b_0} \cdot \sum_{j \in J_{in}} p_j \ .$$

Note that since we made no special assumptions, this inequality applies to every contiguous subset of U involving at least two timespans (and by linearity, to every union of disjoint pairs of timespans) and to the corresponding subsets of J_{OPT}, J_{in}. (For subsets of U consisting of a single timespan the stricter lemma 4.12 holds.) Let us assume an order on J by the jobs' densities from the largest to the smallest; since algorithm LSA considers the jobs in the same order, and so the corresponding members of J_{in} are always considered prior to the corresponding members of J_{OPT}, we can claim that this inequality applies to all the prefixes of J_{OPT}, J_{in} by this order. Now we will utilize lemma 4.9 to achieve:

$$\text{OPT}_\infty(J) = \sum_{j \in J_{\text{OPT}}} \sigma_j p_j \leq \frac{2}{b_0} \cdot \sum_{j \in J_{in}} \sigma_j p_j = \frac{2}{b_0} \cdot \text{val}(J_{in})$$

Note that since LSA operates on classes of jobs separately, this inequality holds for every class J_c.

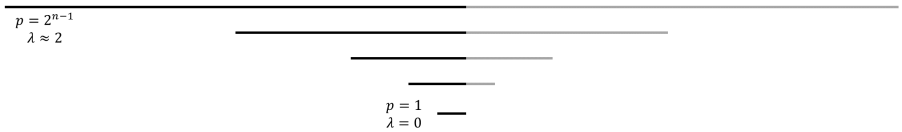

Figure 2: Lower bound for k=0

Proof of Lemma 4.10. Recall that LSA_CS divides the lax jobs into $\log_{k+1} P$ classes J_c, for each of which the above inequality holds with $b_0 \geq \frac{1}{3}$. Therefore, for each class J_c the application of the internal function LSA will yield $J_{c,in}$ s.t. $\forall c, \text{val}(J_{c,in}) \geq \frac{1}{6} \cdot \text{val}(J_{c,\text{OPT}})$. Since necessarily $\exists c^* \in [\log_{k+1} P]$ s.t. $\text{val}(J_{c^*,\text{OPT}}) \geq \frac{1}{\log_{k+1} P} \cdot \text{val}(J_{\text{OPT}})$, it follows that $\text{val}(J_{c^*,in}) \geq \frac{1}{6} \cdot \text{val}(J_{c^*,\text{OPT}}) \geq \frac{1}{6} \cdot \frac{1}{\log_{k+1} P} \cdot \text{val}(J_{\text{OPT}})$.

Thus, classifying all the lax jobs as described, running LSA separately for each of the classes and taking the best result will yield us J_{in} s.t. $\text{val}(J_{in}) \geq \frac{1}{6 \cdot \log_{k+1} P} \cdot \text{val}(J_{\text{OPT}})$. ∎

4.3.3 Overall Price of Preemption. To conclude the proof, we combine the results of lemmas 4.6 and 4.10 into a single algorithm.

Algorithm 3: k-PreemptionCombined

Input: A pair $\langle J, \mathcal{G}_J \rangle$, where J is a set of jobs, and $\mathcal{G}_J$ is a feasible ∞-preemptive schedule for them.

Output: A feasible k-preemptive schedule $\mathcal{G}'_{J'}$ for $J' \subseteq J$, s.t. $\text{val}(J') \geq O(\frac{1}{\log_{k+1} P}) \cdot \text{val}(J_{\text{OPT}})$.

$J_1, \mathcal{G}_{J_1} \leftarrow$ all jobs j and schedules G_j for which $\lambda_j \leq k+1$;

$J_2, \mathcal{G}_{J_2} \leftarrow$ all jobs j and schedules G_j for which $\lambda_j \geq k+1$;

$J'_1 \leftarrow$ LevelledContraction(J_1) ;

$J'_2 \leftarrow$ LSA_CS(J_2) ;

return $\max(J'_1, J'_2)$;

Analysis. Since the union of J_1, J_2 yields back all the jobs J, one of them has at least half the total value of J; similarly, the optimal unbounded preemptive scheduling for one of them is at least half of the optimal one for the whole. Since both algorithms yield the $O(\log_{k+1} P)$-th fraction of the input's optimal unbounded preemption's value, the total value of at least one of J'_1, J'_2 is a $O(\log_{k+1} P)$-th fraction of $\text{val}(J_{\text{OPT}})$ itself. Choosing the better result ensures this factor further.

Remark. Like in subsection 4.1, we can eliminate migration and retain a least a $\frac{1}{6}$ of the optimal job set prior to applying the algorithm LSA_CS. Thus, the price of the migrative setting is still $O(\log_{k+1} P)$.

From the above we can conclude the proof of theorem 4.5.

Proof of Theorem 4.5.

$$PoBP_k \triangleq \sup_J \frac{\text{OPT}_\infty(J)}{\text{OPT}_k(J)} \leq \frac{2 \cdot \max\left(\text{OPT}_\infty(J_1), \text{OPT}_\infty(J_2)\right)}{\max\left(\text{val}(J'_1), \text{val}(J'_2)\right)} = O(\log_{k+1} P)$$

∎

This bound is tight, as we will prove in appendix B:

Theorem 4.13. *The Price of Bounded Preemption for k preemptions is* $\Theta(\log_{k+1} P)$.

4.3.4 Extending to Multiple Machines. The result for strict jobs can be trivially extended to multiple non-migrative machines by applying the reduction from subsection 4.1 and algorithm TM for each machine separately, which yields the same bound on the price.

As for the lax jobs, the LSA_CS algorithm can be extended to multiple non-migrative machines in the following way. In each iteration i, the i-th machine is assigned the schedule given by LSA_CS for the jobs remaining (formally, $J_i = J \setminus \bigcup_{k=1}^{i-1} J'_k$). By a known result from [2], this adds at most 1 to the price, thus preserving the $O(\log_{k+1} P)$ ratio (price).

This overall result for non-migrative machines, being constructive, immediately implies an approximation with the same $O(\log_{k+1} P)$ factor. This approximation can be further extended to the migrative machines setting using the results of [18], which adds a multiplicative factor to the quality of the approximation. This approximation (relative to $\leq J_{\infty,\text{OPT}}$) immediately implies a $O(\log_{k+1} P)$ price for the migrative setting as well (since necessarily $J_{k,\text{OPT}} \leq J_{\infty,\text{OPT}}$ and therefore the price is smaller than the approximation factor).

5 SPECIAL CASE: $k = 0$

Note that the expressions $\log_{k+1} n, \log_{k+1} P$ are defined only for $k \geq 1$. To complete our work, let us now look at the special case where $k = 0$, i.e. we are not allowed to preempt at all whereas the optimal algorithm can preempt as much as is required.

Let us look at a specific example of a series of unit-value jobs as presented in figure 2. Since the jobs' lengths form a geometrical progression with $q = 2$, it is clear that even with one preemption per job allowed, all of the jobs can be accommodated. On the other hand, if no preemptions are allowed, one can only accommodate a single job. Therefore the price of bounding preemption in this case is n and also, due to the nature of the example, $\log P$; these are the lower bounds for the price in case $k = 0$.

These lower bounds can be trivially extended to multiple machines by simply multiplying the setting by the number of machines (along a third axis, so to speak); the bound of $\log P$ remains the same, whereas the bound of n becomes $O(n)$ for any fixed number of machines.

The n lower bound for the price is trivially tight, since we can always choose from a set of jobs the one with the maximal value and thus achieve a feasible schedule; however, the tightness of the $\log P$ approximation ratio should be examined further.

Let us utilize the algorithm LSA_CS, from subsection 4.3, adjusting it for the case $k = 0$, i.e. mandating scheduling to be made solely *en*

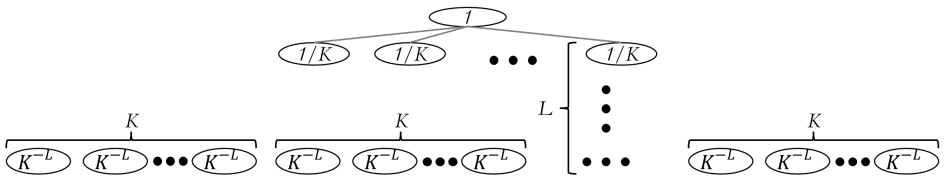

Figure 3: k-BAS lower bound

bloc and classifying the jobs s.t. $\forall J_c, P(J_c) \leq 2$. Lemma 4.11 still holds in this setting.

Let us now look at the set $J_{out} \triangleq J \setminus J'$; specifically, let us look at $j \in J_{out}$ at the moment of its rejection. Again, since the algorithm schedules the jobs in descending order of density, it suffices to compare the total lengths of the busy and idle segments. If j is rejected, this means that the total length of the idle segments in $[r_j, d_j]$ is less than p_j, i.e.$L_I \leq P$. On the other hand, trivially $L_B \geq 1$, since if there were no busy segments in the span, it would be possible to schedule j. Thus

$$\left.\begin{array}{r} L_I \leq P \\ L_B \geq 1 \end{array}\right\} \Rightarrow \frac{L_I}{L_B} \leq P$$

$$b_0 = \frac{L_B}{L_B + L_I} = \frac{1}{1 + \frac{L_I}{L_B}} \geq \frac{1}{1+P} \ .$$

By the same technique as in subsection 4.3 this yields us $\text{val}(J_{in}) \geq \frac{1}{1+P} \cdot \text{val}(J_{\text{OPT}})$ for each class. Since (in this case) we classify the jobs so that $P \leq 2$, by the same reasoning as before, taking the maximal-value $J_{c,in}$ yields us in the overall $\text{val}(J_{in}) \geq \frac{1}{3 \cdot \log P} \cdot \text{val}(J_{\text{OPT}})$. By the same logic as in 4.3.4, this result remains the same for the multiple machines setting.

Thus the upper bound for the price in case $k = 0$ is established to be $O(\log P)$, and in conjunction with the previous, it means that this bound is tight. ■

REFERENCES

[1] S. Albagli-Kim, B. Schieber, H. Shachnai, and T. Tamir. Real-time k-bounded preemptive scheduling. In *Proc. of the 18th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 127–137, 2016.

[2] B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén. On-line competitive algorithms for call admission in optical networks. *Algorithmica*, 31(1):29–43, 2001.

[3] Y. Azar and O. Gilon. Scheduling with deadlines and buffer management with processing requirements. *Algorithmica*, 78(4):1246–1262, 2017.

[4] Y. Azar and O. Regev. Combinatorial algorithms for the unsplittable flow problem. *Algorithmica*, 44(1):49–66, 2006.

[5] P. Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *Journal of Scheduling*, 2:245–252, 1999.

[6] P. Baptiste, M. Chrobak, C. Dürr, W. Jawor, and N. Vakhania. Preemptive scheduling of equal-length jobs to maximize weighted throughput. *Operations Research Letters*, 32(3):258–264, 2004.

[7] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001.

[8] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM J. Comput.*, 31(2):331–352, 2001.

[9] R. Bar-Yehuda and S. Even. A local ratio theorem for approximating the weighted vertex cover problem. *Ann. Discrete Math.*, 25:27–46, 1985.

[10] R. Bar-Yehuda, M. M. Halldórsson, J. S. Naor, H. Shachnai, and I. Shapira. Scheduling split intervals. *SIAM J. Comput.*, 36(1):1–15, 2006.

[11] S. Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *Proc. 17th Euromicro Conf. Real-Time Syst. (ECRTS)*, pages 137–144, 2005.

[12] R. J. Bril, J. J. Lukkien, and W. F. J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. In *Proc. 19th Euromicro Conf. RealTime Syst. (ECRTS)*, pages 269–279, 2007.

[13] G. C. Buttazzo, M. Bertogna, and G. Yao. Limited preemptive scheduling for real-time systems. a survey. *IEEE Transactions on Industrial Informatics*, 9(1):3–15, 2013.

[14] R. Canetti and S. Irani. Bounding the power of preemption in randomized scheduling. *SIAM J. Comput.*, 27(4):993–1015, 1998.

[15] S. Dauzère-Pérès. Minimizing late jobs in the general one machine scheduling problem. *European Journal of Operational Research*, 81(1):134–142, 1995.

[16] L. Epstein, A. Levin, A. J. Soper, and V. A. Strusevich. Power of preemption for minimizing total completion time on uniform parallel machines. *SIAM J. Discrete Math.*, 31(1):101–123, 2017.

[17] R. Graham, E. L. Lawler, J. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Arm. Discr. Math.*, 5:75–90, 1979.

[18] B. Kalyanasundaram and K. Pruhs. Eliminating migration in multi-processor scheduling. *J. Algorithms*, 38(1):2–24, 2001.

[19] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[20] E. L. Lawler. Sequencing to minimize the weighted number of tardy jobs. *Rev. Française d'Automatique, Informatique, Recherche Opérationnel*, 10:27–33, 05 1976.

[21] E. L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26:125–133, 1990.

[22] E. L. Lawler. Knapsack-like scheduling problems, the moore-hodgson algorithm and the 'tower of sets' property. *Mathl. Comput. Modelling*, 20(2):91–106, 1994.

[23] E. L. Lawler and J. M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16:77–84, 1969.

[24] J. M. Moore. Sequencing n jobs on one machine to minimize the number of tardy jobs. *Management Science*, 15:102–109, 1968.

[25] K. Pruhs and G. J. Woeginger. Approximation schemes for a class of subset selection problems. *Theoretical Computer Science*, 382:151–156, 2007.

[26] A. J. Soper and V. A. Strusevich. Power of Preemption on Uniform Parallel Machines. In K. Jansen, J. D. P. Rolim, N. R. Devanur, and C. Moore, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*, volume 28 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 392–402, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[27] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *Proc. 6th IEEE Int. Conf. Real-Time Comput. Syst. Appl. (RTCSA)*, pages 328–335, 1999.

A LOWER BOUND FOR LOSS FACTOR IN k-BAS

In this appendix we establish the lower bound for the loss factor of the k-BAS by providing a specific example, as shown in figure 3. All the claims in this subsection pertain solely to said example.

The nodes of the tree are arranged in $\lfloor \log_K n \rfloor + 1$ *levels*, numbered from 0 (the topmost) to $L \triangleq \lfloor \log_K n \rfloor$ (the lowermost). In each level i, the number of nodes is K^i and the value of each of them K^{-i}. Every node has exactly K children. K is taken to be an arbitrary integer number strictly larger than k (later, we will assign it a specific value); in particular, since $k \geq 1$, this implies $K > 1$.

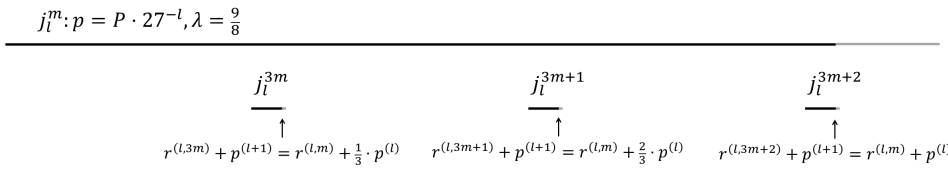

Figure 4: General lower bound: example for $K = 3$

Observation A.1. *The total value of all the nodes in a level of the tree is 1; thus the total value of all the nodes in the tree is $L + 1$.*

Lemma A.2. *For each node v on level i, algorithm* TM *yields*

$$t(v) = K^{-i} \cdot \sum_{j=0}^{L-i} \left(\frac{k}{K}\right)^j, \quad m(v) = K^{-i} \cdot \sum_{j=0}^{L-(i+1)} \left(\frac{k}{K}\right)^j .$$

Note that this implies that for every node v at level i, $t(v) > m(v)$.

Proof. By induction on the application of the algorithm TM (from level L up to 0).
Base: For each leaf v ($i = L$), $t(v) = K^{-L}$, $m(v) = 0$.
Hypothesis: For a node (job) v at level i,

$$t(v) = K^{-i} \cdot \sum_{j=0}^{L-i} \left(\frac{k}{K}\right)^j, \quad m(v) = K^{-i} \cdot \sum_{j=0}^{L-(i+1)} \left(\frac{k}{K}\right)^j .$$

Step: Let u be a job at level $i - 1$. Then:

$$t(u) = \mathrm{val}(u) + \sum_{v_i \in CN_k(u)} t(v_i) = K^{1-i} + k \cdot K^{-i} \cdot \sum_{j=0}^{L-i} \left(\frac{k}{K}\right)^j = K^{1-i} \cdot \sum_{j=0}^{L-i+1} \left(\frac{k}{K}\right)^j$$

$$m(u) = \sum_{v_i \in CN(u)} \max(t(v_i), m(v_i)) = K \cdot K^{-i} \cdot \sum_{j=0}^{L-i} \left(\frac{k}{K}\right)^j = K^{1-i} \cdot \sum_{j=0}^{L-i} \left(\frac{k}{K}\right)^j$$

■

Corollary A.3. *The total value of the k-BAS returned by* TM *on the example is less than $\frac{K}{K-k}$.*

Proof. Denote the root node v_0. The total value of the k-BAS is

$$\mathrm{ALG} = \max\{t(v_0), m(v_0)\} = t(v_0) = \sum_{j=0}^{L} \left(\frac{k}{K}\right)^j < \frac{1}{1 - \frac{k}{K}} = \frac{K}{K-k} .$$

■

Proof of Theorem 3.20. We have established in subsection 3.2 that the TM algorithm yields the optimal result. If we take the example and set $K = 2k$, we get:

$$\left.\begin{array}{l} \mathrm{OPT}_\infty = \lfloor \log_{k+1} n \rfloor + 1 \\ \mathrm{OPT}_k < \frac{2k}{2k-k} = 2 \end{array}\right\} \Rightarrow \frac{\mathrm{OPT}_\infty}{\mathrm{OPT}_k} = \Omega(\log_{k+1} n)$$

■

B LOWER BOUND FOR PRICE IN k-BOUNDED PREEMPTION PROBLEM

In this appendix we establish the lower bound for the price of k-bounded preemption by providing a specific example, as shown in figure 4. All the claims in this section pertain solely to said example.

The example is constructed as follows:

- All the jobs are arranged in $L + 1$ *levels* numbered from 0 to L, where $L \triangleq \lfloor \log_{3K^2} P \rfloor$. Level l contains K^l jobs, numbered from 0 to $K^l - 1$. The m-th job at the l-th level is denoted j_l^m.
- The value of a job j_l^m is K^{-l}.
- The length of a job j_l^m is $p^{(l)} = P \cdot (3K^2)^{-l}$.
- The relative laxity of all the jobs is $\lambda = 1 + \frac{1}{3K-1}$.
- For a job j_l^m, the jobs $\{j_{l+1}^{m'} \mid mK \le m' \le (m+1)K - 1\}$ are called the *child jobs* of j_l^m. This term can be trivially extended to introduce the notion of a *descendant job*.
- The release times are given by the recursive formula $r^{(l+1,m')} = r^{(l,m)} + (m' - mK + 1) \cdot \frac{p^{(l)}}{K} - p^{(l+1)}$, where $mK \le m' \le (m+1)K - 1$ and $r^{(0,0)} = 0$.
- The deadline times are given by $d^{(l,m)} = r^{(l,m)} + p^{(l)} \cdot \lambda$.

From the above construction one can deduce the following (proofs are omitted and will be presented in the full version of the paper):

Lemma B.1. *For every job j_l^m, a single preemption allows the scheduling of at most one of j_l^m's child jobs. Therefore, in a feasible solution, a job is scheduled with at most k child jobs.*

Lemma B.2.

$$\begin{cases} \mathrm{OPT}_\infty = L + 1 > \log_{3K^2} P = \frac{\log_K P}{\log_K (3K^2)} > \frac{1}{3} \cdot \log_K P \\ \mathrm{OPT}_k = \sum_{i=0}^{L} \left(\frac{k}{K}\right)^i = \frac{1 - \left(\frac{k}{K}\right)^{L+1}}{1 - \frac{k}{K}} < \frac{1}{1 - \frac{k}{K}} = \frac{K}{K-k} \end{cases}$$

Proof of Theorem 4.13. Let us choose $K = 2k$; then:

$$\left.\begin{array}{l} \mathrm{OPT}_\infty > \frac{1}{3} \cdot \log_{2k} P \\ \mathrm{OPT}_k < \frac{2k}{2k-k} = 2 \end{array}\right\} \Rightarrow PoBP_k > \Omega(\log_{k+1} P)$$

■

Proof of Theorem 4.3. We use the same example. Then

$$n = \sum_{l=0}^{L} K^l = \frac{K^{L+1} - 1}{K - 1} < \frac{K \cdot K^L}{K - 1} \le \frac{3}{2} \cdot K^L \;\Rightarrow\; \log_K n < L ,$$

and if we, as in the previous paragraph, choose $K = 2k$, we get

$$\left.\begin{array}{l} \mathrm{OPT}_\infty > \log_{2k} n \\ \mathrm{OPT}_k < \frac{2k}{2k-k} = 2 \end{array}\right\} \Rightarrow PoBP_k > \Omega(\log_{k+1} n)$$

■

Remark. For the multiple machine setting, this example can be multiplied "along a third axis", effectively forcing to solve the same basic example for each machine separately. The $\Omega(\log_{k+1} P)$ bound will hold as is; the $\Omega(\log_{k+1} n)$ will hold for any fixed number of machines.