Capacitated max-Batching with Interval Graph Compatibilities

Tim Nonner *
Albert Ludwigs University of Freiburg, Germany
nonner@informatik.uni-freiburg.de

January 5, 2010

Abstract

We consider the problem of partitioning interval graphs into cliques of bounded size. Each interval has a weight, and the cost of a clique is the maximum weight of any interval in the clique. This natural graph problem can be interpreted as a batch scheduling problem. Solving an open question from [7, 4, 5], we show NP-hardness, even if the bound on the clique sizes is constant. Moreover, we give a PTAS based on a novel dynamic programming technique for this case.

1 Introduction

We consider the problem of partitioning interval graphs into cliques of bounded size. Each vertex or rather interval has a weight, and the cost of a clique is the maximum weight of any interval in the clique. Specifically, let J be the intervals of the graph, where each interval $I \in J$ has a weight $w_I \in \mathbb{R}^+$, and let k be the bound on the clique size. We extend the weights to all cliques C of at most size k with $w_C := \max_{I \in C} w_I$. The objective is hence to partition the intervals J into cliques $C_1, C_2, \ldots, C_s$ of at most size k such that $\sum_{i=1}^{s} w_{C_i}$ is minimized. We refer to this problem as CB_k. Note that we can think of an interval as a *job*, and of a clique as a *batch* of jobs which satisfy the compatibility constraint implied by

*Supported by DFG research program No 1103 *Embedded Microsystems*. Parts of this work were done while the author was visiting the IBM T.J. Watson Research Center.

the interval graph structure. In this context, this problem can be interpreted as a capacitated batch scheduling problem, where the maximum weight of a job in a batch gives the time needed to process this batch [7, 4], and hence the objective function above is the total completion time. CB_k can be generalized to arbitrary graphs instead of interval graphs, as done in [8, 7, 5]. In this case, the problem is clearly NP-hard, since it contains graph coloring [10].

Previous work. Finke et al. [7] showed that CB_k can be solved via dynamic programming in polynomial time for $k = \infty$. A similar result was independently obtained by Becchetti et al. [2] in the context of data aggregation. Moreover, this result was extended by Gijswijt, Jost, and Queyranne [8] to value-polymatroidal cost functions, a subset of the well-known submodular cost functions. Using this result as a relaxation, Correa et al. [5] recently presented a 2-approximation algorithm for CB_k for arbitrary k. However, it was raised as an open problem in [7, 4, 5] whether this problem is NP-hard or not.

Note that if the weights on the intervals are uniform, for example $w_I = 1$ for each interval $I \in J$, then CB_k simplifies to finding a clique partition of minimum cardinality where each clique has at most size k [3]. We can also think of this problem as a hitting set problem with uniform capacity k, where we want to hit or *stab* intervals with vertical lines which correspond to the cliques. Since the natural greedy algorithm solves this problem in polynomial time [7], Even et al. [6] addressed the more complicated case of non-uniform capacities. They presented a polynomial time algorithm based on a general dynamic programming approach introduced by Baptiste [1] for the problem of scheduling jobs such that the number of gaps is minimized.

Contributions. We settle the complexity of CB_k by proving its NP-hardness in Section 5, even for $k = 3$, which solves an open problem from [7, 4, 5]. This is tight, since CB_k can be solved in polynomial time for $k = 2$ by using an algorithm for weighted matching. Moreover, we present a dynamic programming based PTAS for any constant k in Section 4. It is worth mentioning that this dynamic program differs significantly from the dynamic programs introduced before for the related problems discussed above. Using the natural geometric interpretation of CB_k used in [7], we briefly discuss these approaches in Subsection 2.1. As an initial building block for the PTAS, we first show in Section 3 that we only need to consider instances with a constant number of different weights. This result holds for general graphs as well, and therefore, it is explained in a separate section.

Related work. Also the complementary problem, where we want to partition a graph into independent sets instead of cliques, has raised a considerable amount of attention [12, 11]. Note that, for $k = \infty$, finding such a partition is equivalent to coloring a graph. Pemmaraju, Raman, and Varadarajan [12] showed that this complementary problem is NP-hard, even for interval graphs and $k = \infty$. Moreover, Pemmaraju, and Raman [11] showed that, for $k = \infty$, a graph class admits a $4c$-approximation algorithm if there is a c-approximation algorithm for the simpler coloring problem. Finally, note that our results for CB_k can be applied to this complementary problem for the graph class of co-interval graphs and constant k.

2 Preliminaries

Let n be the number of intervals in J, and assume that all endpoints of intervals in J are elements in the range $\{1, \ldots, T\}$ for $T := 2n$. This discretisation is not necessary, but simplifies the descriptions of the arguments to follow. We refer to an element in $\{1, \ldots, T\}$ as a *period.* We also refer to a clique-partition $C_1, C_2, \ldots, C_s$ as a *schedule*, and we always denote a schedule with σ. Therefore, define $\mathrm{cost}(\sigma) := \sum_{C \in \sigma} w_C$, and let $\mathrm{OPT} := \mathrm{cost}(\sigma^*)$ denote the cost of an optimal schedule σ^*.

2.1 Geometric interpretation and dynamic programming

Consider some schedule σ for J. For each batch $C \in \sigma$, there is some period t_C such that $t_C \in I$ for each interval $I \in C$. If t_C is non-unique, simply choose t_C to be the smallest such period. Consequently, if we interpret the weight w_I of each interval I as a vertical height, then we can also think of a batch as a vertical line with x-coordinate t_C starting at y-coordinate w_C and ending at y-coordinate 0. We say that each interval contained in C is *stabbed* by this batch. However, since each such batch may stab at most k intervals, we say that it has *capacity* k, and we have to indicate which intervals are stabbed.

Example. We illustrate the geometric interpretation of an example instance J in Figure 1. This instance consists of five intervals, where the vertical height of each interval $I \in J$ is its weight w_I. For $k = 3$, the vertical dashed lines represent of schedule σ containing two batches C_1 and C_2 with $t_{C_1} = 3$, $w_{C_1} = 4$, $t_{C_2} = 7$, and $w_{C_2} = 5$. The fat dots indicate which intervals are stabbed by which batches. We have that $\mathrm{cost}(\sigma) = 9$, and this is clearly optimal for this instance. Note that batch C_2 does not fully exploit its capacity k.

The geometic nature of CB_k makes it a natural target for dynamic programming techniques. Indeed, many algorithms for related problems are dynamic program-

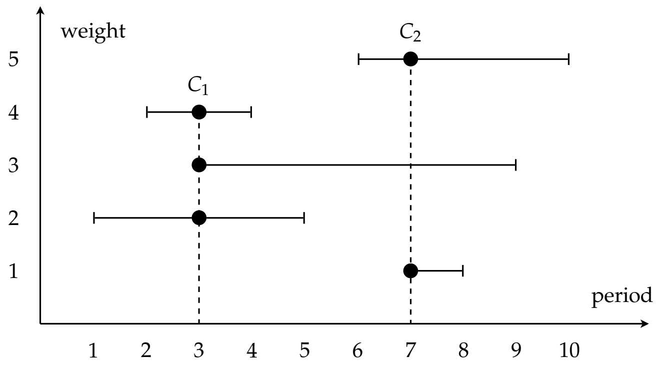


Figure 1: Geometric interpretation of an instance J.

ming based [6, 2, 5, 7].

First, we discuss the approach from [7, 2] for $k = \infty$. Specifically, for any pair of integers $0 \leq s < t \leq T+1$, let $J(s,t)$ denote the subinstance of J that contains all intervals $I \in J$ with $I \subseteq (s,t)$, where (s,t) is the open real interval with endpoints s and t. Observe that $J(0, T+1) = J$. Consider now an interval $I \in J(s,t)$ with maximal weight w_I. Clearly, for any schedule σ, there must be a batch $C \in \sigma$ with $t_C \in I$ and $w_C = w_I$ that stabs interval I. Moreover, such a batch *decomposes* $J(s,t)$ into two new subinstances, namely $J(s, t_C)$ and $J(t_C, t)$, such that each interval $I \in J(s,t)$ is either stabed by batch C, or contained in one of these two new subinstances. Recall here that w_I is maximal and $k = \infty$, and thus, batch C may stab all intervals $I' \in J(s,t)$ with $t_C \in I'$ as well. Since there are at most $T = 2n$ many possible choices for t_C, this straightforward establishes a recurrence relation for a dynamic program, where each subinstance $J(s,t)$ corresponds to an entry in the dynamic programming array.

However, if $k < \infty$, then it is not clear which intervals except I batch C should stab as well, and if an interval $I' \in J(s,t)$ with $t_C \in I'$ is not stabbed, then we need to decide whether to *assign* it to the left or right, i.e., whether to add this interval to the left subinstance $J(s, t_C)$ or to the right subinstance $J(t_C, t)$. We will solve this *left-right assignment dilemma* approximately with Lemma 7, but we need many other ingredients. However, we will use a different type of subinstance and a slightly different way to decompose subinstances. More specifically, we will select a consecutive pair of periods $a < b = a+1$ to decompose such a subinstance J' *between* these periods, i.e., we can think of this as selecting $a < t_C < b$. But analogously, all intervals $I \in J'$ with $I \leq a$ will be assigned to the left, all intervals $I \in J'$ will be assigned to the right, and the intervals $I \in J'$ where the left-right assignment dilemma occurs are the ones with $\{a, b\} \subset I$.

Correa et al. [5] used the dynamic program from [7, 2] explained above to obtain a 2-approximation algorithm for CB_k for arbitrary k. Specifically, they first compute an optimal schedule σ for $k = \infty$ in polynomial time using this approach, and then they decompose each batch $C \in \sigma$ into batches of at most size k.

Even et al. [6] address non-uniform capacities, i.e., each period t has capacity k_t such that each batch C with $t_C = t$ may stab at most k_t intervals, but the interval weights are uniform, for example $w_I = 1$ for each interval $I \in J$. They discuss the hard- and soft-capacitated case. In the *hard-capacitated case*, there must be at most one batch C with $t_C = t$ for each period t, and such a batch adds some non-uniform period-specifice cost w_t to the objective function. Therefore, it is not clear whether there is a feasible solution. On the other hand hand, in the *soft-capacitated case*, there is no limit on the number of such batches C with $t_C = t$, but each such batch C adds w_t to the objective function. Therefore, the soft-capacitated case is more related to CB_k. It is worth mentioning here that our PTAS can be extended to non-uniform capacities k_t as long as all capacities are uniformely bounded by some constant k.

The dynamic program in [6] is based on a technique introduced in [1] which avoids the left-right assignment dilemma by chosing different subinstances. Specifically, they order the intervals in J according to their right endpoints, say $I_1, I_2, \dots, I_n$. Each considered subinstance $J(s,t,i)$ is then defined by two integers $0 \le s < t \le T+1$ and an additional integer $1 \le i \le n$ such that $J(s,t,i) := \{[a,b] \in \{I_1, I_2, \dots, I_i\} \mid s < a < t\}$. Note that these subinstances significantly differ from the subinstances of the form $J(s,t)$ discussed above.

However, all dynamic programs briefly discussed in this subsection fail for CB_k, since the approach in [7, 2] is not able to deal with capacities, and the approach in [6] fails to work for different weights. Therefore, we need a significantly different strategy.

2.2 Interpretation of a schedule as a function

We can also think of a schedule σ as a function $\sigma : J \to \{1, \dots, T\}$ that assigns each interval $I \in J$ to the period $\sigma(I) = t_C$, where $C \in \sigma$ is the batch with $I \in C$. For each period t, let then $\sigma_t := \{I \in J \mid \sigma(I) = t\}$ be the batch of all intervals which are assigned to period t. Given σ_t, it is easy to compute the optimal cost of schedule σ at period t. To this end, we also define a weight w_C for each batch C with $|C| > k$ as follows: Let $C_1, C_2, \dots, C_s$ be a partition of C into subbatches of at most size k such that (1) $C_s \le k$, (2) for each $1 \le i \le s-1$, $|C_i| = k$, and (3), for each pair $1 \le i < j \le s$ and intervals $I \in C_i$ and $I' \in C_j$, $w_I \le w_{I'}$. Define then $w_C := \sum_{i=1}^{s} w_{C_i}$. Observe that this 'top-down' approach is the optimal way to partition the batch C into batches of at most size k, and hence, the optimal cost of σ at period t is w_{σ_t}. Therefore, CB_k is the problem of finding a schedule σ that minimizes $\sum_{t=1}^{T} w_{\sigma_t}$. The same partition was used by Correa et al [5] to obtain a 2-approximation algorithm.

We will alternately use the interpretation of a schedule σ as a function, as introduced in this subsection, or a set of vertical lines, as introduced in Subsection 2.1. Specifically, in Sections 3 and 5, we will use the geometric interpretation as a set of lines, and in Section 4, we will use the interpretation as a function assigning intervals to periods.

3 Reducing the number of different weights

First, we need the following simple lemma, which establishes a simple geometric rounding step.

Lemma 1. *For any $\alpha > 1$, by losing an α-factor in the approximation ratio, we may assume that all interval weights are powers of α.*

Proof. Assume that $\min_{I \in J} w_I = 1$, and then simply round the weight w_I of each interval $I \in J$ up to the next power of α. It is easy to see that any schedule for the old instance yields a schedule for this new rounded instance whose cost is at most the factor α larger, and any schedule for the new instance yields a schedule for the old instance whose cost is even smaller. This proves the claim. ☐

By Lemma 1, we may assume that there is a constant $\alpha > 1$ such that, for each interval $I \in J$, $w_I = \alpha^{i_I - 1}$ for some integer $i_I \geq 1$. Hence, for each batch C, there is also some positive integer i_C such that $w_C = \alpha^{i_C - 1}$. Let $m := \max_{I \in J} i_I$ be the maximal necessary such integer. We also refer to an integer $1 \leq i \leq m$ as a *level* in what follows, and we can think of using levels instead of weights as introducing a logarithmic scale with base α. The following lemma provides a bound on the number of levels m.

Lemma 2. *For any $\epsilon > 0$, by losing an $(1+\epsilon)$-factor in the approximation ratio, we may assume that m is constant.*

Proof. We need the following algorithm, named $\mathcal{A}'$, which is an adaption of the well-known shifting technique of Hochbaum and Maas [9]. An additional input except instance J is an algorithm $\mathcal{A}$ for instances where m is constant. Consider some constant but arbitrarily small $\epsilon > 0$, and assume that $\epsilon m \geq 1$. Otherwise, if $\epsilon m < 1$, then we already have that m is constant. Assume then moreover that $1/\epsilon$ and ϵm are integral. This can be ensured by sufficiently decreasing ϵ. The parameter ϵ will affect the approximation ratio of algorithm $\mathcal{A}'$.

$\mathcal{A}'(J, \epsilon, \mathcal{A})$

---

1. Select an integer $1 \leq z \leq 1/\epsilon$ uniformly at random.
2. For $j = 0, \ldots, \epsilon m$:
   (a) Set $J_j \leftarrow \{I \in J \mid (j-1)/\epsilon + z < i_I \leq j/\epsilon + z\}$.
   (b) Compute a schedule σ_j for J_j with algorithm $\mathcal{A}$.
3. Return the schedule $\sigma \leftarrow \cup_{j=0}^{\epsilon m} \sigma_j$ for J.

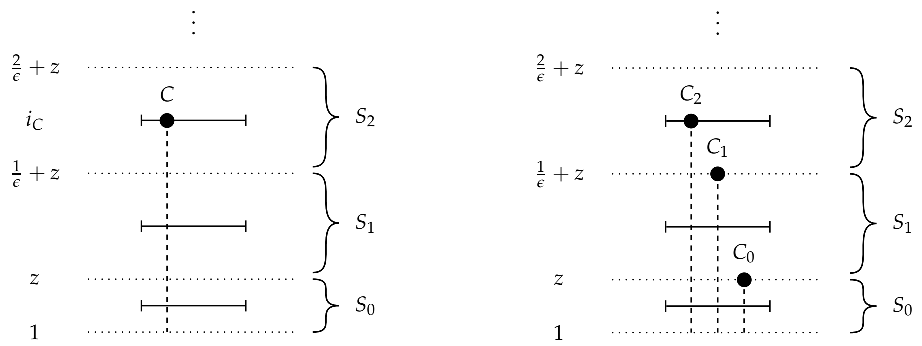
Note that the instances $J_0, J_1, \ldots, J_{\epsilon m}$ are a pairwise disjoint partition of J, and hence, the returned schedule σ is indeed a schedule for J, since each interval in J is stabbed by a batch in one of the schedules $\sigma_0, \sigma_1, \ldots, \sigma_{\epsilon m}$. To finally prove the correctness of algorithm $\mathcal{A}'$, we still have to argue that each of the instances $J_0, J_1, \ldots, J_{\epsilon m}$ has a constant number of levels, since this is required to apply algorithm $\mathcal{A}$. To this end, recall that, for each integer $0 \leq j \leq \epsilon m$ and interval $I \in J_j$, $(j-1)/\epsilon + z < i_I \leq j/\epsilon + z$. Therefore, we can normalize the levels of the intervals in J_j such that $\min_{I \in J_j} i_I = 1$ and $\max_{I \in J_j} i_I \leq 1/\epsilon$. Since $1/\epsilon$ is a constant, this shows that we may assume that the number of levels in J_j is constant.

Consider an optimal schedule σ^*. We create a sequence of schedules $\sigma_0^*, \sigma_1^*, \ldots, \sigma_{\epsilon m}^*$ for the respective instances $J_0, J_1, \ldots, J_{\epsilon m}$ as follows: For each batch $C \in \sigma^*$ and each positive integer j with $(j-1)/\epsilon + z < i_C$, add a batch C_j to σ_j^* with $t_{C_j} := t_C$ and

$$i_{C_j} := \begin{cases} i_C & \text{if } i_C \leq j/\epsilon + z, \\ j/\epsilon + z & \text{otherwise.} \end{cases}$$

For each $0 \leq j \leq \epsilon m$, it is easy to see that σ_j^* is a schedule for J_j if we use each batch $C_j \in \sigma_j^*$ to stab all intervals $I \in J_j$ which are stabbed by the *original* batch $C \in \sigma^*$, i.e., the batch C used to construct C_j. This ensures that the capacity k of batch C_j is not exceeded. However, it might happen that C_j uses far less of its capacity than C. Intuitively, this happens if the intervals in J stabbed by C are spread over many of the instances $J_1, J_2, \ldots, J_{\epsilon m}$.

Example. Consider the example instance J depicted in Subfigure 2(a). The vertical dotted lines represent the levels 1, z, $1/\epsilon + z$, and $2/\epsilon + z$. The three



(a) The schedule σ^*.

(b) The schedules $\sigma_0^*, \sigma_1^*, \sigma_2^*$.

Figure 2: Decomposing an instance J and schedule σ^*.

intervals depicted in Subfigure 2(a) are hence contained in instances J_0, J_1, and J_2 as illustrated on the right of this subfigure. For $k = 3$, consider now the optimal schedule σ^* with the single batch C. We create three schedules $\sigma_0^*, \sigma_1^*, \sigma_3^*$ from σ^*, where these schedules contain the batches C_0, C_1, C_2, respectively, which are depicted in Subfigure 2(b). To disinguish these batches, C_0 and C_1 are moved a little to the right. Each of these batches uses only a third of its total capacity k.

We have that

$$\begin{aligned}
\mathbb{E}\left[\sum_{j=0}^{\epsilon m} \mathrm{cost}(\sigma_j^*)\right] &= \sum_{C\in\sigma^*}\left(\alpha^{i_C-1} + \mathbb{E}\left[\sum_j \alpha^{i_{C_j}-1}\right]\right) \qquad (1)\\
&= \sum_{C\in\sigma^*}\left(\alpha^{i_C-1} + \sum_{x=1}^{1/\epsilon}\left(\Pr[z=x] \sum_{j\geq 0: j/\epsilon+x<i_C} \alpha^{j/\epsilon+x-1}\right)\right)\\
&= \sum_{C\in\sigma^*}\left(\alpha^{i_C-1} + \epsilon\sum_{i=0}^{i_C-2}\alpha^i\right)\\
&< \sum_{C\in\sigma^*}\left(1+\frac{\epsilon}{\alpha-1}\right)\alpha^{i_C-1} = \left(1+\frac{\epsilon}{\alpha-1}\right)\mathrm{OPT}.
\end{aligned}$$

The first line is due to linearity of expectation, and the second line is due to the definition of the batches C_j for each batch $C \in \sigma^*$. The third line is due to the fact that the integer $1 \leq z \leq 1/\epsilon$ is selected uniformly at random, and hence $\Pr[z = x] = \epsilon$ for each fixed integer $1 \leq x \leq 1/\epsilon$. Moreover, note here that, for each level $1 \leq i < i_C - 1$, there is exactly one combination of integers $1 \leq x \leq 1/\epsilon$ and $j \geq 0$ with $j/\epsilon + x = i$. Finally, the fourth line is due to the standard transformation of the geometric series.

Now assume that $\mathcal{A}$ is a β-approximation algorithm. We obtain that

$$\begin{aligned}\mathbb{E}\left[\text{cost}(\sigma)\right] &= \sum_{j=0}^{\epsilon m} \mathbb{E}\left[\text{cost}(\sigma_j)\right] \leq \beta \sum_{j=0}^{\epsilon m} \mathbb{E}\left[\text{cost}(\sigma_j^*)\right] \\ &= \beta \mathbb{E}\left[\sum_{j=0}^{\epsilon m} \text{cost}(\sigma_j^*)\right] < \left(1 + \frac{\epsilon}{\alpha - 1}\right) \beta \text{OPT}.\end{aligned}$$

The first line is due to linearity of expectation and the fact that $\text{cost}(\sigma_j) \leq \beta\text{cost}(\sigma_j^*)$ for each $0 \leq j \leq \epsilon m$. Moreover, the second line is due to linearity of expectation and Inequality (1). Since $\alpha > 1$ is constant and we may choose ϵ arbitrarily small, this proves the claim of the lemma. Specifically, if $\mathcal{A}$ is an approximation scheme, i.e., if we may choose $\beta > 1$ arbitrarily small, then so is $\mathcal{A}'$.

Finally, note that algorithm $\mathcal{A}'$ can be straightforward derandomized by sampling all integers z with $1 \leq z \leq 1/\epsilon$, since there are only constant many. □

4 A PTAS

The goal of this section is to prove the following theorem.

Theorem 3. *There is a PTAS for* CB_k *for any constant* k.

To prove Theorem 3, we first need to introduce the notion of an easy instance in Subsection 4.1. Specifically, such that instance deals with the left-right assignment dilemma by providing some additional information. Moreover, we introduce the notion of a normal schedule in Subsection 4.2, which allows us to restrict the search space. We show then in Subsection 4.3 that there is a dynamic programming based quasipolynomial time algorithm for CB_k with easy instances, and we extend this dynamic program to a QPTAS for general instances in Subsection 4.4. On the other hand, we extend this dynamic program to a PTAS for easy instances in Subsection 4.5. Combining both extensions allows us to finally prove Theorem 3 in Subsection 4.5.

By Lemma 2, assume throughout this section that the number of levels m is constant. For each level i, let $J_i := \{I \in J \mid i_I = i\}$ be all intervals in J with level i. Moreover, assume that all endpoints of intervals in J are distinct, which is possible since we have $T = 2n$. Finally, assume that n and hence T is a power of 2, and that all intervals in J contain at least 2 periods. To avoid rounding steps, for every $\epsilon > 0$ considered in what follows, assume that $1/\epsilon$ is integral.

Recall the interpretation of a schedule as a function assigning intervals to periods introduced in Subsection 2.2. We also allow that a schedule σ is *partial*, i.e., that σ assigns only a subset of intervals $J' \subseteq J$. Hence, in this case, σ assigns an interval $I \in J$ if and only if $\sigma(I)$ is defined. If we do not specify J', then σ assigns all intervals J. Moreover, for any $\epsilon > 0$, we say that a schedule σ assigns all intervals in a subset $J' \subseteq J$ *except an ϵ-fraction* if σ assigns all intervals in a subsets $J'' \subseteq J'$ with $|J''| \geq (1-\epsilon)|J'|$.

4.1 Easy instances

Let R be the rooted binary tree with T leaves and ordered children such that each non-leaf vertex $u \in R$ has a left and right child, denoted $\overleftarrow{u}$ and $\overrightarrow{u}$, respectively. Moreover, let $\hat{u}$ denote the parent of each non-root vertex $u \in R$. For a vertex pair $v, u \in R$, we write $v \overset{\leftarrow}{\succ} u$ and $v \overset{\rightarrow}{\succ} u$ if and only if $\overleftarrow{v}$ and $\overrightarrow{v}$ is an ancestor of u, respectively. We use here that u is an ancestor of itself, and hence $v \overset{\leftarrow}{\succ} \overleftarrow{v}$ and $v \overset{\rightarrow}{\succ} \overrightarrow{v}$. Additionally, we write $v \succ u$ if and only if $v \overset{\leftarrow}{\succ} u$ or $v \overset{\rightarrow}{\succ} u$. Thus, $v \succ u$ implies that $v \neq u$. Finally, let d_u denote the depth of a vertex u in C, where the root has depth 0.

Observe that each vertex u naturally corresponds to a subrange periods $P_u \subseteq \{1, \ldots, T\}$ with $|P_u| = T/2^{d_u}$ as follows: Let $u_1, u_2, \ldots, u_T$ be a ordering of the leaves of C such that, for each $1 \leq i \leq T-1$, $v \overset{\leftarrow}{\succ} u_i$ and $v \overset{\rightarrow}{\succ} u_{i+1}$, where v is the lowest common ancestor of u_i and u_{i+1}. For each $1 \leq i \leq T-1$, define then $P_{u_i} := \{i\}$. Using this, for each non-leaf vertex u, inductively define $P_u := P_{\overleftarrow{u}} \cup P_{\overrightarrow{u}}$, which completes this definition. Note that $P_u = \{1, \ldots, T\}$ for the root u.

For each non-leaf vertex u, define $a_u := \max P_{\overleftarrow{u}}$ and $b_u := \min P_{\overrightarrow{u}}$, and, for each level i, let then R_u^i be all intervals $I \in J_i$ for which u is the non-leaf vertex of maximal depth d_u such that $\{a_u, b_u\} \subseteq I \subseteq [\min P_u, \max P_u]$. Additionally, for each leaf u and level i, define $R_u^i := \emptyset$. Since we assume that each interval in J contains at least 2 periods, we have that the interval sets R_u^i are a pairwise disjoint partition of J.

Example. Consider the four intervals I_1, I_2, I_3, and I_4 in Figure 3 with $T = 8$. Assume that all these intervals have the same level i, and hence, the vertical separation is only used to distinguish these intervals. We obtain a tree R of depth $\log T = 3$. Let u be the root of R. We then have that $P_u = \{1, \ldots, 8\}$, $P_{\overleftarrow{u}} = \{1, \ldots, 4\}$, and $P_{\overrightarrow{u}} = \{5, \ldots, 8\}$, and hence $R_u^i = \{I_2, I_3\}$, $R_{\overleftarrow{u}}^i = \{I_1\}$, and $R_{\overrightarrow{u}}^i = \{I_4\}$.

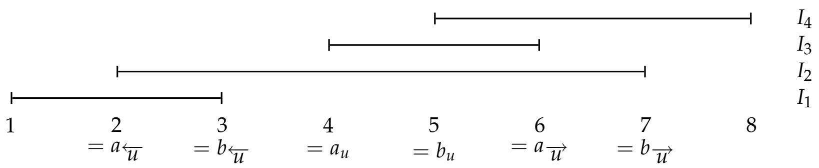


Figure 3: Some sample intervals.

Let A always denote a tuple of interval sets with $A_u^i \subseteq R_u^i$ for each vertex u and level i, and if we say that such a tuple is *u-based*, then A_v^i is only defined for all vertices $v \succ u$. Since m is constant, note that the dimension of a tuple A is $m(2T-1) = O(n)$, but if A is u-based, then its dimension is at most $m \log T = \mathcal{O}(\log n)$. Moreover, we say that a schedule σ *satisfies* a tuple A if, for each vertex u and level i, it holds for each interval $I \in R_u^i$ assigned by σ that

$$\sigma(I) \begin{cases} \leq a_u & \text{if } I \in A_u^i, \\ \geq b_u & \text{otherwise.} \end{cases}$$

Note that it is not necessary for this definition that σ assigns all intervals in J, i.e., σ might be a partial schedule, and that it is possible that σ assigns some intervals not contained in J.

Now we are ready to define the central notion of an easy instance. We say that J is *easy* if we additionally know a tuple A such that there is an optimal schedule σ^* which satisfies A. Clearly, in general, we cannot assume that we know such a tuple A. However, knowing such a tuple allows us to solve the left-right assignment dilemma at each node u such that if we decompose J between periods a_u and b_u, then, for each level i, all intervals in A_u^i can be assigned to the left, and all intervals in $R_u^i \backslash A_u^i$ can be assigned to the right. We will exploit this in Subsection 4.3.

4.2 Normal schedules

Consider a fixed tuple A, vertex pair $v \overleftarrow{\succ} u$, and level i, and let $I_1, I_2, \ldots, I_t$ be an ordering of the intervals A_v^i according to their left endpoints such that, for each $1 \leq j \leq t-1$, the left endpoint of I_j is strictly smaller than the left endpoint of I_{j+1}. Recall that we assume that all endpoints are distinct. Let then

$$A_{vu}^i := \{I_r, I_{r+1}, \ldots, I_s\} \tag{2}$$

be the the consecutive subsequence of intervals in A^i_v whose left endpoints are in P_u, and define

$$A^i_{vu}(j) := \begin{cases} \emptyset & \text{for } 0 \leq j < r, \\ \{I_r, I_{r+1}, \ldots, I_j\} & \text{for } r \leq j \leq s, \\ A^i_{vu} & \text{for } s < j \leq n. \end{cases}$$

Note that we can analogously define A^i_{vu} and $A^i_{vu}(j)$ for $v \overrightarrow{\succ} u$, but in this case, we use $R^i_v \backslash A^i_v$ instead of A^i_v and switch sides, i.e., we use an ordering $I_1, I_2, \ldots, I_t$ of the intervals $R^i_v \backslash A^i_v$ according to their right endpoints such that, for each $1 \leq j \leq t-1$, the right endpoint of I_j is strictly larger than the right endpoint of I_{j+1}. Define then $A^i_{vu} := \{I_r, I_{r+1}, \ldots, I_s\}$ to be the consecutive subsequence of intervals in $R^i_v \backslash A^i_v$ whose right endpoints are in P_u, and finally $A^i_{vu}(j)$ as above. Note that the definition of $A^i_{vu}(j)$ hence includes both cases, $v \overleftarrow{\succ} u$ and $v \overrightarrow{\succ} u$.

Let a always denote an integer tuple with $0 \leq a^i_{vu} \leq n$ for each vertex pair $v \succ u$ and level i, where $a^i_{vu} = n$ if $v = \hat{u}$, and if we say that a is *u-based*, then a^i_{vu} is only defined for all vertices $v \succ u$. Observe that the fact $a^i_{vu} = n$ if $v = \hat{u}$ implies that

$$A^{ai}_{vu} = \begin{cases} A^i_v & \text{if } u = \overleftarrow{v}, \\ R^i_v \backslash A^i_v & \text{if } u = \overrightarrow{v}. \end{cases} \tag{3}$$

Given such an integer tuple a, we abbreviate

$$A^{ai}_{vu} = A^i_{vu}(a^i_{vu}).$$

Consider now a schedule σ, and note that there is obviously some tuple A that is satisfied by σ. Specifically, for each vertex v and level i, simply define $A^i_v := \{I \in R^i_v \mid \sigma(I) \leq a_v\}$. We say that σ is *normal* if there is an integer tuple a such that, for each vertex pair $v \succ u$ and level i, σ assigns an interval $I \in A^i_{vu}$ to a period in P_u if and only if $I \in A^{ai}_{vu}$. We then also say that σ *satisfies* a. The following lemma restricts such a tuple a.

Lemma 4. *Let σ be a normal schedule that satisfies a tuple a. Then, for each vertex pair $v \succ u$ where u is not a leaf and level i, we may assume that*

$$A^{ai}_{v\overleftarrow{u}} \quad \begin{cases} \subseteq A^{ai}_{vu} \cap A^i_{v\overleftarrow{u}} & \text{if } v \overleftarrow{\succ} u, \\ = A^{ai}_{vu} \cap A^i_{v\overleftarrow{u}} & \text{if } v \overrightarrow{\succ} u, \end{cases}$$

$$A^{ai}_{v\overrightarrow{u}} \quad \begin{cases} \subseteq A^{ai}_{vu} \cap A^i_{v\overrightarrow{u}} & \text{if } v \overrightarrow{\succ} u, \\ = A^{ai}_{vu} \cap A^i_{v\overrightarrow{u}} & \text{if } v \overleftarrow{\succ} u, \end{cases}$$

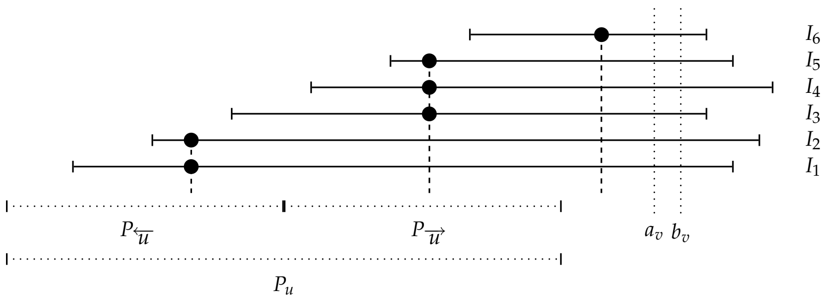


Figure 4: A normal schedule.

and it is possible that the inclusions are not tight.

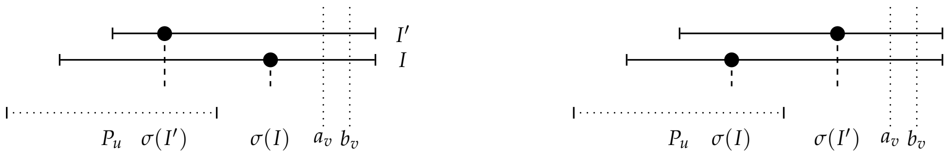
Proof. We only consider the first part with $A^{ai}_{v\overleftarrow{u}}$, since the second part can be proven analogously. Moreover, in this part, if $v \overleftarrow{\succ} u$, then the inclusion $A^{ai}_{v\overleftarrow{u}} \subseteq A^{ai}_{vu} \cap A^{i}_{v\overleftarrow{u}}$ is an easy fact. Thus, it only remains to argue that we may assume that if $v \overrightarrow{\succ} u$, then the equality $A^{ai}_{v\overleftarrow{u}} = A^{ai}_{vu} \cap A^{i}_{v\overleftarrow{u}}$ holds. Note that the inclusion $A^{ai}_{v\overleftarrow{u}} \subseteq A^{ai}_{vu} \cap A^{i}_{v\overleftarrow{u}}$ is also an easy fact. To this end, observe that if $v \overrightarrow{\succ} u$, then it is not possible to assign an interval in $A^{ai}_{vu} \cap A^{i}_{v\overleftarrow{u}}$ to a period in $P_{\overrightarrow{u}}$. Hence, we may even assume that $a^{i}_{v\overleftarrow{u}} = a^{i}_{vu}$, which implies that $A^{ai}_{v\overleftarrow{u}} = A^{ai}_{vu} \cap A^{i}_{v\overleftarrow{u}}$. This completes the proof of the claim. ☐

Example. Consider the interval set $A^i_{vu} = A^i_v = \{I_1, \ldots, I_6\}$ depicted in Figure 4 with $v \overleftarrow{\succ} u$, where these intervals are ordered according to their left endpoints. Since these intervals have the same level i, the vertical separation is only used to distinguish them. Finally, recall that all intervals in A^i_{vu} contain the two periods a_v and b_v as schematically illustrated in Figure 4. We have that $A^i_{v\overleftarrow{u}} = \{I_1, I_2, I_3\}$ and $A^i_{v\overrightarrow{u}} = \{I_4, I_5, I_6\}$. Consider now a normal schedule σ that satisfies some integer tuple a, where the intervals A^i_{vu} are assigned by σ to three different periods according to the fat dots. Hence, we have that $a^i_{vu} = a^i_{v\overrightarrow{u}} = 5$ and $a^i_{v\overleftarrow{u}} = 2$. Note that the conditions in Lemma 4 are satisfied, but the inclusions are not tight, since interval I_3 is assigned to a period in $P_{\overrightarrow{u}}$ and interval I_6 is assigned to a period outside of P_u.

The following lemma allows us to only consider normal schedules, which we will use to prove the correctness of the dynamic programming approach introduced in Subsection 4.3.

Lemma 5. *For any schedule σ, there is a normal schedule σ' assigning the same intervals with* $\mathrm{cost}(\sigma') = \mathrm{cost}(\sigma)$.

Proof. The claim follows from a simple swapping argument. Let A be the tuple that is satisfied by σ. Consider a fixed vertex pair $v \succ u$ and level i. Since the case



(a) before (b) after

Figure 5: Swapping two periods $\sigma(I)$ and $\sigma(I')$.

$v \overrightarrow{\succ} u$ works analogously, assume that $v \overleftarrow{\succ} u$, and moreover assume that there are two intervals $I, I' \in A^i_{vu}$ assigned by σ with the property that the left endpoint of I is smaller than the left endpoint of I', but $\sigma(I) \notin P_u$ and $\sigma(I') \in P_u$, which is exactly the property that avoids that σ is normal. We illustrate this scenario in Subfigure 5(a). However, by the definition of A^i_{vu} and the fact that σ satisfies A, we have that $\sigma(I') \in I$ and $\sigma(I) \in I'$, and therefore, we can swap the respective periods $\sigma(I')$ and $\sigma(I)$ where the intervals I' and I are assigned to without modifying $\mathrm{cost}(\sigma)$ as illustrated in Subfigure 5(b). By iterating this swapping argument, we obtain the claimed normal schedule σ'. $\square$

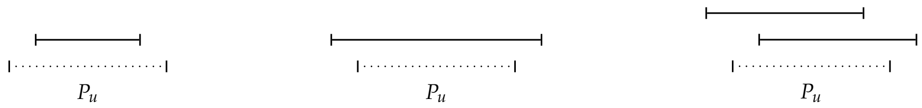
4.3 A quasipolynomial time algorithm for easy instances

In this subsection, we present a dynamic programming based quasipolynomial time algorithm for easy instances. Recall that if J is easy, then we know a tuple A such that there is an optimal schedule σ^* that satisfies A. Hence, we can think of A as a global variable. We will show in Subsection 4.4 how to get rid of this assumption. Let x always denote an integer tuple with $0 \leq x_i \leq n$ for each level i.

Let Π be the dynamic programming array, which, for each vertex u, integer tuple x, and u-based integer tuple a includes an entry $\Pi(u, x, a, A)$ that contains the cost of an optimal schedule σ assigning the intervals

$$\Phi(u, x, a, A) := J[\min P_u, \max P_u] \cup [1, T]^x \cup \bigcup_{i=1}^{m} \bigcup_{v \in R : v \succ u} A^{ai}_{vu}$$

to periods in P_u subject to the constraint that σ satisfies A. The first part $J[\min P_u, \max P_u]$ is the subinstance of all intervals $I \in J$ with $I \subseteq [\min P_u, \max P_u]$, where $\min P_u$ and $\max P_u$ are the minimal and maximal periods in P_u, respectively, and the second part $[1, T]^x$ stands for a multiset of intervals that, for each level i, contains exactly x_i copies of interval $[1, T]$ with level i. Note that such an interval



(a) one (b) two (c) three

Figure 6: Three types of intervals.

can be assigned to any period in P_u. Observe that the first part resembles the subinstances used in the dynamic program for $k = \infty$ [7, 2] described in Subsection 2.1. However, we add some intervals of the form $[1, T]$ with different levels in the second part, which can be assigned to any period. Such an interval will act as a representative for an interval $I \in J$ with $P_u \subset I$. Finally, the intervals in the third part are intervals with the property that exactly one endpoint is contained in P_u. To summarize this, Figure 6 illustrates these three types of intervals corresponding to these three parts with respect to P_u.

Since R has $2T - 1 = \mathcal{O}(n)$ many vertices, m is constant, and a has at most $m \log T = \mathcal{O}(\log n)$ many dimensions, we conclude that the size of Π is at most

$$(2T - 1) \cdot n^m \cdot n^{m \log T} = n^{\mathcal{O}(\log n)}, \tag{4}$$

which gives quasipolynomial running time if can somehow inductively fill this array.

To fill array Π, for each non-leaf vertex u, use the recurrence relation

$$\Pi(u, x, a, A) = \min_{\overleftarrow{x}, \overrightarrow{x}, \overleftarrow{a}, \overrightarrow{a}} \{\Pi(\overleftarrow{u}, \overleftarrow{x}, \overleftarrow{a}, A) + \Pi(\overrightarrow{u}, \overrightarrow{x}, \overrightarrow{a}, A)\}, \tag{5}$$

where, for each level i, we have the constraints

$$a^i_{v\overleftarrow{u}} \in \{0, \dots, n\} \text{ for } v \succ u \quad \text{s.t.} \quad A^{ai}_{v\overleftarrow{u}} \begin{cases} \subseteq A^{ai}_{vu} \cap A^i_{v\overleftarrow{u}} & \text{if } v \overleftarrow{\succ} u, \\ = A^{ai}_{vu} \cap A^j_{v\overleftarrow{u}} & \text{if } v \overrightarrow{\succ} u, \end{cases} \tag{6}$$

$$a^i_{v\overrightarrow{u}} \in \{0, \dots, n\} \text{ for } v \succ u \quad \text{s.t.} \quad A^{ai}_{v\overrightarrow{u}} \begin{cases} \subseteq A^{ai}_{vu} \cap A^i_{v\overrightarrow{u}} & \text{if } v \overrightarrow{\succ} u, \\ = A^{ai}_{vu} \cap A^i_{v\overrightarrow{u}} & \text{if } v \overleftarrow{\succ} u, \end{cases}$$

and

$$\overleftarrow{y}_i \in \{0, \dots, n\} \text{ and } \overrightarrow{y}_i \in \{0, \dots, n\} \quad \text{s.t.} \quad x_i = \overleftarrow{y}_i + \overrightarrow{y}_i, \tag{7}$$

and

$$
\begin{aligned}
\overleftarrow{x}_i &= \sum_{v \in R : v \overrightarrow{\succ} u} |(A^{ai}_{vu} \cap A^{i}_{v\overrightarrow{u}}) \setminus A^{\overrightarrow{a}i}_{v\overrightarrow{u}}| + \overleftarrow{y}_i, \\
\overrightarrow{x}_i &= \sum_{v \in R : v \overleftarrow{\succ} u} |(A^{ai}_{vu} \cap A^{i}_{v\overleftarrow{u}}) \setminus A^{\overleftarrow{a}i}_{v\overleftarrow{u}}| + \overrightarrow{y}_i.
\end{aligned}
\tag{8}
$$

Observe that Constraints (6) correspond to Lemmas 4. In Constraints (7), we simply split x_i into two integral parts, namely $\overleftarrow{y}_i$ and $\overrightarrow{y}_i$. Intuitively, the sums in Constraints (8) are not zero if the inclusions in Constraints (6) are not always tight.

Finally, to initiate array Π, for each leaf u, we set

$$\Pi(u, x, a, A) := \omega_{\Phi(u,x,a,A)}, \tag{9}$$

where the right side is defined in Subsection 2.2, since P_u contains only a single period, and hence $\Phi(u, x, a, A)$ forms a batch of arbitrary size. This completes the definition of the dynamic program. Note that in contrast to the initialization of Π in (9), we treat all levels independently during Recurrence Relation (5). Specifically, we have a seperate set of constraints for each level.

The following lemma gives the correctness of this approach, since this lemma implies that, if u is the root and x the integer tuple that contains only 0s, then $\Pi(u, x, a, A)$ contains the cost of an optimal schedule σ subject to the constraint that σ satisfies A. Recall here that an u-based integer tuple a is empty if u is the root. Moreover, recall the assumption that there is an optimal schedule σ^* satisfying A, and hence $\text{cost}(\sigma) = \text{OPT}$.

Lemma 6. *For each vertex u, integer tuple x, and u-based integer tuple a, we have that $\Pi(u, x, a, A)$ is indeed the cost of an optimal schedule σ which assigns the intervals $\Phi(u, x, a, A)$ to periods in P_u subject to the constraint that σ satisfies A.*

Proof. We will prove the claim via induction on the depth d_u of a vertex u, which mimics the proceeding of the dynamic program, and we start with the leaves.

Induction start. If u is a leaf, then there is only a single period in P_u, and hence, we have no choice where to assign each interval in batch $\Phi(u, x, a, A)$. The induction start follows.

Induction step. Consider some non-leaf vertex u, and assume as induction hypothesis that the claim holds for the its two children, namely $\overleftarrow{u}$ and $\overrightarrow{u}$. Moreover, consider some integer tuple x and u-based integer tuple a. By the induction hypothesis, we only have to show that there are parameters $\overleftarrow{x}$, $\overrightarrow{x}$, $\overleftarrow{a}$, and $\overrightarrow{a}$ that satisfy the following two properties:

(1) Constraints (6), (7), and (8) are satisfied with respect to x and a,

(2) If we have two optimal schedules $\overleftarrow{\sigma}$ and $\overrightarrow{\sigma}$ assigning the intervals $\Phi(\overleftarrow{u}, \overleftarrow{x}, \overleftarrow{a}, A)$ and $\Phi(\overrightarrow{u}, \overrightarrow{x}, \overrightarrow{a}, A)$ to periods in $P_{\overleftarrow{u}}$ and $P_{\overrightarrow{u}}$, respectively, subject to the constraint that both schedules satisfy A, then we can construct an optimal schedule σ which assigns the intervals $\Phi(u, x, a, A)$ to periods in P_u subject to the constraint that σ satisfies A, and $\text{cost}(\sigma) = \text{cost}(\overleftarrow{\sigma}) + \text{cost}(\overrightarrow{\sigma})$.

This implies that $\Pi(u, x, a, A)$ is filled correctly if all entries $\Pi(\overleftarrow{u}, \overleftarrow{x}, \overleftarrow{a}, A)$ and $\Pi(\overrightarrow{u}, \overrightarrow{x}, \overrightarrow{a}, A)$ are filled correcty. The induction step follows.

Consider some optimal schedule σ' which assigns the intervals $\Phi(u, x, a, A)$ to periods in P_u subject to the constraint that σ' satisfies A. By Lemma 5, we may assume that σ' is normal with respect to all intervals in $\Phi(u, x, a, A)$ except the ones of the form $[1, T]$. This implies that we can extend the u-based integer tuple a to all vertex pairs $v \succ \overleftarrow{u}$ and $v \succ \overrightarrow{u}$ such that σ' still satisfies a. Let then $\overleftarrow{a}$ and $\overrightarrow{a}$ be two $\overleftarrow{u}$- and $\overrightarrow{u}$-based integer tuples, where, for each vertex $v \succ u$ and level i, we define

$$\overleftarrow{a}^i_{v\overleftarrow{u}} := a^i_{v\overleftarrow{u}} \text{ and } \overrightarrow{a}^i_{v\overrightarrow{u}} := a^i_{v\overrightarrow{u}},$$

respectively. By Lemma 4, we have that $\overleftarrow{a}$ and $\overrightarrow{a}$ satisfy Constraints (6). Finally, for each level i, let $\overleftarrow{y}_i$ and $\overrightarrow{y}_i$ be the number of intervals in $[1, T]^x$ with level i which σ' assigns to periods in $P_{\overleftarrow{u}}$ and $P_{\overrightarrow{u}}$, respectively. This satisfies Constraints (7). Choose then the two integer tuples $\overleftarrow{x}$ and $\overrightarrow{x}$ such that Constraints (8) are satisfied with respect to $\overleftarrow{y}$, $\overrightarrow{y}$, $\overleftarrow{a}$, $\overrightarrow{a}$, and a. By combining all this, we obtain Property (1).

We still have to show Property (2), i.e., given two arbitrary schedules $\overleftarrow{\sigma}$ and $\overrightarrow{\sigma}$ as described in Property (2) with respect to the constructed parameters $\overleftarrow{x}$, $\overrightarrow{x}$, $\overleftarrow{a}$, and $\overrightarrow{a}$, respectively, we need to show that we can construct a schedule σ with $\text{cost}(\sigma) = \text{cost}(\sigma')$. To this end, consider a fixed level i. Observe that

$$(J[\min P_u, \max P_u] \cap J_i) \cup ([1, T]^{x_i}) \cup \bigcup_{v \in R : v \succ u} A^{ai}_{vu}$$

are the intervals in $\Phi(u, x, a, A)$ with level i which σ' assigns to periods in P_u, where $[1, T]^{x_i}$ denotes the subset of all x_i many intervals in $[1, T]^x$ with level i. We will iteratively decompose this set into two parts, named $\overleftarrow{J}$ and $\overrightarrow{J}$, that contain exactly the intervals which are assigned to periods in $P_{\overleftarrow{u}}$ and $P_{\overrightarrow{u}}$, respectively.

- The intervals in $J[\min P_u, \max P_u] \cap J_i$ which are assigned to periods in $P_{\overleftarrow{u}}$ and $P_{\overrightarrow{u}}$ are exactly

$$A_u^i \cup (J[\min P_{\overleftarrow{u}}, \max P_{\overleftarrow{u}}] \cap J_i) \text{ and } R_u^i \backslash A_u^i \cup (J[\min P_{\overrightarrow{u}}, \max P_{\overrightarrow{u}}] \cap J_i), \quad (10)$$

respectively. The first part on both sides is due to the fact that σ' satisfies A. Therefore, add the left and right side of (10) to $\overleftarrow{J}$ and $\overrightarrow{J}$, respectively.

- Add all intervals in $[1, T]^x \cap J_i$ which are assigned to a period in $P_{\overleftarrow{u}}$ and $P_{\overrightarrow{u}}$ to $\overleftarrow{J}$ and $\overrightarrow{J}$, respectively.

- We have that the intervals in

$$\bigcup_{v \in R: v \overleftarrow{\succ} u} A_{vu}^{ai} \text{ and } \bigcup_{v \in R: v \overrightarrow{\succ} u} A_{vu}^{ai} \quad (11)$$

which are assigned to periods in $P_{\overleftarrow{u}}$ and $P_{\overrightarrow{u}}$ are exactly

$$\bigcup_{v \in R: v \overleftarrow{\succ} u} A_{v\overleftarrow{u}}^{ai} = \bigcup_{v \in R: v \overleftarrow{\succ} u} A_{v\overleftarrow{u}}^{\overleftarrow{a}i} \text{ and } \bigcup_{v \in R: v \overrightarrow{\succ} u} A_{v\overrightarrow{u}}^{ai} = \bigcup_{v \in R: v \overrightarrow{\succ} u} A_{v\overrightarrow{u}}^{\overrightarrow{a}i}, \quad (12)$$

respectively. Recall here that we extended the integer tuple a to all vertex pairs $v \succ \overleftarrow{u}$ and $v \succ \overrightarrow{u}$. Therefore, add the left and right side of (12) to $\overleftarrow{J}$ and $\overrightarrow{J}$, respectively.

- On the other hand, the intervals in the sets (11) which are assigned to periods in $P_{\overrightarrow{u}}$ and $P_{\overleftarrow{u}}$ are exactly the 'complements' of (12), which are

$$\bigcup_{v \in R: v \overleftarrow{\succ} u} (A_{vu}^{ai} \cap A_{v\overleftarrow{u}}^{i}) \backslash A_{v\overleftarrow{u}}^{\overleftarrow{a}i} \text{ and } \bigcup_{v \in R: v \overrightarrow{\succ} u} (A_{vu}^{ai} \cap A_{v\overrightarrow{u}}^{i}) \backslash A_{v\overrightarrow{u}}^{\overrightarrow{a}i}, \quad (13)$$

respectively. Now note that each interval I in the left and right side of (13) satisfies $P_{\overrightarrow{u}} \subset I$ and $P_{\overleftarrow{u}} \subset I$, respectively, and can hence be assigned to any period in $P_{\overrightarrow{u}}$ and $P_{\overleftarrow{u}}$, respectively. Therefore, for each such interval I, add one copy of interval $[1, T]$ with level i to $\overrightarrow{J}$ and $\overleftarrow{J}$, respectively.

By the definitions of $\overleftarrow{J}$ and $\overrightarrow{J}$ above, we have that

$$\overleftarrow{J} = \Phi(\overleftarrow{u}, \overleftarrow{x}, \overleftarrow{a}, A) \text{ and } \overrightarrow{J} = \Phi(\overrightarrow{u}, \overrightarrow{x}, \overrightarrow{a}, A),$$

and hence, we see that $\overleftarrow{\sigma}$ and $\overrightarrow{\sigma}$ correspond to schedules that assign the intervals which σ' assigns to periods in $P_{\overleftarrow{u}}$ and $P_{\overrightarrow{u}}$, respectively. Thus, since $\overleftarrow{\sigma}$, $\overrightarrow{\sigma}$, and σ' are optimal, we find that

$$\operatorname{cost}(\overleftarrow{\sigma}) + \operatorname{cost}(\overrightarrow{\sigma}) = \operatorname{cost}(\sigma').$$

Therefore, by combining the schedules $\overleftarrow{\sigma}$ and $\overrightarrow{\sigma}$, we obtain the claimed schedule σ with $\mathrm{cost}(\sigma) = \mathrm{cost}(\sigma')$. $\square$

4.4 A QPTAS

In this subsection, we extend the dynamic program for easy instances from Subsection 4.3 to general instances. Recall that if J is easy, then we know a tuple A such that there is an optimal schedule σ^* that satisfies A. However, in general, we do not know such a tuple A.

Let K always denote a tuple of sets of interval sets with $K_v^i \subseteq \mathcal{P}(R_v^i)$ for each vertex v and level i, where $\mathcal{P}(R_v^i)$ denotes the power set of R_v^i. Define $|K| := \max_{v,i} |K_v^i|$. For a tuple A, we write $A \in K$ if $A_v^i \in K_v^i$ for each vertex v and level i. Analogously, for a u-based tuple A, we write $A \in K$ if this property holds for each vertex $v \succ u$, and we say that a schedule σ *satisfies* a K-extension of A if we can extend A to all vertex pairs such that still $A \in K$ and σ satisfies A. We need the following lemma to deal with the left-right assignment dilemma, which is proven in the end of this subsection.

Lemma 7. *For any $\epsilon > 0$, we can compute a tuple K with $|K| \leq c$ for some constant c in polynomial time such that there is a tuple $A \in K$ and a schedule σ that satisfies A with* $\mathrm{cost}(\sigma) \leq (1 + \epsilon)\mathrm{OPT}$.

To see how to apply Lemma 7, Recall that, for some vertex v, if we want to decompose J between periods a_v and b_v, then, for each level i, we need to decide which intervals in R_v^i should be assigned to the left or right. If J is easy, then we know a tuple A that guides this assignment. However, if we do not know such a tuple A, then we could enumerate all such tuples, which is not feasible in general, since there are two many ways to select a subset $A_v^i \subseteq R_v^i$. However, Lemma 7 basically says that by losing an $(1 + \epsilon)$-factor, we only have to consider a constant number of subsets, namely all subsets in K_v^i. In combination with the fact that R has logarithmic depth $\log T = \mathcal{O}(\log n)$, this allows us to incorporate an approximate enumeration of all tuples A in the dynamic program.

More specifically, assume that we are given a tuple K as described in Lemma 7, and hence, instead of a global variable A as in Subsection 4.3, we have a global variable K. We extend array Π such that, for each vertex u, integer tuple x, u-based integer tuple a, and u-based tuple $A \in K$, $\Pi(u, x, a, A)$ contains the cost of an optimal schedule assigning the intervals $\Phi(u, x, a, A)$ to periods in P_u subject to the constraint that σ satisfies a K-extension of A. Because $|K| \leq c$ and A has at most $m \log T = \mathcal{O}(\log n)$ dimensions, we find that the size of Π listed in (4) only increases by the polynomial factor

$$c^{m \log T} = n^{\mathcal{O}(1)},$$

and hence, array Π has still quasipolynomial size.

To fill the extended array Π, instead of (5), we use the extended recurrence relation

$$\Pi(u, x, a, A) = \min_{\overleftarrow{x}, \overrightarrow{x}, \overleftarrow{a}, \overrightarrow{a}, \overline{A}} \left\{ \Pi(\overleftarrow{u}, \overleftarrow{x}, \overleftarrow{a}, \overline{A}) + \Pi(\overrightarrow{u}, \overrightarrow{x}, \overrightarrow{a}, \overline{A}) \right\}$$

subject to Constraints (6), (7) and (8), but we have the additional constraints that, for each level j,

$$\overline{A}^i_v \begin{cases} \in K^i_u & \text{for } v = u, \\ = A^i_v & \text{for } v \succ u. \end{cases} \tag{14}$$

Note that A is now a parameter in the array Π instead of a global variable. The new Constraints (14) are there to restrict the parameter $\overline{A}$, which is a $\overleftarrow{u}$- and $\overrightarrow{u}$-based tuple, with respect to A and K such that $\overline{A} \in K$.

The correctness of this extension can be straightforward proven by extending the induction used to prove Lemma 6. Therefore, if u is the root and x the tuple that contains only 0s, then $\Pi(u, x, a, A)$ contains now the cost of an optimal schedule σ subject to the constraint that σ satisfies a K-extension of A. Note that a u-based tuple A is empty if u is the root, and hence, this K-extension can be any tuple $A \in K$. Because we choose K according to Lemma 7, this shows that the extension of the dynamic program introduced in this subsection yields a QPTAS for CB_k.

To prove Lemma 7, we first need to prove the following two lemmas. For simplicity, in what follows, let A sometimes also denote an interval set and K a set of interval sets, which we can interpret as tuples with one dimension. The first lemma is general helper lemma that will be used later on again, and the second lemma is a 'local' version of Lemma 7. Recall parameter α from Subsection 3.

Lemma 8. *Consider two tuples A and a, and, for some $\epsilon > 0$, let σ be a schedule which satisfies the following properties:*

(1) σ assigns all intervals in J except an ϵ-fraction,

(2) σ satisfies A (and a),

(3) $\mathrm{cost}(\sigma) \leq \mathrm{OPT}$.

Then there is a schedule σ satisfying A (and a) with $\mathrm{cost}(\sigma) \leq (1 + \epsilon k \alpha^m \mathrm{OPT}$.

Proof. Since there at most k intervals in each batch, and each batch C has at least weight $w_C = \alpha^0 = 1$, we obtain the simple lower bound

$$\mathrm{OPT} \geq \frac{n}{k}. \tag{15}$$

Consider a schedule σ satisfying the three prerequisites in the claim with respect to some $\epsilon > 0$. Then, for each of the at most ϵn many intervals I which are not assignment by σ, we create a batch only containing I. We can do this such that σ still satisfies A (and a). Due to this additionally assigning of intervals, $\mathrm{cost}(\sigma)$ increases by at most $\alpha^m \epsilon n$, which gives with Inequality (15) that now

$$\mathrm{cost}(\sigma) \leq \mathrm{OPT} + \alpha^m \epsilon n \leq (1 + \epsilon k \alpha^m)\,\mathrm{OPT}.$$

The claim follows. This is the only lemma where we need that k is constant. □

Lemma 9. *For any $\epsilon > 0$ and any interval set $J' \subseteq J$ with the property that each interval in J' contains two fixed consecutive periods a and $b = a + 1$, we can compute a set K of subsets of J' with $|K| \leq 4^{1/\epsilon}$ in polynomial time such that, for any schedule σ' which assigns exactly the intervals J', there exists a set $A \in K$ and a schedule σ that satisfy the following properties:*

(1) σ assigns all intervals in J' except an ϵ-fraction,

(2) for each interval $I \in A$, if σ assigns I, then $\sigma(I) \leq a$, and, for each interval $I \in J' \setminus A$, if σ assigns I, then $\sigma(I) \geq b$,

(3) for each period t, the number of intervals in J' that σ assigns to t is at most the number of intervals in J' that σ' assigns to t, i.e., $|\sigma^{\leftarrow}(t)| \leq |\sigma'^{\leftarrow}(t)|$.

Proof of Lemma 7. Consider some optimal schedule σ^* with $\mathrm{cost}(\sigma^*) = \mathrm{OPT}$. For any $\epsilon > 0$, we can simultaneously apply Lemma 9 to all vertices v and levels i, where we use $J' = R_v^i$, $a = a_v$, $b = b_v$, and σ' is the restriction of σ^* to R_v^i. Then we combine the resulting sets of interval sets $K_v^i = K$ to a tuple of sets of interval sets, also named K, and the resulting schedules σ to a single one, also named σ. The three properties in Lemma 9 imply that σ satisfies the respective prerequisites in Lemma 8, which completes the proof of the claim, since $k\alpha^m$ is constant, but we may choose $\epsilon > 0$ arbitrarily small. □

We prove Lemma 9 in the remainder of this subsection. Given σ', we will first show how to explicitly construct a set A and a schedule σ that satisfy all three properties in Lemma 9 with respect to σ'. Note that if we know σ', then constructing such a set A is not difficult in general. However, to prove Lemma 9, we need to be independent of σ'. Therefore, without knowing σ', we show afterwards how to construct a set of interval sets K that contains any such set A.

Consider a fixed but arbitrary small $\epsilon > 0$, and assume that $1/\epsilon$ is integral and $1/\epsilon \leq N := |J'|$. Otherwise, if $1/\epsilon > N$, then $4^{1/\epsilon} > |\mathcal{P}(J')| = 2^N$, where $\mathcal{P}(J')$ is the power set of J', and hence, we may select $K := \mathcal{P}(J')$. Assume then moreover that $1/\epsilon$ divides N.

We use the following algorithm, named $\mathcal{A}$, to construct A and σ. For the sake of exposition, this algorithm also constructs the complement $B = J' \backslash A$, and consequently, (A, B) is a 2-partition of J'. Since we will use these notions in algorithm $\mathcal{A}$, note that, for example, the two *leftmost intervals with respect to their right endpoints* in Figure 3 are I_1 and I_3, and the two *rightmost intervals with respect to their left endpoints* are I_3 and I_4.

$\mathcal{A}(J', \sigma', \epsilon)$

---

1. Set $A \leftarrow \emptyset$, $B \leftarrow \emptyset$, and $J'' \leftarrow \emptyset$.
2. Set $J^- \leftarrow \{I \in J' \mid \sigma'(I) \leq a\}$ and $J^+ \leftarrow \{I \in J' \mid \sigma'(I) \geq b\}$.
3. While $J' \neq \emptyset$:
   (a) If $|J'' \cap J^-| \geq |J'' \cap J^+|$, then set $J^* \leftarrow J^-$, add the ϵN leftmost intervals in J' with respect to their right endpoints to A, and remove these intervals from J'.
   (b) Otherwise, if $|J'' \cap J^-| < |J'' \cap J^+|$, then set $J^* \leftarrow J^+$, add the ϵN rightmost intervals in J' with respect to their left endpoints to B, and remove these intervals from J'.
   (c) For each interval I added to $A \cup B$ in the last two steps:
      i. If $I \in J^*$, then set $\sigma(I) \leftarrow \sigma'(I)$.
      ii. Otherwise, if $I \notin J^*$, but there is an interval $I' \in J'' \cap J^*$ with $\sigma'(I') \in I$, then set $\sigma(I) \leftarrow \sigma'(I')$, remove I' from J'', and add I to J''.
      iii. Finally, if none of the two cases above applies to I, then leave $\sigma(I)$ undefined, and add I to J''.
4. Return (A, B) and σ.

In what follows, an *iteration* is an iteration of the loop in Step 3. In each iteration, we remove some intervals from J', and, for each interval $I \in J'$, we have that $\sigma(I)$ takes its final value in the iteration when it is removed from J'. However, if we add I to J'', then $\sigma(I) \neq \sigma'(I)$, where it is also possible that $\sigma(I)$ is left undefined. We can think of such an interval $I \in J''$ as an interval whose 'capacity' $\sigma'(I)$ has not been used yet. Recall that we remove ϵN many intervals from S' in each iteration. Therefore, since we assume that $1/\epsilon$ divides N, we have exactly $1/\epsilon$ iterations. Finally, note that, in each iteration, either $J^* = J^-$ or $J^* = J^+$, depending on whether $|J'' \cap J^-| \geq |J'' \cap J^+|$ or $|J'' \cap J^-| < |J'' \cap J^+|$. This technical detail allows us to use Step 3c in both cases.

Lemma 10. *Just before each iteration, it holds for each interval pair $I' \in J''$ and $I \in J'$ that we can assign I to period $\sigma'(I')$, i.e., $\sigma'(I') \in I$.*

Proof. We show the claim of the lemma via induction on the iterations.

Induction start. The induction claim trivially holds just before the first iteration, since then $J'' = \emptyset$.

Induction step. Assume as induction hypothesis that the claim holds just before an arbitrary iteration. We need to show that it then still holds just before the next iteration. To this end, consider some intervals I' added to J'' in this iteration, and assume that $|J'' \cap J^-| \geq |J'' \cap J^+|$, and hence $J^* = J^-$. Note that it is necessary that $I' \notin J^-$, since otherwise, we would not have added I' to J'', but we would have set $\sigma(I') \leftarrow \sigma'(I')$ in Step 3c. Consequently, we have that $I' \in J^+$, and hence $\sigma'(I') \geq b$. On the other hand, since we select the leftmost intervals with respect to their right endpoints in Step 3a, we have that the the right endpoint of I' is smaller than the right endpoint of any remaining interval $I \in J'$. Combining these two facts gives that even $\sigma'(I') \in I$, which proves the induction step, and hence the claim of the lemma. ☐

Lemma 11. *The returned 2-partition (A, B) and schedule σ satisfy the following properties:*

(1) σ assigns all intervals in J' except an 2ϵ-fraction,

(2) for each interval $I \in A$, if σ assigns I, then $\sigma(I) \leq a$, and, for each interval $I \in B$, if σ assigns I, then $\sigma(I) \geq b$,

(3) for each period t, the number of intervals that σ assigns to t is at most the number of intervals that σ' assigns to t.

Proof. Properties (2) and (3) immediately follow from the definition of algorithm $\mathcal{A}$. Therefore, we only need to show Property (1). It is easy to see that the number of unassigned intervals in J' is exactly the final value $|J''|$. Therefore, we show via induction on the iterations that $|J''| \leq 2\epsilon N$ after each iteration.

Induction start. Since initially $J'' = \emptyset$, and at most ϵN intervals are added to J'' in each iteration, we have that even $|J''| \leq \epsilon N$ after the first iteration.

Induction step. Assume as induction hypothesis that $|J''| \leq 2\epsilon m$ after an arbitrary iteration. We have to show that this does not change during the next iteration. Assume that $|J'' \cap J^-| \geq |J'' \cap J^+|$, and hence $J^* = J^-$. Clearly, all new intervals added to J'' during the next iteration are in J^+. On the other hand, by Lemma 10, it holds for each $I' \in J'' \cap J^-$ and each interval I added to $J'' \cap J^+$ during the next iteration that $\sigma'(I') \in I$. Therefore, by the definition of Step 3c, every time an interval is added to $J'' \cap J^+$, one interval is removed from $J'' \cap J^-$ if still possible, i.e., if still $J'' \cap J^- \neq \emptyset$. Since $|J'' \cap J^-| \geq |J'' \cap J^+|$, $|J''| \leq 2\epsilon m$, and at most ϵm intervals are added to J'' in the next iteration, this shows that still $|J''| \leq 2\epsilon m$ after the next iteration. This proves the induction step. ☐

We know from Lemma 11 that set A and schedule σ returned by algorithm $\mathcal{A}$ have exactly the properties needed for Lemma 9. The following algorithm constructs a set of 2-partitions K' that contains the returned 2-partition (A, B), and hence, it contains such a 2-partition for any schedule σ'. This gives us the required set of subsets $K := \{A \mid (A, B) \in K'\}$.

$\mathcal{A}'(J', \epsilon)$

---

1. Set $K' \leftarrow \emptyset$.
2. For each tuple $w \in \{0, 1\}^{1/\epsilon}$:
   (a) Set $A \leftarrow \emptyset$ and $B \leftarrow \emptyset$.
   (b) For $j = 1, \ldots, 1/\epsilon$:
      i. If $w_j = 1$, then add the ϵN leftmost intervals in J' with respect to their right endpoints to A, and remove them from J'.
      ii. Otherwise, if $w_j = 0$, then add the ϵN rightmost intervals in J' with respect to their left endpoints to B, and remove them from J'.
   (c) Add (A, B) to K'.
3. Return K'.

Proof of Lemma 9. The returned set of 2-partitions K' has clearly size $2^{1/\epsilon}$, since this is the number of tuples in $\{0,1\}^{1/\epsilon}$. On the other hand, note that K' contains the 2-partition (A, B) computed by algorithm $\mathcal{A}$, and by Lemma 11, the set A contained in this 2-partition has the claimed properties. Combining this proves the claim. □

4.5 A PTAS for easy instances

In this subsection, we extend the dynamic program for easy instances described in Subsection 4.3 to a PTAS for easy instances. Recall that if J is easy, then we know a tuple A such that there is an optimal schedule σ^* that satisfies A. However, we have already shown in Subsection 4.4 how to get rid of this assumption, but, for the sake of exposition, we assume in this subsection again that J is easy.

Observe that the base n of the quasipolynomial running time listed in (4) is due to the fact that we only have the bound $0 \leq a^i_{vu} \leq n$ on the entries of the integer tuple a. But if we would even have a bound $0 \leq a^i_{vu} \leq c$ for some constant c, then we would immediately obtain an array Π of polynomial size, and hence polynomial running time. However, restricting the search space in this way does clearly not yield an approximation scheme. A more general way to restrict the search space is to only restrict the number of different values each entry of a may take. To this end, we say that an integer tuple a is *c-restricted* for some positive integer c if, for each vertex pair $v \succ u$ and level i, it satisfies

$$a^i_{vu} \in B^{ci}_{vu} := \{y \in z\mathbb{N} \mid r - z \leq y \leq s + z\},$$

where z is the smallest power of 2 such that $|A^i_{vu}|/z \leq 2c$, and the integers $0 \leq r \leq s \leq n$ are such that $\{I_r, I_{r+1}, \ldots, I_s\} = A^i_{vu}$ as given in (2). We additionally require that $a^j_{vu} = \max B^{ci}_{vu}$ if $v = \hat{u}$, which ensures that Equation (3) holds as well for c-restricted integer tuples. On the other hand, note that if $a^i_{vu} = \min B^{ci}_{vu}$, then $A^{ai}_{vu} = \emptyset$. The following lemma, which is proven in the end of this subsection, states that c-restricted tuples yield an arbitrary good approximation.

Lemma 12. *For any $\epsilon > 0$, there is a constant c such that there is a c-restricted integer tuple a and a schedule σ satisfying A and a with* $\mathrm{cost}(\sigma) \leq (1 + \epsilon)\mathrm{OPT}$.

We can restrict the dynamic program to c-restricted integer tuples by adding the constraints that, for each level i,

$$\begin{aligned} \overleftarrow{a}^{i}_{v\overleftarrow{u}} &\in B^{ci}_{v\overleftarrow{u}} \text{ for } v \succ u, \\ \overrightarrow{a}^{i}_{v\overrightarrow{u}} &\in B^{ci}_{v\overrightarrow{u}} \text{ for } v \succ u. \end{aligned} \tag{16}$$

In this case, since each entry a^i_{vu} of a c-restricted integer tuple a may take at most $2c+2$ many different values, in contrast to (4), we obtain an array Π of polynomial size

$$(2T-1) \cdot n^m \cdot (2c+2)^{m \log T} = n^{\mathcal{O}(1)}.$$

However, we need to argue that adding the new Constraints (16) is indeed correct. To this end, consider some fixed vertex pair $v \succ u$ and level i. Since $|A^i_{vu}| \geq |A^i_{v\overleftarrow{u}}|, |A^i_{v\overrightarrow{u}}|$, we have that $B^{ci}_{vu} \subseteq B^{ci}_{v\overleftarrow{u}} \cup B^{ci}_{v\overrightarrow{u}}$. Using this, it is easy to see that Lemma 4 holds as well for a schedule σ satisfying a c-restricted integer tuple a. Therefore, by extending the induction used to prove Lemma 6, we can straightforward prove that, if u is the root and x the tuple that contains only 0s, then $\Pi(u, x, a, A)$ contains now the cost of an optimal schedule σ subject to the constraint that σ satisfies A and a c-restricted integer tuple. Together with Lemma 12, this shows that the extension of the dynamic program introduced in this subsection yields a PTAS for CB_k with easy instances. Moreover, in combination with the extension introduced in Subsection 4.4, this finally proves Theorem 3.

In the remainder of this subsection, we will prove Lemma 12, whereas we use b to denote the claimed c-restricted integer tuple. Let σ^* be an optimal schedule that satisfies A. We know from Lemma 5 that we can assume that σ^* also satisfies an (unrestricted) integer tuple a. The natural way to construct b from a is the following inductive greedy approach, where the constant c will be defined later with respect to some $\epsilon > 0$: To start this inductive construction recall that $b^i_{u\overleftarrow{u}}$ and $b^i_{u\overrightarrow{u}}$ are already defined for each non-leaf vertex u. Now, consider a vertex pair $v \succ u$ with $v \neq \hat{u}$, and assume that b^i_{vu} is already defined. Choose then $b^i_{v\overleftarrow{u}}$ and $b^i_{v\overrightarrow{u}}$ maximal subject to

$$A^{bi}_{v\overleftarrow{u}} \begin{cases} \subseteq A^{bi}_{vu} \cap A^{ai}_{v\overleftarrow{u}} & \text{for } v \overleftarrow{\succ} u, \\ = A^{bi}_{vu} \cap A^{i}_{v\overleftarrow{u}} & \text{for } v \overrightarrow{\succ} u, \end{cases} \tag{17}$$

$$A^{bi}_{v\overrightarrow{u}} \begin{cases} \subseteq A^{bi}_{vu} \cap A^{ai}_{v\overrightarrow{u}} & \text{for } v \overrightarrow{\succ} u, \\ = A^{bj}_{vu} \cap A^{i}_{v\overrightarrow{u}} & \text{for } v \overleftarrow{\succ} u, \end{cases}$$

respectively, where we discussed above that Lemma 16 holds as well for c-restricted integer tuples, and hence, using similar arguments as in the proof of this lemma, it is easy to see that the equations in the second parts can be satisfied. This inductive construction clearly defines b. The motivation behind taking the maximum is that this minimizes the 'rounding error'. However, to prove Lemma 12, we still need to show that there is a schedule σ satisfying b with $\mathrm{cost}(\sigma) \leq (1+\epsilon)\mathrm{OPT}$.

Consider some fixed vertex u and level i. Due to the inductive construction of b described above, we immediately obtain that, for each vertex $v \succ \hat{u}$,

$$A^{bi}_{vu} \subseteq A^{bi}_{v\hat{u}} \cap A^{ai}_{vu}. \tag{18}$$

Therefore, when comparing b to a, we can think of each interval $I \in (A^{bi}_{v\hat{u}} \cap A^{ai}_{vu}) \backslash A^{bj}_{vu}$ as an interval which is assigned by b to the *wrong area*. Specifically, for the optimal schedule σ^* satisfying a, we have that $\sigma^*(I) \in P_u$. However, for any schedule σ that satisfies b and assigns I, we have that $\sigma(I) \notin P_u$. That is why we say that I is assigned to the wrong area. The set of intervals which are assigned at node u to the wrong area is then

$$\bigcup_{v \in R: v \succ \hat{u}} (A^{bi}_{v\hat{u}} \cap A^{ai}_{vu}) \backslash A^{bi}_{vu},$$

and let $\overleftarrow{X}^i_u$ and $\overrightarrow{X}^i_u$ be a 2-partition of this set into two subsets, where

$$\overleftarrow{X}^i_u := \bigcup_{v \in R: v \overleftarrow{\succ} \hat{u}} (A^{bi}_{v\hat{u}} \cap A^{ai}_{vu}) \backslash A^{bi}_{vu} \text{ and } \overrightarrow{X}^i_u := \bigcup_{v \in R: v \overrightarrow{\succ} \hat{u}} (A^{bi}_{v\hat{u}} \cap A^{ai}_{vu}) \backslash A^{bi}_{vu}.$$

Finally, define

$$\overleftarrow{X}^i := \bigcup_{u \in R} \overleftarrow{X}^i_u, \overrightarrow{X}^i := \bigcup_{u \in R} \overrightarrow{X}^i_u, \text{ and } X := \bigcup_{i=1}^{m} (\overleftarrow{X}^i \cup \overrightarrow{X}^i).$$

We need the following lemma to prove Lemma 12, where we assume that $1/\epsilon$ is integral.

Lemma 13. *For any $\epsilon > 0$ and each level i, if we choose $c = 1/\epsilon$, then there is a schedule σ for the intervals $\overleftarrow{X}^i$ with the following properties:*

(1) σ assigns all intervals in $\overleftarrow{X}^i$ except ϵT many,

(2) σ satisfies A and b,

(3) for each period t, the number of intervals in $\overleftarrow{X}^i$ that σ assigns to t is at most the number of periods in $\overleftarrow{X}^i$ that σ^* assigns to t.

Moreover, the analogous statement holds for $\overrightarrow{X}^i$.

Proof of Lemma 12. Note that we can select σ such that it assigns all intervals which are assigned to the *correct area*, namely $J\backslash X$, exactly as σ^*. Specifically, σ assigns each interval $I \in J\backslash X$ to period $\sigma^*(I)$, i.e., $\sigma(I) = \sigma^*(I)$. So far we have that σ satisfies A and b with $\text{cost}(\sigma) \leq \text{OPT}$. Note that we even have that, for each period t and level i, the number of intervals in J_i which σ assigns to t is at most the number of intervals in J_i that σ^* assigns to t. Hence, if moreover $|X| \leq \epsilon n$, then σ assigns all intervals in J except an ϵ-fraction, and we are done. However, we cannot expect that $|X| \leq \epsilon n$. Consequently, we need to extend σ to intervals X by applying Lemma 13. Specifically, we apply this lemma seperately for each level i, and then $\overleftarrow{X}^i$ and $\overrightarrow{X}^i$. Doing this, we conclude that, for any $\epsilon > 0$, if we choose $c = 1/\epsilon$, then we can extend σ such that the following properties are satisfied:

(1) σ assigns all intervals in J except $\epsilon 2mT$ many,

(2) σ satisfies A and b,

(3) $\text{cost}(\sigma) \leq \text{OPT}$.

Using this, the claim follows with Lemma 8, since $2mT = \mathcal{O}(n)$ and $|J| = n$, but c may be chosen arbitrarily large, and hence ϵ may be chosen arbitrarily small. □

To prove Lemma 13, we first need to prove three structural lemmas about the interval sets $\overleftarrow{X}^i_u$. Assume that $c = 1/\epsilon$, and consider a fixed level i in what follows. Recall that a non-root vertex u is the left and right child of its parent $v = \hat{u}$ if $u = \overleftarrow{v}$ and $u = \overrightarrow{v}$, respectively.

Lemma 14. *For each vertex u which is a left child and vertex u' with $\hat{u} \overset{\rightarrow}{\succ} u'$, we have for each interval pair $I \in \overleftarrow{X}^i_u$ and $I' \in \overleftarrow{X}^i_{u'}$ that I can be assigned to period $\sigma^*(I')$, i.e., $\sigma^*(I') \in I$.*

Proof. Let $v \overset{\leftarrow}{\succ} \hat{u}$ be the vertex with $I \in A^i_{vu}$. First note that since I' is assigned to the wrong area at vertex u', we have that $\sigma^*(I') \in P_{u'}$. Therefore, since $\hat{u} \overset{\rightarrow}{\succ} u'$, we have that $\sigma^*(I') \geq \max P_u + 1$. On the other hand, $\hat{u} \overset{\rightarrow}{\succ} u'$ and $v \overset{\leftarrow}{\succ} \hat{u}$ imply that $v \overset{\leftarrow}{\succ} u'$. Consequently, we find that $\sigma^*(I') \leq a_v$. By combining these fact, we conclude that $\sigma^*(I') \in \{\max P_u + 1, \ldots, a_v\}$. On the other hand, since $I \in A^i_{vu}$, we have that $\{\max P_u + 1, \ldots, a_v\} \subset I$, which proves the claim. We depict this situation schematically in Figure 7. □

Lemma 15. *For each vertex u which is a left child, we have that $|\overleftarrow{X}^i_u| \leq \epsilon T/2^{d_u}$.*

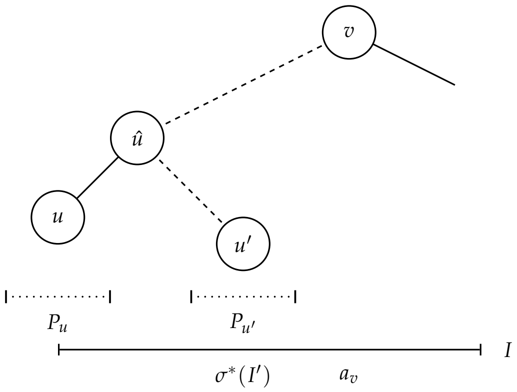


Figure 7: The vertices u, $\hat{u}$, u', v, and the interval I.

Proof. First, for $c = 1/\epsilon$, the maximality condition used during the inductive construction of b implies that, for each vertex $v \overleftarrow{\succ} \hat{u}$,

$$|(A^{bi}_{v\hat{u}} \cap A^{ai}_{vu}) \backslash A^{bi}_{vu}| \leq |A^i_{vu}|/c = \epsilon |A^i_{vu}|.$$

Using this, we obtain that

$$\begin{aligned} |\overleftarrow{X}^i_u| &\leq \epsilon \sum_{v \in R : v \overleftarrow{\succ} \hat{u}} |A^i_{vu}| \\ &\leq \epsilon T / 2^{d_u}, \end{aligned}$$

which proves the claim. The first line is due to the definition of $\overleftarrow{X}^i_u$ and Inequality (19). Finally, the second line is due to the facts that the sets A^i_{vu} are pairwise disjoint, we assume that all endpoints are distinct, and $|P_u| = T/2^{d_u}$. Note that we do not need that u is a left child, but if u is a right child, then the following lemma provides an even stronger statement . ☐

Lemma 16. *For each vertex u which is a right child, we have that $|\overleftarrow{X}^i_u| = 0$.*

Proof. If u is a right child, then we have that

$$\begin{aligned} \overleftarrow{X}^i_u &= \bigcup_{v \in R : v \overleftarrow{\succ} \hat{u}} (A^{bi}_{v\hat{u}} \cap A^{ai}_{vu}) \backslash (A^{bi}_{v\hat{u}} \cap A^i_{vu}) \\ &= \emptyset \end{aligned}$$

which proves the claim. The first equality is due to the definition of $\overleftarrow{X}^i_u$ and the equalities in the second parts of Constraints (17). The second line is due to the simple fact that $A^{ai}_{vu} \subseteq A^i_{vu}$. ☐

Lemmas 14, 15, and 16 allow us to use a relaxation. Specifically, for each vertex

u, consider an arbitrary set Z_u with

$$|Z_u| \leq \begin{cases} \epsilon T/2^{d_u} & \text{if } u \text{ is a left child,} \\ 0 & \text{if } u \text{ is a right child.} \end{cases}$$

For simplicity, we also refer to an element $I \in Z_u$ as an *interval.* Define

$$Z := \bigcup_{u \in R} Z_u.$$

Moreover, let f always denote an injective partial function $f : Z \to Z$ with the property that if $f(I) = I'$, then there is a pair of vertices u, u' with $\hat{u} \overrightarrow{\succ} u'$ such that $I \in Z_u$ and $I' \in Z_{u'}$. Moreover, let $|f| := |Z \setminus f^{\leftarrow}(Z)|$ denote the number of intervals in Z where f is not defined. By Lemmas 14, 15, and 16, if we can show that, for any configuration of the sets Z_u, we can find such a function f with $|f| \leq \epsilon T$, then Lemma 13 holds. To see this, observe that the configuration of the sets Z_u could for example be the sets X_u^i, and if $f(I) = I'$, then set $\sigma(I) := \sigma^*(I')$, which is possible because of Lemma 14. However, we also allow other configurations, which might not be realizable as X_u^i. The following lemma gives us the worst case configuration of the sets Z_u.

Lemma 17. *For each vertex u which is a left child, we may assume that $|Z_u| = \lfloor \epsilon T/2^{d_u} \rfloor$.*

Proof. Consider the worst case configuration of the sets Z_u, i.e., the configuration such that $|f|$ is maximal for the optimal function f, that is, the function that minimizes $|f|$. Then, assume that there is a vertex u which is a left child with $|Z_u| < \lfloor \epsilon n/2^{d_u} \rfloor$. Moreover, let f' be an optimal function after adding an additional interval I to Z_u. We will show that $|f| \leq |f'|$, which implies that $|Z_u| = \lfloor \epsilon T/2^{d_u} \rfloor$ is also a worst case. To this end, assume for contradiction that $|f| > |f'|$. Since all other cases work analogously, assume moreover that there are two vertices v and u' with two respective intervals $I' \in Z_v$ and $I'' \in Z_{u'}$ such that $f'(I') = I$ and $f'(I) = I''$, and hence $\hat{v} \overrightarrow{\succ} u$ and $\hat{u} \overrightarrow{\succ} u'$. Because this relation is clearly transitive, we find that also $\hat{v} \overrightarrow{\succ} u'$. Therefore, we can remove I, and then set $f'(I') := I''$. Since $|f'|$ does not change and $|f| > |f'|$, this contradict the optimality of f. The claim follows. $\square$

Proof of Lemma 13. By the arguments above and Lemma 17, we only have to show that if, for each vertex u,

$$|Z_u| = \begin{cases} \lfloor \epsilon T/2^{d_u} \rfloor & \text{for } u \text{ is a left child,} \\ 0 & \text{for } u \text{ is a right child,} \end{cases} \tag{19}$$

then there is a function f with $|f| \leq \epsilon T$. To this end, consider a fixed vertex u which is a left child. We need to define the function f on the set Z_u.

Assume that $1/\epsilon \leq T$. Otherwise, if $1/\epsilon > T$, then $c > T$, and hence $|X| = 0$, i.e., no intervals are assigned to the wrong area. Moreover, assume that $1/\epsilon$ is a power of 2, and let d be the positive integer such that $2^d/\epsilon = T$. Recall that we assume that also T is a power of 2. Therefore, for each vertex v with $d_v > d$, Equation (19) implies that $|Z_v| = 0$. Consequently, we may assume that $d_u \leq d$. Let $v = \hat{u}$, $u_0 := \overrightarrow{v}$, and $s := d - d_u$, and let $u_0, u_1, u_2, \ldots, u_s$ be the simple path from u_0 to a vertex u_s with $d_{u_s} = d$, where, for each $1 \leq j \leq s$, $u_{j+1} := \overleftarrow{u_j}$. This path is depicted in Figure 8. We have that

$$\begin{aligned}\sum_{j=1}^{s} |Z_{u_i}| &= \sum_{j=1}^{s} \lfloor \epsilon T/2^{d_{u_j}} \rfloor = \sum_{j=1}^{s} \epsilon T/2^{d_{u_j}} = \sum_{j=1}^{s} \epsilon T/2^{d_u+j} \\ &= \epsilon T/2^{d_u} - 1 \\ &= |Z_u| - 1.\end{aligned}$$

The first line is due to Equation (19), since all vertices $u_1, u_2, \ldots, u_s$ are left children, and the simple fact that all numbers in this sum are integral. Moreover, the second line is due to the standard transformation of the geometric series. Finally, the third line is again due to Equation (19). Therefore, we can define f on the intervals Z_u such that (1) $f(Z_u) \subseteq \cup_{j=1}^{s} Z_{u_j}$ and (2) f is defined on all intervals in Z_u except one. Property (1) is visualized as the dotted arrow in Figure 8. Moreover, this scheme can be extended to all vertices u that are left children with $1 \leq d_u \leq d$. We conclude with Property (2) that since, for each depth $1 \leq j \leq d$, there are 2^j many vertices u with $d_u = j$, and half of these vertices are left children, we can choose f such that

$$|f| = \frac{1}{2} \sum_{j=1}^{d} 2^j = 2^d - 1 < \epsilon T,$$

which completes the proof of the lemma. The same arguments work for $\overrightarrow{X}^i$ if we switch sides. $\square$

5 NP-hardness

In this section, we prove the following theorem.

Theorem 18. CB_k *is strongly NP-hard, even for* $k = 3$ *and two different interval*

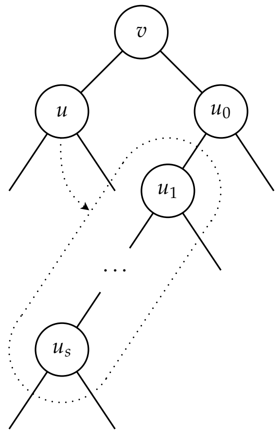


Figure 8: The path $u_1, u_2, \ldots, u_s$.

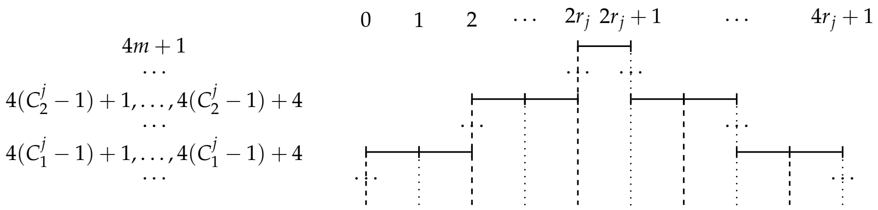


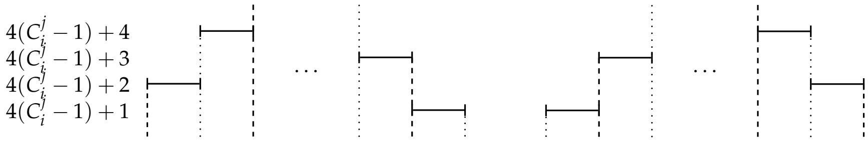
Figure 9: The instance S_j.

lengths.

Assume that $k = 3$, and let Q be a 3-SAT instance with m clauses labeled $1, 2, \ldots, m$ and variables $v_1, v_2, \ldots, v_n$. We will construct a CB$_k$-instance S with $4m + 1$ different weights from Q. Note that we use n here to denote the number of variables instead of the number of intervals.

Consider some variable v_j, and let $C_1^j > C_2^j > \ldots > C_{r_j}^j$ be the r_j clauses that contain this variable. We will construct an instance S_j from these clauses, which we will use later as a building block to construct S. The instance S_j is illustrated in Figure 9. All intervals in this instance have length 1. The vertical height of the intervals represent their weights, which are listed on the left. Hence, there is one interval $[2r_j, 2r_j + 1]$ with weight $4m + 1$, and for each clause C_i^j, there are four intervals distributed over the four weights $4(C_i^j - 1) + 1, \ldots, 4(C_i^j - 1) + 4$. This distribution depends on the clause C_i^j as illustrated in Figure 10. Specifically, if variable v_j appears not negated in C_i^j, then we distribute these intervals as illustrated in Subfigure 10(a), and otherwise, we distribute them as illustrated in Subfigure 10(b). Note that the maximum possible batch size in S_j is 2.

The dashed and dotted vertical lines in Figure 9 represent two schedules for S_j, say σ_0 and σ_1, respectively, where, for each clause C_i^j, the exact weights of the



(a) not negated (b) negated

Figure 10: The intervals corresponding to variable v_j and clause C_i^j.

batches in $4(C_i^j - 1) + 1, \ldots, 4(C_i^j - 1) + 4$ are illustrated in Figure 10. For the sake of exposition, we omit the usual dots that indicate which intervals are exactly stabed in Figures 9 and 10. Note that these dots are not necessary, since no batch exploits its full capacity. It is easy to compute that

$$\mathrm{cost}(\sigma_0), \mathrm{cost}(\sigma_1) = W_j, \text{ where } W_j := 4m + 1 + 5r_j + 8\sum_{i=1}^{r_j}(C_i^j - 1). \tag{20}$$

Moreover, in both schedules, all intervals in S_j except one of the two with lowest weights are *matched* by a batch, i.e., there is a batch that stabs exactly these two intervals, and the single interval not matched is *exclusively* stabbed by a batch, i.e., the batch that stabs this interval does not stab any other interval. We refer to this property as the *matching property.* Note that σ_0 and σ_1 are the only schedules with this property. We obtain the following simple lemma.

Lemma 19. *The two schedules σ_0 and σ_1 are the only optimal schedules for S_j.*

Proof. We already know from (20) that σ_0 and σ_1 have the same cost, and hence, if one them is optimal, then they are both.

Assume for contradiction that we have another optimal schedule σ without the matching property described above. Since we have an odd number of intervals in S_j, there must be an unmatched interval $[a, a + 1]$ which is not one of the two intervals with lowest weights. Hence, $[a, a + 1]$ is exclusively stabbed by a batch C. Assume w.l.o.g. that $a \geq 2r$ and that C stabs $[a, a + 1]$ at its right endpoint $a + 1$, i.e., $t(C) = a + 1$. Note that there is an interval $[a + 1, a + 2]$ which has a lower weight than $[a, a + 1]$, and this interval can therefore be stabbed by C as well. Now, if $[a + 1, a + 2]$ is not matched with some other interval, then we can decrease $\mathrm{cost}(\sigma)$ by removing the batch so far stabbing $[a + 1, a + 2]$. On the other hand, if $[a + 1, a + 2]$ is matched with another interval, then this interval is $[a + 2, a + 3]$ with an even lower weight than $[a + 1, a + 2]$. Consequently, we can also decrease $\mathrm{cost}(\sigma)$ by lowering the weight of the batch so far matching $[a + 1, a + 2]$ with $[a + 2, a + 3]$ to the weight of $[a + 2, a + 3]$. Therefore, in both

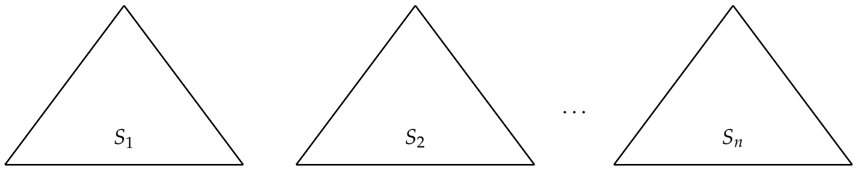


Figure 11: The instance S'.

cases, we obtain a contradiction to the assumption that σ is optimal. The claim of the lemma follows. $\square$

Before constructing S, we first construct an intermediate instance S'. To this end, for each variable v_j, we construct such an instance S_j, and then *merge* the instances $S_1, S_2, \ldots, S_n$ to one large instance S' such that the subinstances $S_1, S_2, \ldots S_n$ do not intersect by moving them appropriately to the right as depicted in Figure 11. Since the instances $S_1, S_2, \ldots, S_n$ are triangle-shaped, we represent them as a triangle in this figure. We already know from Lemma 19 how an optimal schedule for each of the instances $S_1, S_2, \ldots, S_n$ looks like. Hence, we also know how all optimal schedule for S' look like, these are basically all combinations of the two schedules σ_0 and σ_1, and that all these combinations have the same cost

$$W := \sum_{j=1}^{n} W_j.$$

Let $\mathcal{P}$ be the set of these combined schedules for S, and therefore $|\mathcal{P}| = 2^n$. Each schedule $\sigma \in \mathcal{P}$ *corresponds* to an assignment for Q as follows: v_j is true if the part of σ stabbing the intervals in S_j is σ_1, and otherwise, v_j is false if the part of σ stabbing the intervals in S_j is σ_0.

The instance S' consists only of *short intervals*, i.e., intervals of length 1. Let S be an instance which contains the same short intervals, but we additionally add several *long intervals*, i.e., intervals of the form $[0, \infty]$, where ∞ is simply a value which is larger than all right endpoints of the short intervals already contained in S'. We add long intervals to S as follows:

- n long intervals with weight $4m + 1$,
- for each clause i, one long interval with weight $4(i-1) + 4$ and five long intervals with weight $4(i-1) + 1$,
- for each variably v_j, one long interval with weight $4(i-1)+1$ for the minimum labeled clause i containing v_j, i.e., $i = \min\{1 \leq r \leq m \mid r \text{ contains } v_j\}$.

Note that the added long intervals *cross* all subinstances $S_1, S_2, \ldots, S_n$ of S. Theorem 18 follows from the construction of S and the following lemma.

Lemma 20. *The instance S has a schedule of cost W if and only if Q is satisfiable.*

Proof. Consider some fixed schedule $\sigma \in \mathcal{P}$ with $\mathrm{cost}(\sigma) = W$. Since each batch $C \in \sigma$ stabs at most two short intervals, C may stab one or two additional long intervals. In the latter case, there is some subinstance S_j such that C stabs one of the two short intervals with lowest weights in S_j, and hence, the weight of C is also either $4(i-1)+1$ or $4(i-1)+2$, where i is the minimum labeled clause containing variable v_j. The question is whether we can choose σ such that this additional stabbing capacity is sufficient to stab all long intervals in S, since in this case, we have that σ is also a schedule for S. We will show that this is possible if and only if σ corresponds to a satisfying assignment for Q, which proves the claim. To this end, we show via induction that, for any $0 \leq i \leq m$, all long intervals with at least weight $4(i-1)+5$ can be stabbed by batches in σ if and only if all clauses $m, m-1, \ldots, i+1$ are satisfied by the assignment corresponding to σ, and then these batches cannot stab any more long intervals at lower weights $1, 2, \ldots, 4(i-1)+4$.

Induction start. We start the induction by showing that none of the batches with weight $4(m-1)+5 = 4m+1$ may stab any more long intervals at lower weights. But this is easy to see, since each of these n batches needs to stab one of the n long intervals at this weight.

Induction step. Assume as induction hypothesis that the claim holds for some fixed $1 \leq i \leq m$. Moreover, assume first that, for each variable v_j contained in clause i, i is not the minimum labeled clause containing v_j.

Since Q is a 3-SAT instance, there are exactly 6 additional long intervals from Group (2) at weights $4(i-1)+1, \ldots, 4(i-1)+4$, and σ contains exactly 6 batches with these weights. Each of these batches may stab one additional long interval, and therefore, since the induction assumption implies that none of the batches in σ with starting weight at least $4(i-1)+5$ may stab another long interval, we have to *match* these 6 batches with the 6 long intervals, i.e., each batch needs to stab exactly one of the long intervals. Observe that we can match any of these batches with any long interval at weight $4(i-1)+1$. Hence, we only need that there is at least one batch with weight $4(i-1)+4$ to match the single long interval at this weight. Observe that, by the construction of S and σ, it is easy to see that this holds if and only if clause i is satisfied by the assignment corresponding to σ, and hence, the induction hypothesis holds for $i+1$ as well.

Finally, consider the case that, for some variable v_j contained in clause i, i is the minimum labeled clause containing v_j, and hence, there is a batch in σ whose weight is either $4(i-1)+1$ or $4(i-1)+2$ that may stab two additional long intervals instead of one. However, since we added for this case also one more long interval with weight $4(i-1)+1$, this additional capacity is needed to stab this long interval. This completes the proof of the induction step, and hence the claim of the lemma. $\square$

6 Conclusions

In this paper, we showed the NP-hardness of CB_k, even for $k = 3$, and presented a PTAS for any constant k. This PTAS is based on a novel dynamic programming approach that is likely to find application in related problems where we need to stab intervals subject to some sophisticated cost function. Specifically, to initialize the array Π, we could replace the function ω by any other *well-behaving* function. On the other hand, it is also possible to incorporate the penalization of gaps as in [1]. However, due to space limitations, we have to defer such a discussion to a full version of this paper. For the sake of simplicity, we did not analyze the exact running time of the PTAS.

To obtain the PTAS, we first showed that we may assume that the number of levels is constant, and then we gave a PTAS for this special case. However, it is not clear whether this special case is still NP-hard, and so there might be a polynomial time algorithm. Finding such an algorithm is an interesting open problem, although this would also just yield a PTAS for CB_k.

Finally, our PTAS requires that k is constant. On the other hand, a 2-approximation algorithm for arbitrary k is known [5], but it is an open problem whether this case admits a PTAS as well. We think that the methods developed in this paper are limited to constant k.

7 Acknowledgments

The author would like to thank Maxim Sviridenko and Nicole Megow for valuable discussions.

References

[1] Philippe Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 364–367, 2006.

[2] Luca Becchetti, Peter Korteweg, Alberto Marchetti-Spaccamela, Martin Skutella, Leen Stougie, and Andrea Vitaletti. Latency constrained aggregation in sensor networks. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA'06)*, pages 88–99, 2006.

[3] Hans L. Bodlaender and Klaus Jansen. Restrictions of graph partition problems. part i. *Theor. Comput. Sci.*, 148(1):93–109, 1995.

[4] Mourad Boudhar. Dynamic scheduling on a single batch processing machine with split compatibility graphs. *J. Math. Model. Algorithms*, 2(1):17–35, 2003.

[5] J. Correa, Nicole Megow, R. Raman, and Karol Suchan. Cardinality constrained graph partitioning into cliques with submodular costs. In *Proceedings of the 8th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW'09)*, pages 347–350, 2009.

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

[7] Gerd Finke, Vincent Jost, Maurice Queyranne, and András Sebö. Batch processing with interval graph compatibilities between tasks. *Discrete Applied Mathematics*, 156(5):556–568, 2008.

[8] Dion Gijswijt, Vincent Jost, and Maurice Queyranne. Clique partitioning of interval graphs with submodular costs on the cliques. *RAIRO - Operations Research*, 41(3):275–287, 2007.

[9] Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32(1):130–136, 1985.

[10] M.R.Garey and D.S. Johnson. *Computers and Intracability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[11] Sriram V. Pemmaraju and Rajiv Raman. Approximation algorithms for the max-coloring problem. In *Proceedings of 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, pages 1064–1075, 2005.

[12] Sriram V. Pemmaraju, Rajiv Raman, and Kasturi Varadarajan. Buffer minimization using max-coloring. In *Proceedings of the 15th annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 562–571, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.