Scheduling Bidirectional Traffic on a Path

Yann Disser, Max Klimm, and Elisabeth Lübbecke

Department of Mathematics, Technische Universität Berlin
{disser,klimm,eluebbecke}@math.tu-berlin.de

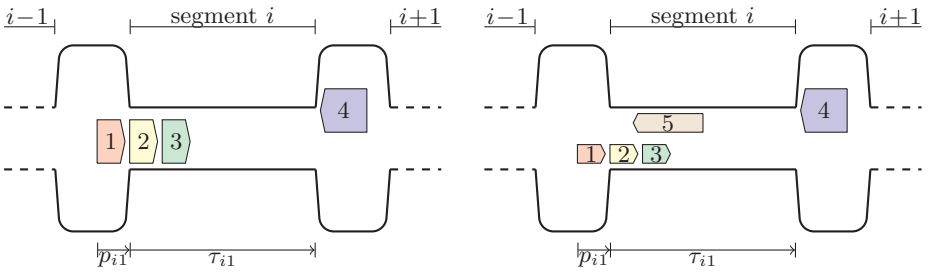
Abstract. We study the fundamental problem of scheduling bidirectional traffic along a path composed of multiple segments. The main feature of the problem is that jobs traveling in the same direction can be scheduled in quick succession on a segment, while jobs in opposing directions cannot cross a segment at the same time. We show that this tradeoff makes the problem significantly harder than the related flow shop problem, by proving that it is NP-hard even for identical jobs. We complement this result with a PTAS for a single segment and non-identical jobs. If we allow some pairs of jobs traveling in different directions are allowed to cross a segment concurrently, the problem becomes APX-hard even on a single segment and with identical jobs. We give polynomial algorithms for the setting with restricted compatibilities between jobs on a single and any constant number of segments, respectively.

1 Introduction

The scheduling of bidirectional traffic on a path is essential when operating single-track infrastructures such as single-track railway lines, canals, or communication channels. Roughly speaking, the schedule governs when to move jobs from one node of the path to another along the segments of the path. The goal is to schedule all jobs such that the sum of their arrival times at their respective destinations is minimized. A central feature of real-world single-track infrastructures is that after one job enters a segment of the path, further jobs moving in the *same* direction can do so with relatively little headway, while traffic in the *opposite* direction usually has to wait until the whole segment is empty again (cf. Figure 1a for a schematic illustration).

Formally, in the bidirectional scheduling problem we are given a path of consecutive segments connected at nodes, and a set of jobs, each with a release date and a designated start and destination node. The time job j needs to traverse segment i is governed by two quantities: its *processing time* p_{ij} and its *transit time* τ_{ij}. While the former prevents the segment from being used by any other job (running in *either* direction), the latter only blocks the segment from being used by jobs running in *opposite* direction. For example, this allows us to model settings with bidirectional train traffic on a railway line split into single-track segments that are connected by turnouts (cf. Lusby et al. [16, Section 2]). In this setting, jobs correspond to trains, the processing time of a job is the time needed for the train to fully enter the next segment, and the transit time is the time to traverse the segment (and entirely move into the next turnout). While a train is entering a single-track segment of the line, no other train may do so. The next train in the same direction can enter immediately afterwards, whereas trains in opposite direction have to wait until the segment is clear again in order to prevent a collision.

Figure 2 shows the path-time-diagram of a feasible schedule for two segments and four jobs. Jobs are represented by parallelograms of the same color. The



(a) without compatibilities (b) with compatibilities

Fig. 1: Bidirectional scheduling of ship traffic through a canal, with and without compatibilities. The processing time p_{ij} of job j is the time needed to enter segment i with sufficient security headway, i.e., the delay before other jobs in the same direction may enter the segment. The travel time τ_{ij} is the time needed to traverse the entire segment once entered. In both (a) und (b), jobs $1, 2, 3$ can enter the segment in quick succession, while job 4 has to wait until they left the segment. In (b), job 5 is compatible with jobs $1, 2, 3$ so that they may cross concurrently. The time to cross turnouts is assumed to be negligible.

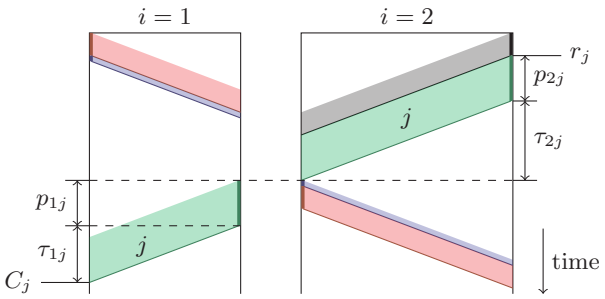


Fig. 2: Representation of a schedule on two segments ($i = 1, 2$) and four jobs as a path-time-diagram. In this example, all jobs are processed immediately at their release date. Job j is released at time r_j at the right end of segment 2 and needs to reach the left end of segment 1. Since it never has to wait, its completion time is smallest possible: $C_j = r_j + p_{2j} + \tau_{2j} + p_{1j} + \tau_{1j}$.

processing time of a job on a segment is reflected by the height of the corresponding parallelogram, while the transit time is the remaining time (y-distance) to the lowest point of the parallelogram. In a feasible schedule, jobs may not intersect, and, in particular, a job can only begin being processed at a segment once it has fully exited the previous segment. Note that in the example it makes sense for the two rightbound jobs to switch order while waiting at the central node.

We also study a generalization of the model to situations where some of the jobs are allowed to pass each other when traveling in different directions (cf. Figure 1b). This is a natural assumption, e.g., when scheduling the ship traffic on a canal, where smaller ships are allowed to pass each other while larger ships are not (cf. Lübbecke et al. [15]). In practice, the rules that decide which ships are allowed to pass each other are quite complex and depend on multiple parameters of the ships such as length, width, and draught (e.g., cf. [5]). We model these complex rules in the most general way by a bipartite compatibility graph for each segment, where vertices correspond to jobs and two jobs running in different directions are connected by an edge if they can cross the segment concurrently.

Table 1: Overview of our results for bidirectional scheduling.
[1] even if $p = 0$, $\tau_i = 1$, [2] only if $p = 1, \tau_i \leq$ const, [3] even if $\tau_i = p = 1$.

	Number m of segments		
compatibilities	$m = 1$	m const.	m arbitrary
Different jobs $p_{ij} = p_j$, $\tau_{ij} = \tau_i$			
none/all compatible	PTAS [Thm. 2]		NP-hard[1] [Thm. 1]
	NP-hard [14]		
Identical jobs $p_{ij} = p$, $\tau_{ij} = \tau_i$			
none compatible	polynomial [Thm. 5]	polynomial[2] [Thm. 6]	NP-hard[1] [Thm. 1]
const. # types			
arbitrary	APX-hard[3] [Thm. 4]		

Our results. Table 1 gives a summary of our results. We first show that scheduling bidirectional traffic is hard, even without processing times and with identical transit times (Section 3). The proof is via a non-standard reduction from MaxCut. The key challenge is to use the local interaction of the jobs on the path to model global interaction between the vertices in the MaxCut. We overcome this issue by introducing polynomially many vertex gadgets encoding the partition of each vertex and synchronizing these copies along the instance. We complement this result with a polynomial time approximation scheme (PTAS) for a single segment and arbitrary processing times (Section 4) using the $(1 + \epsilon)$-rounding technique of Afrati et al. [1].

We then show that bidirectional scheduling with arbitrary compatibility graphs is APX-hard already on a single segment and with identical processing times (Section 5). The proof is via a reduction from a variant of Max-3-Sat which is NP-hard to approximate within a factor smaller than 1016/1015, as shown by Berman et al. [3]. As a byproduct, we obtain that also minimizing the makespan is APX-hard in this setting. We again complement our hardness result by polynomial algorithms for identical jobs on constant numbers of segments and with a constant number of compatibility types (Section 6).

Significance. With this paper we initiate the mathematical study of optimized dispatching of traffic in networks with bidirectional edges, e.g. train networks, ship canals, communication channels, etc. In all of these settings, traffic in one direction limits the possible throughput in the other direction. While in the past decades a wealth of results has been established for the unidirectional case (i.e., classical scheduling, and, in particular, flow show models), surprisingly, and despite their practical importance, bidirectional infrastructures have not received a similar attention so far.

The bidirectional scheduling model that we propose captures the essence of bidirectional traffic by distinguishing processing and transit times. This simple framework already allows to exhibit the computational key challenges of this setting. In particular, we show that bidirectional scheduling is already hard for identical jobs on a path, which is in contrast to the unidirectional case. We observe another increase in complexity when allowing specific types of traffic to use an edge concurrently in both directions. In practice, this is reasonable e.g. for ship traffic in a canal, where small vessels may pass each other. In that sense, we show that scheduling ship traffic is already hard on a single edge and, thus, considerably harder than scheduling train traffic.

While bidirectional scheduling is hard in general, we show that certain features of real-world scenarios can make the problem tractable, e.g., a small number of turnouts along a single path and/or a small number of different vessels. In this work we restrict ourselves to simple paths, but we hope that our results are a first step towards understanding traffic in general bidirectional networks.

Related work. Scheduling problems are a fundamental class of optimization problems with a multitude of known hardness and approximation results (cf. Lawler et al. [12] for a survey). To the best of our knowledge, the bidirectional scheduling model that we propose and study in this paper has not been considered in the past nor is it contained as a special case in any other scheduling model. We give an overview of known results for related models.

For a single segment and jobs traveling from left to right, bidirectional scheduling reduces to the classical single machine scheduling problem, which Lenstra et al. [14] showed to be hard when minimizing total completion time. Afrati et al. [1] gave a PTAS with generalizations to multiple identical or a constant number of unrelated machines. Chekuri and Khanna [6] further generalized the result to related machines. We give a different generalization for bidirectional scheduling. For unrelated machines Hoogeveen et al. [10] showed that the completion time cannot be approximated efficiently within arbitrary precision, unless $\mathsf{P} = \mathsf{NP}$.

Bidirectional scheduling also has similarities to scheduling of two job families with a setup time that is required between jobs of different families. The general comments in Potts and Kovalyov [18] on dynamic programs for such kinds of problems apply in part to our technique for Theorem 5.

When all jobs need to be processed on all segments in the same order and all transit times are zero, bidirectional scheduling reduces to flow shop scheduling. Garey et al. [9] showed that it is NP-hard to minimize the sum of completion times in flow shop scheduling, even when there are only two machines and no release dates. They showed the same result for minimizing the makespan on three machines. Hoogeveen et al. [10] showed that there is no PTAS for flow shop scheduling without release dates, unless $\mathsf{P} = \mathsf{NP}$. In contrast, Brucker et al. [4] showed that flow shop problems with unit processing times can be solved efficiently, even when all jobs require a setup on the machines that can be performed by a single server only.

Job shop scheduling is a generalization of flow shop scheduling that allows jobs to require processing by the machines in any (not necessarily linear) order, cf. Lawler et al. [12, Section 14] for a survey. In this setting, the minimization of the sum of completion times was proven to even be $\mathsf{MAX\text{-}SNP}$-hard by Hoogeveen et al. [10]. Queyranne and Sviridenko [19] gave a $\mathcal{O}((\log(m\mu)/\log\log(m\mu))^2)$-approximation for the weighted case with release dates, where μ denotes the maximum number of operations per job.

Fishkin et al. [7] gave a PTAS for a constant number of a machines and operations per job. It is worth noting that job shop scheduling does not contain bidirectional scheduling as a special case, since it does not incorporate the distinction between processing and transit times for jobs passing a machine in different directions.

Job shop scheduling problems with unit jobs are strongly related to packet routing problems where general graphs are considered, see the discussion in seminal paper by Leighton et al. [13]. They proved that the makespan of any packet routing problem is linear in two trivial lower bounds, called the congestion and the dilation. For more recent progress in this direction, see, e.g., Scheideler [20] and Peis and Wiese [17]. All these works, however, consider minimizing the makespan and assume that the orientation of the graph is fixed. Antoniadis et al. [2] also consider average

flow time on a directed line. They give lower bounds for competitive ratios in the online setting and $\mathcal{O}(1)$ competitive algorithms with resource augmentation for the maximum flow time.

2 Preliminaries

In the bidirectional scheduling problem, we are given a set $M = \{1, \dots, m\}$ of segments which we imagine to be ordered from left to right. Further, we are given two disjoint sets of J^{r} and J^{l} of *rightbound* and *leftbound* jobs, respectively, with $J = J^{\mathrm{r}} \cup J^{\mathrm{l}}$ and $n = |J|$. Each job is associated with a *release date* $r_j \in \mathbb{N}$, a *start segment* s_j and a *target segment* t_j, where $s_j \leq t_j$ for rightbound jobs and $s_j \geq t_j$ for leftbound jobs. A rightbound job j needs to cross the segments $s_j, s_j+1, \dots, t_j-1, t_j$, and a leftbound job needs to cross the segments $s_j, s_j-1, \dots, t_j+1, t_j$. We denote by M_j the set of segments that job j needs to cross. Each job j is associated with a processing time $p_j \in \mathbb{N}$ and each segment i is associated with a transit time $\tau_i \in \mathbb{N}$. Note that we restrict ourselves to identical processing times for a single job and identical transit times for a single segment. We call $p_j + \tau_i$ the *running time* of job j on segment i.

A *schedule* is defined by fixing the start times S_{ij} for each job j on each segment $i \in M_j$. The *completion time* of job j on segment i is then defined as $C_{ij} = S_{ij} + p_j + \tau_i$. The overall completion time of job j is $C_j = C_{t_j j}$. A schedule is feasible if it has the following properties.

1. Release dates are respected, i.e., $r_j \leq S_{s_j j}$ for each $j \in J$.
2. Jobs travel towards their destination, i.e., $C_{ij} \leq S_{i+1,j}$ (resp. $C_{ij} \leq S_{i-1,j}$) for rightbound (resp. leftbound) jobs j and $i \in M_j \setminus \{t_j\}$.
3. Jobs j, j' traveling in the same direction are not processed on segment $i \in M_j \cap M_{j'}$ concurrently, i.e., $[S_{ij}, S_{ij} + p_j) \cap [S_{ij'}, S_{ij'} + p_{j'}) = \emptyset$.
4. Jobs j, j' traveling in different directions are neither processed nor in transit on segment $i \in M_j \cap M_{j'}$ concurrently, i.e., $[S_{ij}, C_{ij}) \cap [S_{ij'}, C_{ij'}) = \emptyset$.

We consider the objective of minimizing the *total completion time* $\sum C_j = \sum_{j \in J} C_j$.

Other natural objectives are the minimization of the *makespan* $C_{\max} = \max\{C_j \mid j \in J\}$ or the *total waiting time* $\sum W_j = \sum_{j \in J} W_j$ where the individual waiting time of a job j is defined as $W_j = C_j - \sum_{i \in M_j} (p_j + \tau_i) - r_j$. Note that minimizing the total waiting time is equivalent to minimizing the total completion time.

We also consider a generalization of the model, where some of the jobs traveling in different directions are allowed to pass each other. Formally, for each segment i, we are given a bipartite *compatibility graph* $G_i = (J^{\mathrm{r}} \cup J^{\mathrm{l}}, E_i)$ with $E_i \subseteq J^{\mathrm{r}} \times J^{\mathrm{l}}$. Two jobs j, j' that are connected by an edge in G_i are allowed to run on segment i concurrently, i.e., condition 4 above need not be satisfied. Specifically, jobs j, j' may be processed or be in transit simultaneously.

All proofs omitted in the following sections can be found in the appendix.

3 Hardness of bidirectional scheduling

In this section we show that scheduling bidirectional traffic is hard, even when all processing times are zero and all transit times coincide. In other words, we eliminate all interaction between jobs in the same direction and show that hardness is merely due to the decision when to switch between left- and rightbound operation of each segment. This is in contrast to one-directional (flow shop) scheduling with identical processing times, which is trivial. Formally, we show the following result.

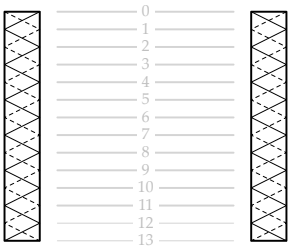


Fig. 3: Illustration of the vertex gadget in the leftbound (left) and the rightbound (right) state. At each time $t = 0, \ldots, 11$ multiple right- and leftbound jobs are released. Since all jobs have processing time 0, jobs in the same direction can be processed simultaneously. The only two sensible schedules differ in whether leftbound jobs are processed at even or odd times.

Theorem 1. *The bidirectional scheduling problem is* NP*-hard even if* $p_j = 0$ *and* $\tau_i = 1$ *for each* $j \in J$ *and* $i \in M$*.*

We reduce from the MaxCut problem which is contained in Karp's list of 21 NP-complete problems [11].

MaxCut
Input: An undirected graph $G = (V, E)$ and $k \in \mathbb{N}$.
Problem: Is there a partition $V = V_1 \dot\cup V_2$ with $|E \cap (V_1 \times V_2)| \geq k$?

For a given instance $\mathcal{I}$ of MaxCut we construct an instance of the bidirectional scheduling problem which can be scheduled without exceeding some specific waiting time if and only if $\mathcal{I}$ admits a solution. The translation to sum of completion times is then straightforward. We give an intuitive overview of our construction and defer all details to Appendix A.

A cornerstone of our construction is the *vertex gadget* that occupies a fixed time interval on a single segment and can only be (sensibly) scheduled in two ways (cf. Figure 4), which we interpret as the choice whether to put the corresponding vertex in the first or second part of the partition, respectively. We introduce multiple *vertex segments* that each have exactly one vertex gadget for each vertex in $\mathcal{I}$ and add further gadgets that ensure that the state of all vertex gadgets for the same vertex is the same across all segments. These gadgets allow us to synchronize vertex gadgets on consecutive vertex segments in two ways. We can either simply synchronize vertex gadgets that occupy the same time interval on the two vertex segments (*copy gadget*), or we can synchronize pairs of vertex gadgets occupying the same consecutive time intervals on the two vertex segments by linking the first gadget on the first segment with the second one on the second segment and vice-versa, i.e., we can transpose the order of two consecutive gadgets from one vertex segment to the next (*transposition gadget*).

We construct an edge gadget for each edge in $\mathcal{I}$ that incurs a small waiting time if two vertex gadgets in consecutive time intervals and segments are in different states and a slightly higher waiting time if they are in the same state. By tuning the multiplicity of each job, we can ensure that only schedules make sense where vertex gadgets are scheduled consistently. Minimizing the waiting time then corresponds to maximizing the number of edge gadgets that link vertex gadgets in different states, i.e., maximizing the size of a cut.

In order to fully encode the given MaxCut instance $\mathcal{I}$, we need to introduce an edge gadget for each edge in $\mathcal{I}$. However, edge gadgets can only link vertex

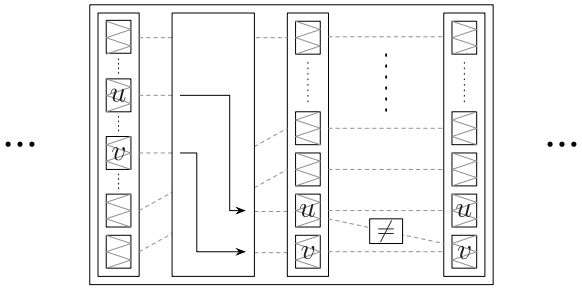


Fig. 4: Illustration of our hardness construction for a single edge $e = \{u, v\}$. First, a sequence of segments is used to change the order of vertex gadgets, such that the vertex gadgets corresponding to u and v occupy consecutive time intervals. Then, an edge gadget is added that incurs an increased waiting time if the vertex gadgets for u and v are in the same state.

gadgets in consecutive time intervals. We can overcome this limitation by adding a sequence of vertex segments and transposing the order of two vertex gadgets from one segment to the next as described before. With a linear number of vertex segments we can reach an order where the two vertex gadgets we would like to connect with an edge gadget are adjacent. At that point, we can add the edge gadget, and then repeat the process for all other edges in $\mathcal{I}$ (cf. Figure 4).

We can reformulate Theorem 1 for nonzero processing times, simply by making the transit time large enough that the processing time does not matter.

Corollary 1. *The bidirectional scheduling problem is* NP*-hard even if* $p_j = 1$ *and* $\tau_i = \tau$ *for each* $j \in J$ *and* $i \in M$*.*

4 A PTAS for bidirectional scheduling

In this section, we give a polynomial time approximation scheme (PTAS) for bidirectional scheduling on a single segment with general processing times. Lenstra et al. [14] showed this problem to be hard even if all jobs have the same direction. Afrati et al. [1] gave a PTAS, i.e., a polynomial $(1+\varepsilon)$-approximation algorithm for each $\varepsilon > 0$. Based on the same technique, we extend their result to the bidirectional case, provided that the jobs are either all pairwise in conflict or pairwise compatible. The main issue when trying to adopt the technique of [1] is to account for the different roles of processing and transit times for the interaction of jobs in the same and different directions.

Theorem 2. *The bidirectional scheduling problem on a single segment and with compatibility graph* $G_1 \in \{K_{n_r,n_l}, \emptyset\}$ *admits a PTAS.*

The first part of the proof in [1] is to restrict to processing times and release dates of the form $(1+\varepsilon)^x$ for some $x \in \mathbb{N}$ and $r_j \geq \varepsilon(p_j + \tau_1)$. Allowing fractional processing and release times we can show that any instance can be adapted to have these properties, without making the resulting schedule worse by a factor of more than $(1+\varepsilon)$. We may thus partition the time horizon into intervals $I_x = [(1+\varepsilon)^x, (1+\varepsilon)^{x+1}]$, such that every job is released at the beginning of an interval. Since jobs are not released too early, we may conclude that the maximum number of intervals σ covered by the running time of a single job is constant. This allows us

to group intervals together in blocks $B_t = \{I_{t\sigma}, I_{t\sigma+1}, \ldots, I_{(t+1)\sigma-1}\}$ of σ intervals each, such that every job scheduled to start in block B_t will terminate before the end of the next block B_{t+1}.

To use the fact that each block only interacts with the next block in our dynamic program, we need to specify an interface for this interaction. For that purpose we introduce the notion of a *frontier*. A block *respects an incoming frontier* $F = (f_\mathrm{l}, f_\mathrm{r})$ if no leftbound (rightbound) job scheduled to start in the block starts earlier than f_l (f_r). Similarly, a block *respects an outgoing frontier* $F = (f_\mathrm{l}, f_\mathrm{r})$ if no leftbound or rightbound job scheduled to start in the block would interfere with a leftbound (rightbound) job starting at time f_l (f_r). The symmetrical structure of the compatibility graph ($K_{n_\mathrm{r},n_\mathrm{l}}$ or $\emptyset$) allows us to use this simple interface. We introduce a dynamic programming table with entries $T[t, F, U]$ that are designed to hold the minimum total completion time of scheduling all jobs in $U \subseteq J$ to start in block B_t or earlier, such that B_t respects the outgoing frontier F. We define $C(t, F_1, F_2, V)$ to be the minimum total completion time of scheduling all jobs in V to start in B_t with B_t respecting the incoming frontier F_1 and the outgoing frontier F_2 (and ∞ if this is impossible). We have the following recursive formula for the dynamic programming table:

$$T[t, F, U] = \min_{F', V \subseteq U} \{T[t-1, F', U \setminus V] + C(t, F', F, V)\}.$$

To turn this into an efficient dynamic program, we need to limit the dependencies of each entry and show that $C(\cdot)$ can be computed efficiently. The number of blocks to be considered can be polynomially bounded by $\log D$, where $D = \max_j r_j + n \cdot (\max_j p_j + \tau_1)$ is an upper bound on the makespan. The following lemma shows that we only need to consider polynomially many other entries to compute $T[t, F, U]$ and we only need to evaluate $C(\cdot)$ for job sets of constant size, which we can do in polynomial time by simple enumeration.

Lemma 1. *There is a schedule with a sum of completion times within a factor of* $(1+\varepsilon)$ *of the optimum and with the following properties:*

1. *The number of jobs scheduled in each block is bounded by a constant.*
2. *Every two consecutive blocks respect one of constantly many frontiers.*

Proof (sketch). Partitioning the released jobs of each interval direction-wise by processing time into *small* and *large* jobs and bundling small jobs into packages of roughly the same size allows us to bound the number of released jobs per interval by a constant, similarly as in [1]. Furthermore, we establish that we may assume jobs to remain unscheduled only for constantly many blocks.

For the second property, we stretch all time intervals by a factor of $(1+\varepsilon)$, which gives enough room to decrease the start times of those jobs interfering with two blocks such that an $1/\varepsilon^2$-fraction of an interval separates jobs starting in two consecutive blocks. Thus, we only need to consider $\frac{\sigma}{\varepsilon^2}$ possible frontier values per direction, or a total of $\left(\frac{\sigma}{\varepsilon^2}\right)^2$ possible frontiers. □

5 Hardness of custom compatibilities

In Section 3, we showed that bidirectional scheduling is hard on an unbounded number of machines, even for identical jobs. As the main result of this section, we show that for arbitrary compatibility graphs the problem is APX-hard already on a single segment and with unit processing and transit times. For ease of exposition,

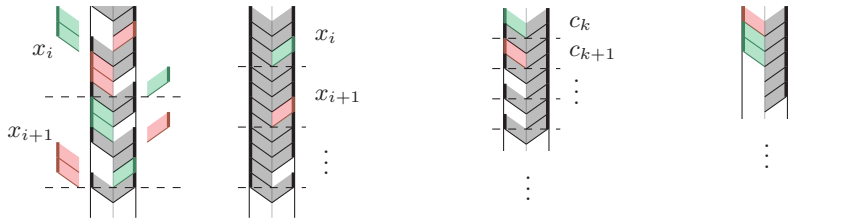


Fig. 5: Illustration (colored) of the four parts of our construction. Time is directed downwards, rightbound (leftbound) jobs are depicted on the left (right) of each figure.

we first show that the minimization of the makespan is NP-hard. Later we extend thus result towards minimum completion time and APX-hardness.

Theorem 3. *The bidirectional scheduling problem on a single segment and with an arbitrary compatibility graph is* NP*-hard even if* $p_j = \tau_1 = 1$ *for each* $j \in J$.

We give a reduction from an NP-hard variant of Sat (cf. [8]). Note the difference to the polynomially solvable $(3,3)$-Sat, where each variable appears in *exactly* three clauses [21].

$(\leq 3, 3)$-Sat
Input: A formula with a set of clauses C of size three over a set of variables X, where each variable appears in at most three clauses.
Problem: Is there a truth assignment of X satisfying C?

For a given $(\leq 3, 3)$-Sat formula we construct a bidirectional scheduling instance that can be scheduled within some specific makespan T if and only if the given formula is satisfiable. Our construction is best explained by partitioning the time horizon $[0, T]$ into four parts (cf. Figure 5 along with the following).

We use a frame of blocking jobs that need to be scheduled at their release date. We can enforce this by making sure that at least one blocking job is released at (almost) each unit time step and that blocking jobs that are not supposed to run concurrently are incompatible. We release variable jobs that have to be scheduled into gaps between the blocking jobs. More precisely, in the first part of the construction we release 6 jobs within a separate time interval for each variable. Two of these jobs are leftbound and need to be scheduled within the first two parts of the construction, which implies that one of the two remaining pairs of rightbound jobs must be scheduled after the second part. If the first pair is delayed we interpret this as an assignment of *true* to the variable and otherwise as *false*.

The third part of the construction has a gap for each clause, with compatibilities ensuring that only variable jobs can be scheduled into the gap which satisfy the clause. Since each literal can only appear in at most two clauses, there are enough variable jobs to satisfy all clauses if the formula is satisfied. Finally, the last part has $2|X| - |C|$ gaps that fit any variable job. In order to schedule all variable jobs before the end of the last part, we thus need to schedule a variable job into each gap of a clause. This is possible if and only if the given $(\leq 3, 3)$-Sat formula is satisfiable. We can easily extend our result to completion or waiting times by adding many blocking jobs after the last part, such that violating the makespan also ruins the the total completion time.

With a slight adaption of the construction and more involved arguments, we can even show APX-hardness of the problem. We reduce from a specific variant of Max-3-Sat, where each literal occurs exactly twice, and which is NP-hard to approximate within a factor of 1016/1015, see Berman et al. [3].

Theorem 4. *The bidirectional scheduling problem on a single segment and with an arbitrary compatibility graph is* APX*-hard even if* $p_j = \tau_1 = 1$ *for each* $j \in J$.

6 Dynamic programs for restricted compatibilities

After establishing the hardness of bidirectional scheduling with a general compatibility graph in the last section, in this section we turn to the case of a constant number of different compatibility types. We first show that the problem is easy for a single segment, and then expand our result to any fixed number of segments. Due to the identical processing times, the jobs in each direction can simply be scheduled in the order of their release dates. The only decision left is when to switch between left- and rightbound operation of the segments. This decision is hard in the general case (Theorem 1), but we are able to formulate a dynamic program for any constant number of segments.

Our result generalizes to the case when some jobs of different directions are compatible (i.e., may pass each other), as long as the number of *compatibility types* is constant, where two jobs j_1, j_2 in the same direction are defined to have the same compatibility type if the set of jobs compatible with j_1 is equal to the set of jobs compatible with j_2 on each segment. Formally, j_1 and j_2 have the same compatibility type if $\{j : \{j_1, j\} \in E_i\} = \{j : \{j_2, j\} \in E_i\}$ for the compatibility graphs $G_i = (J^{\mathrm{l}} \dot\cup J^{\mathrm{r}}, E_i)$ of each segment i.

We partition J into κ subsets of jobs $J^1, \dots, J^\kappa$ where all jobs of J^c, $c \in 1, \dots, k$, have the same compatibility type c, and let $n_c = |J^c|$. Since the jobs of each subset only differ in their release dates, they can again be scheduled in the order of their release dates. This allows us to expand the dynamic program to encompass any constant number of compatibility types. We obtain the following result for a single segment.

Theorem 5. *The bidirectional scheduling problem can be solved in polynomial time if* $m = 1$, κ *is constant and* $p_j = p$ *for each* $j \in J$.

Proof. We consider each subset J^c ordered non-increasingly by release dates and denote by J^c_i the i-th job of J^c in this order, i.e., the $(n_c - i)$-th job to be released. Each entry $T[i_1, t_1, \dots, i_\kappa, t_\kappa; c]$ of our dynamic programming table is designed to hold the minimum sum of completion times that can be achieved when scheduling only the $i_{c'}$ jobs of largest release date of each compatibility type c', such that $J^{c'}_{i_{c'}}$ is not scheduled before time $t_{c'}$ and $J^c_{i_c}$ is the first job that is scheduled. We start by setting $T[0, t_1, \dots, 0, t_\kappa; c] = 0$ and define the dependencies between table entries in the following.

Let $C(j, t) = \max\{t, r_j\} + p + \tau_1$ denote the smallest possible completion time of job j when scheduling it not before t. Depending on the types of jobs j_1, j_2 (and in particular of their directions), we can compute in constant time the earliest time $\theta(j_1, t_1, j_2, t_2)$ not before t_1 that job j_1 can be scheduled at, assuming that j_2 is scheduled earlier at time $\max\{t_2, r_{j_2}\}$. We let $\delta_{cc'} = 1$ if $c = c'$ and $\delta_{cc'} = 0$ otherwise, abbreviate $\theta_{c'} = \theta(J^{c'}_{i_{c'}}, t_{c'}, J^c_{i_c}, t_c)$, and get the following recursive formula for $i_c > 0$:

$$T[i_1, t_1, \dots, i_\kappa, t_\kappa; c] = \min_{c' : i_{c'} \neq 0} \{T[i_1 - \delta_{1c}, \theta_1, \dots, i_\kappa - \delta_{\kappa c}, \theta_\kappa; c'] + C(J^c_{i_c}, t_c)\}.$$

We can fill out our table in order of increasing sums $\sum i_c$ and finally obtain the desired minimum completion time as $\min_c T[n_1, 0, \ldots, n_\kappa, 0; c]$. We can reconstruct the schedule from the dynamic programming table in straightforward manner. It remains to argue that we only need to consider polynomially many times t_c. This is true, since all relevant times are contained in the set $\{r_j + k\tau + \ell p \mid j, k, \ell \leq n\}$ of cardinality $\mathcal{O}(n^3)$. □

We now consider a constant number of segments $m > 1$. The main complication in this setting is that decisions on one segment can influence decisions on other segments, and, in general, every job can influence every other job in this way. In particular, we need to keep track of how many jobs of each type are in transit at each segment, and we can thus not easily adapt the dynamic program for a single segment. We propose a different dynamic program that relies on all transit times being bounded by a constant.

Theorem 6. *The bidirectional scheduling problem can be solved in polynomial time if m, κ, and τ_i are constant for each $i \in M$, and $p_j = 1$ for each $j \in J$.*

Proof. Again, we consider subsets of identical jobs. In addition to their conflict type c, we further distinguish jobs by their start and target segments s, t and form subsets $J^c_{s,t}$ correspondingly. The number of subsets is bounded by κm^2. Since all release times are integer and since $p_j = 1$, we only need to consider integer points in time. Hence, only $\tau_i + 1$ possible positions need to be considered for a job running on segment i, and no two jobs of the same direction can occupy the same position. The state of the system can be fully described by (i) the number of available jobs per segment and $J^c_{s,t}$, and (ii) for each position on each segment and each $J^c_{s,t}$, the fact whether a job of $J^c_{s,t}$ is occupying this position. The number of states is bounded by $\prod_{i=1}^{m} n^{\kappa m^2} \cdot \prod_{i=1}^{m} 2^{\kappa m^2 (\tau_i + 1)} = \text{poly}(n)$.

We define the successors of each state to be all states that can be reached in one time step where not all jobs wait, or by waiting for the next release date. This way, the state representation changes from one state to the next. The system always makes progress towards the final state where each job has arrived at its target. The state graph can thus not have a cycle, and we may consider states in a topological order. We formulate a dynamic program that computes for each state the smallest partial completion time to reach the state, where the partial completion time is defined as the sum of completion times of all completed jobs plus the current time for each uncompleted job. The dynamic program is well-defined as each value only depends on predecessor states. □

We conclude a complementary result to Theorem 1.

Corollary 2. *The bidirectional scheduling problem can be solved in polynomial time if m and κ are constant, $\tau_i = 1$ for each $i \in M$, and $p_j = 0$ for each $j \in J$.*

Proof. Since all release dates are integer, at each integer point in time no jobs are running on any segment. We can thus use a simpler version of the dynamic program we introduced in the proof of Theorem 6. □

References

1. F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proc. 40th Symposium on Foundations of Computer Science (FOCS)*, pages 32–43, 1999.
2. A. Antoniadis, N. Barcelo, D. Cole, K. Fox, B. Moseley, M. Nugent, and K. Pruhs. Packet forwarding algorithms in a line network. In *Proc. 11th Latin American Theoretical Informatics Symposium (LATIN)*, volume 8392 of *LNCS*, pages 610–621. 2014.
3. P. Berman, M. Karpinski, and A. D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. *Electronic Colloquium on Computational Complexity (ECCC)*, 10(49), 2003.
4. P. Brucker, S. Knust, and G. Wang. Complexity results for flow-shop problems with a single server. *European J. Oper. Res.*, 165:398–407, 2005.
5. Bundesamt für Seeschifffahrt und Hydrographie (BSH). *German Traffic Regulations for Navigable Maritime Waterways*. Hamburg and Rostock, Germany, 2013.
6. C. Chekuri and S. Khanna. A PTAS for minimizing weighted completion time on uniformly related machines. In *Proc. 28th Colloquium on Automata, Languages and Programming (ICALP)*, pages 848–861. 2001.
7. A. V. Fishkin, K. Jansen, and M. Mastrolilli. On minimizing average weighted completion time: A PTAS for the job shop problem with release dates. In *Proc. 14th Symposium an Algorithms and Computation (ISAAC)*, volume 2906 of *LNCS*, pages 319–328. 2003.
8. M. R. Garey and D. S. Johnson. *Computers and intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, 1979.
9. M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.*, 1(2):117–129, 1976.
10. H. Hoogeveen, P. Schuurman, and G. J. Woeginger. Non-approximability results for scheduling problems with minsum criteria. In *Proc. 6th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 353–366. 1998.
11. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. 1972.
12. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In *Handbooks in Operations Research and Management Science*, volume 4, pages 445–522. 1993.
13. F. T. Leighton, Bruce M. Maggs, and Satish B. Rao. Packet routing and job-shop scheduling in o(congestion+dilation) steps. *Combinatorica*, 14(2):167–186, 1994.
14. J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Ann. Discrete Math.*, 1:343–362, 1977.
15. Elisabeth Lübbecke, Marco E. Lübbecke, and Rolf H. Möhring. Ship traffic optimization for the Kiel Canal. Technical Report 4681, Optimization Online, 12 2014.
16. R. M. Lusby, J. Larsen, M. Ehrgott, and D. Ryan. Railway track allocation: models and methods. *OR Spectrum*, 33(4):843–883, 2011.
17. B. Peis and A. Wiese. Universal packet routing with arbitrary bandwidths and transit times. In *Proc. 15th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 362–375, 2011.
18. C. N. Potts and M. Y. Kovalyov. Scheduling with batching: A review. *European J. Oper. Res.*, 120(2):228 – 249, 2000.
19. M. Queyranne and M. Sviridenko. New and improved algorithms for minsum shop scheduling. In *Proc. 11th Symposium on Discrete Algorithms (SODA)*, pages 871–878, 2000.
20. C. Scheideler. Offline routing protocols. In *Universal Routing Strategies for Interconnection Networks*, pages 57–71. 1998.
21. C. A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85 – 89, 1984.

A Proofs of Section 3 – Hardness of bidirectional scheduling

In this section, we give a detailed proof of the hardness of the bidirectional scheduling problem for a constant number of segments and identical processing and transit times. We describe our reduction from MaxCut. Let an instance $\mathcal{I} = (G_\mathcal{I}, k)$ of MaxCut be given, with $G = (V_\mathcal{I}, E_\mathcal{I})$, $|V_\mathcal{I}| = n_\mathcal{I}$, and $|E_\mathcal{I}| = m_\mathcal{I}$. We introduce a set of jobs on polynomially many segments that can be scheduled with a total waiting time of W if and only if $\mathcal{I}$ admits a solution. Our construction is comprised of various gadgets which we describe in the following. We make use of suitably large parameters $x \gg y \gg z \gg 1$ that we will specify later. For example, x is chosen in such a way that if ever x jobs are located at the same segment, these jobs need to be processed immediately in order to achieve a waiting time of W. Note that because jobs take no time in being processed (i.e., $p_j = 0$), we can schedule any number of jobs sharing direction simultaneously on a single segment. Also, since $\tau = 1$, it makes no sense for a segment to stay idle if jobs are available. This allows us to restrict our analysis to schedules that are *sensible* in the sense that for each segment and at every time step all jobs in one direction available at the segment get scheduled. On the other hand, the non-zero transit time induces a cost of switching the direction of jobs that are processed at a segment.

Vertex gadget. Each of the segments $1, 10, 19, 28, \ldots$ hosts one vertex gadget for each of the vertices in $V_\mathcal{I}$ (cf. Figure 3 with the following). Each vertex gadget g_t on segment $9\ell + 1$ occupies a distinct time interval $[13t, 13(t + 1))$, $t < n_\mathcal{I}$, on the segment and is associated with one of the vertices $v \in V_\mathcal{I}$. The gadget comes with $24y$ *vertex jobs* that only need to be processed at segment $9\ell + 1$, half of them being leftbound, half being rightbound. Exactly y jobs of each direction are released at times $13t, 13t + 1, \ldots, 13t + 11$. We say that g_t is scheduled *consistently* if either all leftbound vertex jobs are processed immediately when they are released and all rightbound jobs wait for one time unit, or vice-versa. We say the gadget is in the *leftbound* (*rightbound*) *state* and interpret this as vertex v being part of set V_1 (V_2) of the partition of $V_\mathcal{I} = V_1 \dot\cup V_2$ we are implicitly constructing. A schedule is *consistent* if all vertex gadgets are scheduled consistently. The following lemma allows us to distinguish consistent schedules.

Lemma 2. *The vertex jobs of a single vertex gadget can be scheduled consistently with a waiting time of* $12y$*, while every inconsistent schedule has waiting time at least* $13y$*.*

Proof. Since $p = 0$, we can schedule all available jobs with the same direction simultaneously. It follows that both consistent schedules are valid, and, since in both exactly half of the vertex jobs wait for one unit of time, the total waiting time of such a schedule is $12y$. Any inconsistent (sensible) schedule would have to send jobs in the same direction in two consecutive unit time intervals, which means that in addition to the minimum waiting time of $12y$, at least y jobs have to wait an extra unit of time. □

Synchronizing vertex gadgets. Since every vertex $v \in V_\mathcal{I}$ is represented by multiple vertex gadgets on different segments, we need a way to ensure that all vertex gadgets for v are in agreement regarding which part of the partition v is assigned to. We introduce two different gadgets that handle synchronization. The

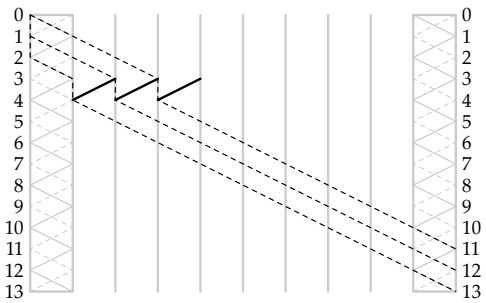


Fig. 6: Illustration of the copy gadget between two vertex gadgets. The dashed lines depict all sensible trajectories of the synchronizing jobs, assuming that the vertex gadgets are in the same state.

copy gadget synchronizes the vertex gadgets g_t occupying the same time interval on segments $9\ell + 1$ and $9\ell + 10$, while the *transposition gadget* synchronizes gadgets g_t, g_{t+1} on segment $9\ell + 1$ with gadgets g_{t+1}, g_t on segment $9\ell + 10$. Using a combination of copy and transposition gadgets, we can transition between any two orders of vertex gadgets on distant segments.

We first specify the copy gadget that synchronizes the vertex gadgets g_t on two segments $9\ell + 1$ and $9\ell + 10$ (cf. Figure 6 with the following). The gadget consists of $2z$ rightbound *synchronization jobs*, half of which are released at time $13t$ and half at time $13t + 1$. The jobs need to be processed on all segments $9\ell + 1, \ldots, 9\ell + 10$ in this order. In addition, we introduce $3x$ *blocking jobs* that are used to enforce that specific time intervals on a segment are reserved for leftbound/rightbound operation. Essentially, releasing x blocking jobs at time t on a single segment prevents any jobs to be processed in opposite direction during the time interval $[t, t + 1)$ (and even earlier). In this manner, we block the interval starting at time $13t + 3$ on segments $9\ell + 2, 9\ell + 3, 9\ell + 4$.

Lemma 3. *In any consistent schedule, the synchronization jobs of a single copy gadget can be scheduled with a waiting time of* $3z$ *if the two corresponding vertex gadgets are in the same state, otherwise their waiting time is at least* $5z$*.*

Proof. Since $x \gg z$, we need to schedule all blocking jobs as soon as they are released. If both vertex gadgets g_t linked by the copy gadget are in the rightbound state, the synchronization jobs released at time $13t$ only have to wait for one time unit at segment $9\ell + 4$, while the other jobs have to wait at segments $9\ell + 1$ and $9\ell + 2$. Similarly, if the vertex gadgets are in the leftbound state, the first half of the jobs have to wait at segments $9\ell + 1$ and $9\ell + 3$, while the other half only has to wait at segment $9\ell + 3$. The waiting time in either case is $3z$. If the vertex gadgets are in opposite states, all jobs have to additionally wait at segment $9\ell + 10$, which results in a total waiting time of at least $5z$. □

We now describe the transposition gadget that synchronizes the vertex gadgets g_t, g_{t+1} on segment $9\ell + 1$ with the vertex gadgets g_{t+1}, g_t on segment $9\ell + 10$ (cf. Figure 7 with the following). The challenge here is that jobs synchronizing the different pairs of vertex gadgets need to pass each other without interfering. We achieve this by making sure that the jobs never meet while being in transit at the same segment. The gadget consists of $4z$ synchronization jobs, half being rightbound and half being leftbound. Half of each are released at times $13t + 6$ and $13t + 7$, and all need to be processed at segments $9\ell + 1, \ldots, 9\ell + 10$ (in different directions).

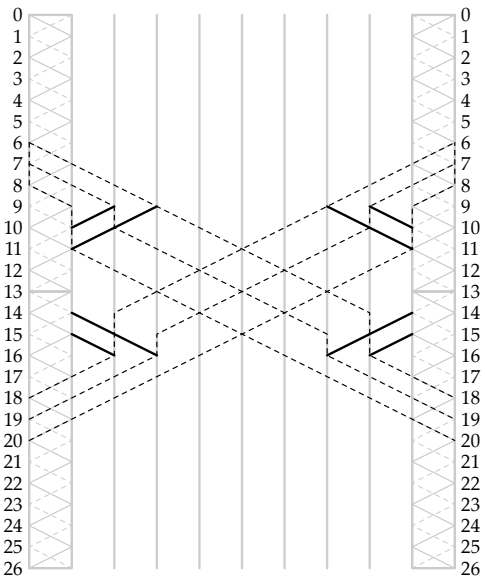


Fig. 7: Illustration of the transposition gadget. The dashed lines depict all sensible trajectories of the synchronizing jobs, assuming that the vertex gadgets are pairwise in the same states. Note that jobs in different directions never meet while in transit through the same segment.

In addition, we introduce $12x$ blocking jobs to block the intervals starting at the following times: at times $13t + 9$, $13t + 10$ for rightbound jobs and at times $13t + 14$, $13t + 15$ for leftbound jobs on segment $9\ell + 2$, at times $13t + 9$ for rightbound and at $13t + 15$ for leftbound on segment $9\ell + 3$, and the corresponding (symmetrical) intervals in opposite direction on segments $9\ell + 8$ and $9\ell + 9$ (cf. Figure 7).

Lemma 4. *In any consistent schedule, the synchronization jobs of a single transposition gadget can be scheduled with a waiting time of* $10z$ *if each of the two pairs of corresponding vertex gadgets are in the same state, otherwise their waiting time is at least* $12z$*.*

Proof. Since $x \gg z$, we need to schedule all blocking jobs as soon as they are released. It is easy to verify that all synchronization jobs wait at exactly 2 segments due to blocking jobs. In addition, half of the jobs wait for one unit of time at the segment where they are released – for a total of $10z$ time units. If the pair of vertex gadgets is in opposite states, all connecting synchronization jobs need to wait at least one additional unit of time at their last segment. Observe that synchronization jobs in opposite directions are never in transit on the same segment at the same time. □

Edge gadget. The purpose of an edge gadget between vertex gadget g_t on segment $9\ell + 1$ and g_{t+1} on segment $9\ell + 10$ is to produce a small additional waiting time if the two vertex gadgets are in the same state (cf. Figure 8 with the following). We will introduce edge gadgets between vertex gadgets representing two vertices u, v that share an edge in G. This way, every edge that connects vertices in different parts of the partition is beneficial for the resulting waiting time. The edge gadget itself consists of 2 rightbound *edge jobs*, one being released at time $13t + 7$ and the

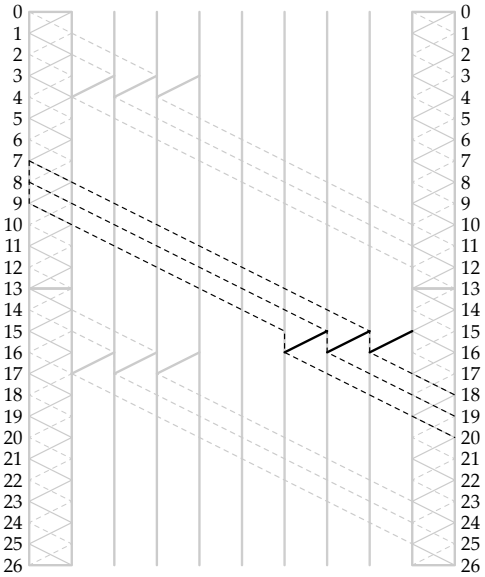


Fig. 8: Illustration of the edge gadget. The dashed lines depict all sensible trajectories of the synchronizing jobs, assuming that the vertex gadgets are in opposite states. Note that edge jobs do not interact with synchronization jobs of copy gadgets for both vertices.

other at time $13t+8$. Both jobs need to be processed on segments $9\ell+1,\ldots,9\ell+10$. We add $3x$ blocking jobs to block the unit time interval starting at time $13t+15$ on segments $9\ell+7, 9\ell+8, 9\ell+9$.

Lemma 5. *In any consistent schedule, the edge jobs of a single edge gadget can be scheduled with a waiting time of* 3 *if the two connected vertex gadgets are in opposite states, otherwise their waiting time is at least* 5*.*

Proof. One job always has to wait for a time unit at the first segment. Both jobs have to wait for the blocking jobs (since $x \gg 1$). If the vertex gadgets are in the same state, both jobs have to wait an additional unit of time at the last segment. □

Construction. We are now ready to combine our gadgets and explain the final construction.

Theorem 1. *The bidirectional scheduling problem is* NP*-hard even if* $p_j = 0$ *and* $\tau_i = 1$ *for each* $j \in J$ *and* $i \in M$*.*

Proof. We start by introducing a vertex gadget g_t on segment 1 for each vertex $v_t \in V_{\mathcal{I}}$ of the given MaxCut-instance. For each edge $\{u, v\}$ we extend the construction by appending more segments as follows. We add a sequence of blocks of 9 segments, the last of which contains again a vertex gadget for each vertex. In between we add copy and transposition gadgets in such a way that on the last segment i the vertex gadgets g_0 and g_1 represent the vertices u and v. We can achieve this by adding less than $n_{\mathcal{I}}$ segments. We add an additional block of 9 segments, and add copy gadgets for each of the variables. Finally, we add an edge gadget connecting vertex gadget g_0 on segment i with g_1 on the last segment. Observe that the edge jobs do not interfere with any of the synchronization jobs for the copy gadgets for the first two

vertices (cf. Figure 8). We repeat the process once for each edge. The total number of segments is $\mathcal{O}(n_{\mathcal{I}} m_{\mathcal{I}})$, and the total number of jobs is $\mathcal{O}(n_{\mathcal{I}}^2 m_{\mathcal{I}}(x+y+z))$. The number of vertex gadgets is $n_v < n_{\mathcal{I}}^2 m_{\mathcal{I}}$, and the number of transposition and copy gadgets is $n_t < n_c < n_v$.

We claim that if the MaxCut instance admits a solution $\mathcal{S}$, we can schedule all jobs with waiting time at most $W = 12n_v y + 3n_c z + 10n_t z + 5m_{\mathcal{I}} - 2k$. We do this by scheduling all vertex gadgets consistently in the state corresponding to the part of the partition the corresponding vertex belongs to in $\mathcal{S}$. Lemmas 2 through 4 guarantee that we can schedule everything but the edge jobs without incurring a waiting time greater than $12n_v y + 3n_c z + 10n_t z$. Finally, since at least k edges in the MaxCut solution are between vertices in different sets of the partition, and the vertex gadgets are set accordingly, by Lemma 5, we obtain an additional waiting time of at most $5m_{\mathcal{I}} - 2k$ as claimed.

It remains to establish that the waiting time exceeds W in case the MaxCut instance does not admit a solution. We set $x = W + 1$, such that all blocking jobs have to be scheduled as soon as they are released. By Lemma 2, scheduling at least one vertex gadget inconsistently produces a total waiting time of at least $12n_v y + y$. We now set $y = 18n_{\mathcal{I}}^2 m_{\mathcal{I}} z > 3n_c z + 10n_t z + 5m_{\mathcal{I}}$ for the vertex jobs, such that a single inconsistent vertex gadget results in a waiting time greater than W. Hence, each vertex gadget needs to be scheduled consistently. By Lemmas 3 and 4, we have that if not all vertex gadgets corresponding to the same vertex are in the same state, the waiting time for vertex and synchronization jobs is at least $12n_v y + 3n_c z + 2n_t z + z$. We set $z = 5m_{\mathcal{I}}$, which allows us to conclude that all vertex gadgets are in agreement regarding the partition of the vertices. Finally, Lemma 5 enforces that there are at least k edge gadgets between vertices in different states. This however is impossible as our MaxCut instance does not admit a solution. □

Corollary 1. *The bidirectional scheduling problem is* NP*-hard even if* $p_j = 1$ *and* $\tau_i = \tau$ *for each* $j \in J$ *and* $i \in M$.

Proof. We adapt our construction by setting $p = 1$ and $\tau = n^2 m$ and scaling all release times by $n^2 m$, where n, m are the number of jobs and segments, respectively. We claim that the original instance admits a solution of some waiting time W if and only if it now admits a solution with waiting time in $[W\tau, (W+1)\tau)$. This proves the Corollary, as the intervals are pairwise disjoint for different (integer) values of W.

If the original construction (with $p = 0$ and $\tau = 1$) does not admit a solution with waiting time at most W, then a scaled version with $p = 0$ and $\tau = n^2 m$ does not admit a solution with waiting time at most $W\tau$. But the lowest possible waiting is monotonically increasing with increasing processing times, hence the adapted instance with $p = 1$ does not admit a solution of waiting time at most $W\tau$.

Conversely, assume we have a solution of the original instance with waiting time W. We fix the order in which jobs are processed along each segment and construct a schedule for the setting $p = 1$, $\tau = n^2 m$ by introducing additional waiting periods for each job. Clearly, each job has to wait at most one time unit for each other job to be processed at each segment. Hence, the additional waiting time overall is smaller than $n^2 m = \tau$. □

B Proofs of Section 4 – A PTAS for bidirectional scheduling

In this Section we state the Lemmas with detailed proofs that are necessary to show the existence of a PTAS if the processing times of the jobs are not restricted to be equal in the case of a single segment. More precisely, we consider the bidirectional scheduling problem on a single segment with compatibility graph $G_1 \in \{K_{n_r,n_l}, \emptyset\}$. Following the proof scheme of [1], we introduce several lemmas that allow us to make assumptions at "$\mathcal{O}(1+\varepsilon)$-loss", meaning that we can modify any input instance and optimum schedule to adhere to these assumptions, such that the resulting schedule is within a factor polynomial in $(1+\varepsilon)$ of the optimum schedule for the original instance. To not complicate matters unnecessarily, in the following we allow fractional release dates and processing times.

Lemma 6. *With $\mathcal{O}(1+\varepsilon)$-loss we can assume that $r_j, p_j \in \{(1+\varepsilon)^x \mid x \in \mathbb{N}\} \cup \{0\}$, $r_j \geq \varepsilon(p_j + \tau_1)$, and $r_j \geq 1$ for each job $j \in J$.*

Proof. Increasing any value $v \in \mathbb{R}$ to the smallest power of $(1+\varepsilon)$ not smaller than v yields a value $v' = (1+\varepsilon)^x = (1+\varepsilon)(1+\varepsilon)^{x-1} \leq (1+\varepsilon)v$. Hence, multiplying all start times of a schedule by $(1+\varepsilon)$ gives a feasible schedule even when rounding up all nonzero processing times to the next power of $(1+\varepsilon)$. The total completion time does not increase by more than a factor of $(1+\varepsilon)$.

By shifting the completion times of a schedule with adapted processing times by a factor of $(1+\varepsilon)$, we obtain increased start times S'_j for each job j:

$$S'_j = (1+\varepsilon)C_j - (p_j + \tau_1) = (1+\varepsilon)(S_j + p_j + \tau_1) - (p_j + \tau_1) \geq \varepsilon(p_j + \tau_1).$$

Hence, by losing not more than a $(1+\varepsilon)$-factor we may assume that all jobs have release dates of at least an ε fraction of their running time. Now, we can scale the instance by some power of $(1+\varepsilon)$, such that the earliest release date is at least one (since jobs with $r_j = p_j = \tau_1 = 0$ can be ignored).

Finally, multiplying again all start times of a schedule with adapted processing times and release dates by $(1+\varepsilon)$ yields a feasible schedule even when rounding up all nonzero release dates to the next power of $(1+\varepsilon)$. □

We define $R_x = (1+\varepsilon)^x$ and consider time intervals $I_x = [R_x, R_{x+1}]$ of length εR_x.

Lemma 7. *Each job runs for at most $\sigma := \lceil \log_{1+\varepsilon} \frac{1+\varepsilon}{\varepsilon} \rceil$ intervals, i.e., a job starting in interval I_x is completed before the end of $I_{x+\sigma}$.*

Proof. Consider some job j and assume that j starts in I_x in some schedule. By Lemma 6 we get

$$|I_x| = \varepsilon R_x \geq \varepsilon r_j \geq \varepsilon^2(p_j + \tau_1).$$

Thus, the running time of j is bounded by $|I_x|/\varepsilon^2$. The constant upper bound of $1/\varepsilon^2$ for the number of used intervals can still be improved since the length of the next σ succeeding intervals with increasing size is sufficient to cover a length of $|I_x|/\varepsilon^2$. Using the fact that $\sum_{k=0}^{n} z^k = \frac{1-z^{n+1}}{1-z}$ we get

$$\begin{aligned}
\sum_{i=0}^{\sigma} |I_{x+i}| &= \sum_{i=0}^{\sigma} (R_{x+i+1} - R_{x+i}) = |I_x| \sum_{i=0}^{\sigma} (1+\varepsilon)^i \\
&= |I_x| \frac{1-(1+\varepsilon)^{\sigma+1}}{1-(1+\varepsilon)} \\
&\geq |I_x| \frac{1-\frac{1+\varepsilon}{\varepsilon}}{-\varepsilon} = |I_x| \frac{1+\varepsilon-\varepsilon}{\varepsilon^2} = \frac{|I_x|}{\varepsilon^2},
\end{aligned}$$

which concludes the proof. □

We use the common technique of *time-stretching*. We shift each start time (or completion time) to the next interval while maintaining the same offset to the beginning of the interval. This way, the schedule remains feasible and the objective is increased by a factor of at most $(1+\varepsilon)$. Intuitively, this process can be interpreted as stretching the length of each time interval I_x by a factor of $(1+\varepsilon)$, i.e., its length is increased by $\varepsilon|I_x|$. When applying (multiple) time-stretches we use the following observation to assess the additional empty space created between jobs:

Lemma 8. *Consider two distinct times $T_1 < T_2$ with $T_1 \in I_{x(1)}$ and $T_2 \in I_{x(2)}$. Applying ℓ time-stretches yields shifted times $T_1' < T_2'$ with*

$$(T_2' - T_1') \geq (T_2 - T_1) + \Xi[x(1), x(2)], \tag{1}$$

where $\Xi[x(1), x(2)] := \sum_{x(1)\leq x<x(2)} \ell\varepsilon|I_x|$.

Proof. We calculate

$$\begin{aligned}(T_2' - T_1') &= R_{x(2)+\ell} + (T_2 - R_{x(2)}) - [R_{x(1)+\ell} + (T_1 - R_{x(1)})] \\ &= ((1+\varepsilon)^\ell - 1)R_{x(2)} + T_2 - ((1+\varepsilon)^\ell - 1)R_{x(1)} - T_1 \\ &\geq (T_2 - T_1) + (1 + \ell\varepsilon - 1)(R_{x(2)} - R_{x(1)}) \\ &= (T_2 - T_1) + \ell\varepsilon \sum_{x(1)\leq x<x(2)} |I_x|.\end{aligned}$$

□

We can now apply time-stretches to the start or completion times of all jobs and use the above observation to quantify the additional space created in the schedule. Consider two jobs $j, k \in J$ with starting times $S_j < S_k$, and let $s(j), s(k)$ (resp. $c(j), c(k)$) denote the intervals in which their start (completion) times fall, i.e., $S_j \in I_{s(j)}$ (and $C_j \in I_{c(j)}$). E.g., if we apply ℓ time-stretches to starting times, we obtain an additional gap of $\Xi[s(j), s(k)]$ between the new starting and completion times. Table 2 summarizes the resulting gaps depending on whether start or completion times are stretched and whether j, k travel in the same or opposite directions.

Table 2: Summary of the increased differences between start and completion times of jobs $j, k \in J$, $S_j < S_k$, when stretching start times (denoted by $'$) or completion times (denoted by $''$). We use Lemma 8 together with the fact that j and k did not overlap before the time-stretch.

time-stretch on	same direction	opposite direction
start times (1)	$S_k' \geq S_j' + p_j + \Xi[s(j), s(k)]$	$S_k' \geq S_j' + p_j + \tau + \Xi[s(j), s(k)]$
	$\Rightarrow C_k' \geq C_j' + p_k + \Xi[s(j), s(k)]$	$C_k' \geq C_j' + p_k + \tau + \Xi[s(j), s(k)]$
compl. times (1)	$C_k'' \geq C_j'' + p_k + \Xi[c(j), c(k)]$	$C_k'' \geq C_j'' + p_k + \tau + \Xi[c(j), c(k)]$
	$\Rightarrow S_k'' \geq S_j'' + p_j + \Xi[c(j), c(k)]$	$S_k'' \geq S_j'' + p_j + \tau + \Xi[c(j), c(k)]$

To analyze the set of jobs released within each interval we partition them as follows. A job j released at R_x is called *small* if $p_j \leq \frac{\varepsilon^2}{4}|I_x|$ and *large* otherwise. With

this, we partition for each direction $\mathrm{d} \in \{\mathrm{r}, \mathrm{l}\}$ the jobs $J_x^{\mathrm{d}} := \{j \in J^{\mathrm{d}} \mid r_j = R_x\}$ released at R_x into the subsets $S_x^{\mathrm{d}} = \{j \in J_x^{\mathrm{d}} \mid j \text{ is small}\}$ and $L_x^{\mathrm{d}} = \{j \in J_x^{\mathrm{d}} \mid j \text{ is large}\}$. We will see that the arrangement of jobs of each S_x^{d} does not influence the remaining jobs too much such that we can assume a fixed order for each of these sets. To do so, we say that a subset $J' \subseteq J$ of jobs is scheduled in *SPT order* (shortest processing time first) if $S_{j_1} \leq S_{j_2}$ for any pair of jobs $j_1, j_2 \in J'$ with $p_{j_1} < p_{j_2}$. Furthermore, we denote the sum of processing times of J' as $p(J')$ and the union of small jobs released up to some point R_x with direction $\mathrm{d} \in \{\mathrm{r}, \mathrm{l}\}$ by $S_{\leq x}^{\mathrm{d}} = \bigcup_{x' \leq x} S_{x'}^{\mathrm{d}}$.

Lemma 9. *With $\mathcal{O}(1+\varepsilon)$-loss we can restrict to schedules such that for each $x \geq 0$ and each $\mathrm{d} \in \{\mathrm{r}, \mathrm{l}\}$:*

1. *the processing of no small job contains a release date,*
2. *jobs contained in $S_{\leq x}^{\mathrm{d}}$ are scheduled in SPT order within I_x, and*
3. *$p(S_x^{\mathrm{d}}) \leq |I_x|$.*

Proof. To prove claim 1 we consider some schedule and apply a time-stretch via start times. Observe that no further crossing of a processing over a release date is produced for small jobs. If there was a release date $R_{s(j)+1}$ contained in the processing interval of a small job of $I_{s(j)}$ it is moved behind the processing since we get by Lemma 8 that $R_{s(j)+2} - S'_j \geq R_{s(j)+1} - S_j + \varepsilon |I_{s(j)}|$ which gives an increase larger than the processing time of this job.

For a proof of claim 2 consider a schedule S where no processing of a small job contains a release date and apply one time-stretch via start times. This increases the objective value by at most a $1+\varepsilon$ factor. Denote the resulting schedule as S'. To achieve the demanded properties, apply the following procedure for each direction $\mathrm{d} \in \{\mathrm{r}, \mathrm{l}\}$. First, remove all small jobs from schedule S'. Now consider each interval $I_x, x = 0, 1, \ldots$. Denote by A_x the set of removed jobs from I_x. If jobs have been removed in I_x there are idle intervals where jobs in direction d can be scheduled. Denote the subset of $S_{\leq x}^{\mathrm{d}}$ already scheduled in earlier intervals by $B_{<x}$ and order the subset $C_x := S_{\leq x}^{\mathrm{d}} \setminus B_{<x}$ of unscheduled jobs in SPT order. Define for $t \in I_x$ by $p_t(A_x) := p(\{j \in A_x \mid S'_j \leq t\})$ the amount of processing time of jobs started before time t in S'. Now let $C_x(t)$ be the smallest SPT-subset of C_x such that $p(C_x(t)) \geq p_t(A_x)$ or $C_x(t) = C_x$. Iterate from the earliest created maximal empty interval to the latest and fill each interval $[t_1, t_2]$ in SPT order such that the jobs of $p(C_x(t_2))$ start before t_2. Note that $p(C_x(t_2)) \leq p_{t_2}(A_x) + \frac{\varepsilon^2}{4}|I_x|$ since we consider only small jobs. To maintain feasibility we increase the start of the following jobs from $J \setminus C_x$, if necessary. (This decreases eventually the size of the following empty interval which is no problem). Nevertheless, the start time of no job from $J \setminus C_x$ is increased by more than $\frac{\varepsilon^2}{4}|I_x|$. Hence, their completion time is increased by less than a $1+\varepsilon$ factor and the jobs starting after R_{x+1} are not affected. Note that no processing of the assigned small jobs $B_x := C_x(R_{x+1})$ contains R_{x+1}.

Since we used in each interval an assignment via SPT order we know that at each point in time the number of already started small jobs has not been decreased. Therefore, the total completion time of small jobs overall has not been increased.

To prove claim 3 consider for each $x = 0, 1, \ldots$ the largest SPT-subset J'_x of S_x^{d}, such that $p(J'_x) \leq |I_x|$. By assumptions 2 and 1 we can be sure that all jobs of $S_x^{\mathrm{d}} \setminus J'_x$ are not scheduled within I_x and thus, we can move their release dates to R_{x+1}. □

Once, we have a fixed order to schedule small jobs with the same release date we are able to glue them to job packs of a certain minimum size. For this purpose we apply a further time-stretch to join the processing of jobs assigned to the same

pack. This increases for each interval I_y the amount of processing per direction and each earlier interval I_x by at most the size of one job being small at time R_x. The following lemma yields that the extra space of one interval created by one time-stretch is sufficient to cover this amount for all earlier Intervals.

Lemma 10. *We have* $\sum_{x<y} \varepsilon^2 |I_x| \leq \varepsilon |I_y|$.

Proof. To prove the claim we again use that $\sum_{k=0}^{n} z^k = \frac{1-z^{n+1}}{1-z}$:

$$\varepsilon^3 \sum_{x<y} (1+\varepsilon)^x = \varepsilon^3 \frac{1-(1+\varepsilon)^y}{1-(1+\varepsilon)} = \varepsilon^2((1+\varepsilon)^y - 1) \leq \varepsilon |I_y|.$$

□

Lemma 11. *With* $\mathcal{O}(1+\varepsilon)$*-loss we can restrict to schedules such that for each* $x \geq 0$ *and each* $\mathrm{d} \in \{\mathrm{r}, \mathrm{l}\}$ *the jobs of* S_x^{d} *in SPT order are joined to unsplittable job packs with size of at most* $\frac{\varepsilon^2}{4}|I_x|$ *and at least* $\frac{\varepsilon^2}{8}|I_x|$ *each.*

Proof. Consider a schedule satisfying at $\mathcal{O}(1+\varepsilon)$-loss the properties of Lemma 9 and apply one time-stretch via start times. We now apply the following procedure for each direction $\mathrm{d} \in \{\mathrm{r}, \mathrm{l}\}$ and each $x = 0, 1, \ldots$. Recall that the jobs of S_x^{d} are scheduled in SPT order. Let $T_x^{\mathrm{d}} = \{j \in S_x^{\mathrm{d}} \mid p_j < \frac{\varepsilon^2}{8}|I_x|\}$ be the subset of jobs being too small. Remove the jobs of T_x^{d} from the current schedule and join the jobs of T_x^{d} successively in SPT order to minimal job packs such that the processing times of each job pack sum up to at least $\frac{\varepsilon^2}{8}|I_x|$. (The processing time of the last pack is artificially increased if necessary.) We now reassign complete job packs to the empty intervals similarly to the procedure in the proof of Lemma 9. Hence, no start time of T_x^{d} has been increased and the start time of no job in $J \setminus T_x^{\mathrm{d}}$ has been increased by more than $\frac{\varepsilon^2}{4}|I_x|$.

In total, the start time of no job starting in interval I_{y+1} has been increased by more than $2 \cdot \sum_{x<y} \frac{\varepsilon^2}{4}|I_x| + 2 \cdot \frac{\varepsilon^2}{4}|I_y| \leq \varepsilon |I_y|$ due to Lemma 10. By Lemma 8 (or Table 2) we can conclude that no job has been delayed to a later interval by the rearrangement. Note that properties 1 and 3 of Lemma 9 still hold whereas property 2 (SPT order) remains true only within each S_x^{d}. □

Therefore, we can consider each job pack simply as one small job. Nevertheless, the original jobs must be used for the evaluation of the completion times. Besides the scheduling restrictions for small jobs we can also bound the amount of large jobs released at the beginning of each interval.

Lemma 12. *With* $\mathcal{O}(1+\varepsilon)$*-loss we can assume for each* $x \geq 0$ *and each* $\mathrm{d} \in \{\mathrm{r}, \mathrm{l}\}$ *that:*

1. *the number of possible processing times in* L_x^{d} *is bounded by* $5 \log_{(1+\varepsilon)} \frac{1}{\varepsilon}$*, and*
2. *the number of jobs per processing time in* L_x^{d} *is bounded by* $\frac{4}{\varepsilon^2}$*.*

Proof. Consider some scheduling instance, some $\mathrm{d} \in \{\mathrm{r}, \mathrm{l}\}$ and some $x \geq 0$. The processing time of the jobs in L_x^{d} are, by definition, at least $\frac{\varepsilon^3}{4}(1+\varepsilon)^x$. On the other hand, by Lemma 6, the processing times are at most $\frac{1}{\varepsilon}(1+\varepsilon)^x$. Let x_j be such that $p_j = (1+\varepsilon)^{x_j}$. We get

$$\begin{aligned} & \tfrac{\varepsilon^3}{4} \leq \tfrac{(1+\varepsilon)^{x_j}}{(1+\varepsilon)^x} \leq \tfrac{1}{\varepsilon} \\ \implies \quad & \log_{(1+\varepsilon)} \tfrac{\varepsilon^3}{4} \leq x_j - x \leq \log_{(1+\varepsilon)} \tfrac{1}{\varepsilon} \end{aligned}$$

The difference of these bounds is $4\log_{(1+\varepsilon)}\frac{1}{\varepsilon}+\log_{(1+\varepsilon)}4$ which gives a constant number of possible integer values for x_j and, hence, a constant number of possible processing times for each job in L_x^{d}. Finally, since each large job in I_x has a processing time of at least $\frac{\varepsilon^2}{4}|I_x|$, we can schedule at most $4/\varepsilon^2$ jobs per direction within I_x, and the remaining jobs need to start after R_{x+1}. □

Lemma 13. *With $\mathcal{O}(1+\varepsilon)$-loss we can assume, that each job is finished within a constant number of intervals after its release.*

Proof. Consider the set of jobs J_x released at time R_x. By Lemma 6 the running time of each such job is at most R_x/ε. Therefore, applying Lemmas 9 and 12 we can bound the time needed to first schedule all jobs of one direction and afterward all jobs of the other direction:

$$\begin{aligned}\sum_{\mathrm{d}\in\{\mathrm{r},\mathrm{l}\}}\left[p(S_x^{\mathrm{d}})+p(L_x^{\mathrm{d}})+\tau_1\right] &\le 2\Bigg[\varepsilon(1+\varepsilon)^x \\ &\qquad +\frac{4}{\varepsilon^2}\cdot\frac{1}{\varepsilon}(1+\varepsilon)^x\cdot 5\log_{(1+\varepsilon)}\frac{1}{\varepsilon}\Bigg] \\ &= \varepsilon^2(1+\varepsilon)^x\cdot 2\left[\frac{1}{\varepsilon}+\frac{20}{\varepsilon^5}\log_{(1+\varepsilon)}\frac{1}{\varepsilon}\right] \\ &\le \varepsilon^2(1+\varepsilon)^x(1+\varepsilon)^{\sigma'-1}=\varepsilon|I_{x+\sigma'-1}|,\end{aligned}$$

where σ' is the smallest possible integer such that $2\left[\frac{1}{\varepsilon}+\frac{20}{\varepsilon^5}\log_{(1+\varepsilon)}\frac{1}{\varepsilon}\right]\le(1+\varepsilon)^{\sigma'-1}$. Note, that σ' is constant.

Applying one time-stretch on the start times creates idle time for each interval I_x somewhere after σ' intervals that is sufficient to host all unfinished jobs of J_x, cf. Lemma 8 and Table 2. If no job was running at time $R_{x+\sigma'}$ before the time-stretch this created idle time is now part of interval $I_{x+\sigma'}$. Otherwise let j be the latest of these jobs with start time $S_j\in I_{s(j)}$ and completion time $C_j\in I_{c(j)}$ before the time-stretch. Note that $s(j)\le x+\sigma'-1$ which induces $c(j)\le x+\sigma'+\sigma-1$ due to Lemma 7. By Lemma 8 we can be sure that after the time-stretch there is idle time of $\sum_{x=s(j)}^{c(j)-1}\varepsilon|I_k|$ before 1. the start of the next job after j and 2. the end of interval I_{x_c+1}. By definition of σ', this time is sufficient to first schedule all jobs of J_x in heading of j and then all remaining. This way, all jobs of J_x are scheduled before the end of interval $I_{x+\sigma'+\sigma}$.

Note that this argument assumes that there are no compatibilities. An analog reasoning concerning only the processing times works if all opposed jobs are compatible. □

We can now limit the interface of our dynamic program by showing Lemma 1 of Section 4.

Lemma 1. *There is a schedule with a sum of completion times within a factor of $(1+\varepsilon)$ of the optimum and with the following properties:*

1. *The number of jobs scheduled in each block is bounded by a constant.*
2. *Every two consecutive blocks respect one of constantly many frontiers.*

Proof. By Lemma 11 we may assume that small jobs in S_x^{d} have processing time at least $\varepsilon^2|I_x|/8$. By Lemma 9, the total processing time of these jobs is at most $|I_x|$, and hence the number of jobs in S_x^{d} is bounded by a constant. The same is true for

large jobs, by Lemma 12. Finally, together with Lemma 13, this implies that the number of jobs running during each interval is bounded by a constant.

For the second property, we apply one time-stretch on the completion times. Consider now the latest job j of each direction that starts within block B_t and is completed in interval $I_{c(j)}$ of the following block. By Lemma 8 (and Table 2) we know that there is idle time of at least $\varepsilon|I_{c(j)} - 2|$ before the start of job j (or before the start of the earliest job aligned with j with completion time in $I_{c(j)}$ and start time in B_t. Hence, we can decrease the start time of these jobs such that the values C_j and $S_j + p_j$ fall below the next $\frac{1}{\varepsilon^2}$ fraction of $I_{c(j)}$, i.e., by an amount of at most $\varepsilon^2|I_{c(j)}| \leq \varepsilon|I_{c(j)} - 2|$. Hence, the first job starting in B_{t+1} (of each direction in case of compatibilities) can be scheduled at an $\frac{1}{\varepsilon^2}$ fraction of $I_{c(j)}$ without any further loss. Thus, we only need to consider $\frac{\sigma}{\varepsilon^2}$ possible frontier values per direction, or a total of $\left(\frac{\sigma}{\varepsilon^2}\right)^2$ possible frontiers.

□

C Proofs of Section 5 – Hardness of custom compatibilities

In this section we give a detailed hardness proof for bidirectional scheduling on a single segment where jobs can be compatible. Our proof holds even for unit processing and transit times. We first consider the makespan objective and extend the proof in a second step to waiting time and total completion time.

C.1 NP-Hardness of Makespan Minimization

Theorem 7. *Minimizing the makespan for $m = 1$ with an arbitrary compatibility graph G_1 is* NP*-hard even if $p_j = \tau_1 = 1$ for each $j \in J$.*

In the following, we explain the construction of a bidirectional scheduling instance for a given $(\leq 3, 3)$-Sat instance with variable set $X = \{x_i \mid i = 0, \ldots, |X|-1\}$ and clause set $C = \{c_k \mid k = 0, \ldots, |C|-1\}$. The constructed instance yields a demanded makespan $C_{\max}$ if and only if the given $(\leq 3, 3)$-Sat formula is satisfiable. For the construction, we partition the time horizon into four parts $P_1, \ldots, P_4$ with start time $A_1 = 0, A_2 = 6|X|, A_3 = 10|X|$, and $A_4 = 10|X| + 2|C|$. There is a (virtual) last part starting at time $A_5 = 12|X| + |C|$. The demanded makespan $C_{\max} = A_5 + 1$ will enforce that all jobs start before the end of the fourth part.

The rough idea is as follows: In the first four parts we release a tight frame of *blocking jobs* B and *dummy jobs* H that have to start running immediately at their release date in any schedule that achieves $C_{\max}$. We use these jobs to create gaps for *variable jobs* that represent the variable assignments. By defining the compatibilities for the blocking jobs we are able to control which of these variable jobs can be scheduled into each gap. In the first part of our construction, we release all variable jobs, which come in two *types*: one type representing a *true* assignment to the corresponding variable and the other type representing a *false* assignment. Our construction will enforce the following properties in each of its parts:

Lemma 14. *In every feasible schedule with makespan $C_{\max}$, all jobs released before A_3 are scheduled in parts P_1 and P_2, except for two rightbound variable jobs of same type for each variable.*

Lemma 15. *In every feasible schedule with makespan $C_{\max}$, the only jobs released before A_3 and scheduled in P_3 are rightbound variable jobs each corresponding to a variable assignment satisfying a different clause.*

Lemma 16. *In every feasible schedule with makespan $C_{\max}$, the only jobs released before A_4 and scheduled in P_4 are rightbound variable jobs, and there are not more than $2|X| - |C|$ of them.*

In the following we explicitly define the released jobs of each part achieving the above properties. Each part is accompanied by a figure illustrating when jobs are released, the respective compatibility graph and an example of a schedule. In all figures, time is directed downwards, and all rightbound jobs are depicted to the left and all leftbound jobs to the right of the segment. Since compatible jobs can run concurrently, the schedules of the leftbound and the rightbound jobs are drawn separately.

It is convenient to prove Lemmas 6 to 8 in reverse order. To this end, we start by specifying the jobs released in P_4.

Jobs released in P_4. In the fourth part, we release a set of $2|X| - |C|$ leftbound blocking jobs $B_4 = \{b_i \mid i = 0, \ldots, 2|X| - |C| - 1\}$. Each blocking job b_i is released at time $A_4 + i$. The purpose of a blocking job is to leaving space for a leftover rightbound variable job that has not been scheduled until the beginning of this part. Each blocking jobs $b_i \in B_4$ is only compatible with all rightbound variable jobs.

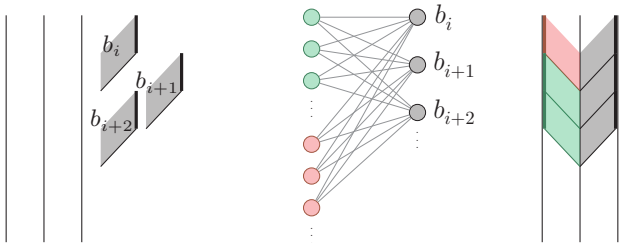


Fig. 9: Part P_4 with blocking jobs reserving space for all remaining rightbound variable jobs.

We are now in position to prove Lemma 16, i.e., in a schedule with makespan $C_{\max}$ the only jobs released before A_4 that can be scheduled in P_4 are up to $2|X| - |C|$ rightbound variable jobs.

Proof (Proof of Lemma 16). First, observe that with the required makespan of $A_5 + 1 = A_4 + 2|X| - |C| + 1$ each blocking job of B_4 must be scheduled directly at its release date. Consequently, there is no room to delay the start of any leftbound job released before P_4 to this part. Due to the compatibilities, the rightbound blocking and dummy jobs released before P_4 are also forced to run before the start of P_4. Therefore, there are exactly $2|X| - |C|$ open slots within P_4 reserved for rightbound variable jobs. □

We proceed to explain the jobs released in the third part of our construction.

Jobs released in P_3. The third part (Figure 10) is responsible for the assignment of satisfying literals to each clause. During that part, we release a set of blocking jobs $B_3 = \{b_k \mid k = 0, \ldots, |C| - 1\}$ which contains one leftbound blocking job b_k for each clause c_k. Each blocking job B_k is released at time $A_3 + 2k$ and is compatible with each rightbound variable job that represents a variable assignment that satisfying the corresponding clause c_k. The gaps between the release times of the blocking jobs are filled with a set dummy jobs $H_3 = \{h^{\mathrm{r}}_k \mid k = 0, \ldots, |C| - 1\} \cup \{h^{\mathrm{l}}_k \mid k = 0, \ldots, |C| - 1\}$ containing one rightbound job h^{r}_k and one leftbound job h^{l}_k with release date $A_3 + 2k + 1$ each. Each leftbound dummy job h^{l}_k is compatible with all rightbound variable jobs, furthermore each rightbound dummy job h^{r}_k is compatible with the three leftbound jobs released during the time interval $[r_{h^{\mathrm{r}}_k} - 1, r_{h^{\mathrm{r}}_k} + 1]$.

We are now in position to prove Lemma 15, i.e., in a schedule with makespan $C_{\max}$ the only jobs released before A_3 that can be scheduled in P_3 are one rightbound variable job for each clause such that the variable assignment satisfies the clause.

Proof (Proof of Lemma 15). By Lemma 16 all jobs released within P_3 must start before the end of P_3. Hence, each leftbound dummy and blocking job is forced to

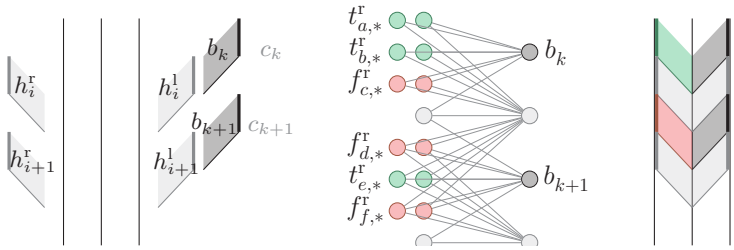


Fig. 10: Part P_3 for $c_k = (x_a \vee x_b \vee \bar{x}_c)$ and $c_{k+1} = (\bar{x}_d \vee x_e \vee \bar{x}_f)$. Note that each variable job can be adjacent with more than one clause job (although this does not occur in the example).

start at its release date. Therefore, due to the compatibilities, each rightbound dummy job must be scheduled directly when released. The only remaining $|C|$ free slots can be filled with rightbound variable jobs – exactly one free slot per clause c_k reserved for a variable job representing an assignment that satisfies c_k. □

We proceed to explain the jobs released in parts P_1 and P_2.

Jobs released in P_1. The first two parts are responsible for obtaining a correct assignment of the variables. In the first part, we release different types of jobs for each variable x_i, $i = 0, \ldots, |X| - 1$, cf. Figure 11 with the following. For each variable x_i, $i = 0, \ldots, |X| - 1$, we release

- two rightbound true variable jobs $t_{i,1}^{\mathrm{r}}$, $t_{i,2}^{\mathrm{r}}$ at times $6i$ and $6i + 1$, respectively,
- two rightbound false variable jobs $f_{i,1}^{\mathrm{r}}$, $f_{i,2}^{\mathrm{r}}$ at times $6i+3$ and $6i+4$, respectively,
- one leftbound true variable job t_i^{l} at time $6i + 4$,
- one leftbound false variable job f_i^{l} at time $6i + 1$,
- two leftbound indefinite variable jobs q_i^{t}, q_i^{f} at times $6i+1$ and $6i+4$, respectively.
- two leftbound blocking jobs b_i^{t}, b_i^{f} at times $6i$ and $6i + 3$, respectively.
- two leftbound dummy jobs h_i^{lt}, h_i^{lf} at times $6i + 2$ and $6i + 5$, respectively.
- two rightbound dummy jobs h_i^{rt}, h_i^{rf} at times $6i + 2$ and $6i + 5$, respectively.

In the following, we write $T^{\mathrm{r}} = \{t_{i,1}^{\mathrm{r}}, t_{i,2}^{\mathrm{r}} \mid x_i \in X\}$ for the set of rightbound true variable jobs, $F^{\mathrm{r}} = \{f_{i,1}^{\mathrm{r}}, f_{i,2}^{\mathrm{r}} \mid x_i \in X\}$ for the set of rightbound false variable jobs and $Q = \{q_i^{\mathrm{t}}, q_i^{\mathrm{f}} \mid x_i \in X\}$ for the set of indefinite jobs.

The compatibility graph G_1 is defined such that

- each blocking job b_i^{t} is compatible with the corresponding true variable jobs $t_{i,1}^{\mathrm{r}}$ and $t_{i,2}^{\mathrm{r}}$,
- each blocking job b_i^{f} is compatible with with the corresponding false variable jobs $f_{i,1}^{\mathrm{r}}$ and $f_{i,2}^{\mathrm{r}}$,
- each indefinite job q_i^{t} is compatible with the corresponding rightbound true variable jobs $t_{i,1}^{\mathrm{r}}$ and $t_{i,2}^{\mathrm{r}}$
- each indefinite job q_i^{f} is compatible with the corresponding rightbound false variable jobs $f_{i,1}^{\mathrm{r}}$ and $f_{i,2}^{\mathrm{r}}$.
- each dummy job h is compatible with the opposed jobs released in $[r_h - 1, r_h + 1]$,
- none of the remaining pairs of jobs are compatible.

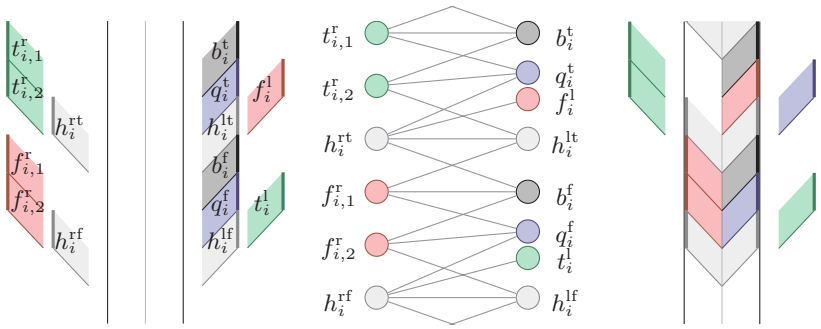


Fig. 11: Released jobs per variable x_i in P_1, the corresponding compatibilities given by G_1 and a scheduled example for a true variable assignment.

Jobs released in P_2. In the second part (Figure 12), there is room for exactly one indefinite job and one leftbound variable job per variable. This is realized by a set of rightbound blocking jobs $B_2 = \{b_{i,1}, b_{i,2} \mid x_i \in X\}$ where each blocking job $b_{i,1}$ is released at time $A_2 + 4i$ and is compatible with the corresponding two indefinite jobs q^{t}_i and q^{f}_i. Each blocking job $b_{i,2}$ is released at time $A_2 + 4i + 2$ and is compatible with the corresponding two leftbound variable jobs f^{l}_i and t^{l}_i. The gaps between two subsequent released blocking jobs are closed in both directions by dummy jobs $H_2 = \{h^{\mathrm{r}}_{i,1}, h^{\mathrm{l}}_{i,1} \mid x_i \in X\} \cup \{h^{\mathrm{r}}_{i,2}, h^{\mathrm{l}}_{i,2} \mid x_i \in X\}$ released at times $A_2 + 4i + 1$ and $A_2 + 4i + 3$, respectively. Each dummy job is compatible with all jobs of $Q, T^{\mathrm{l}}, F^{\mathrm{l}}$, or B_2 and the corresponding opposed dummy job released concurrently.

We are now in position to prove Lemma 14.

Proof (Proof of Lemma 14). By Lemmas 15 and 16, each rightbound dummy and blocking job of H_2 and B_2 must be scheduled before the end of P_2 and hence, directly at its release. By the given compatibilities this is also true for the leftbound dummy jobs of H_2. Therefore, there are exactly two open slots per variable x_i, one reserved for the two corresponding indefinite jobs $q^{\mathrm{t}}_i, q^{\mathrm{f}}$ and one for the two corresponding leftbound variable jobs $f^{\mathrm{l}}_i, t^{\mathrm{l}}_i$. Since no further space is left, for both pairs exactly one can be scheduled within P_2. The remaining one must be completed already by the end of P_1.

Also, for the first part, we can conclude that no blocking and no dummy job released in P_1 can start after the end of P_1. Consider now one variable x_i and assume that no job corresponding to x_i can start within part P_1 after $6i + 5$. This assumption holds obviously for x_n. Then, h^{rf}_i and h^{lf}_i, the latest released jobs corresponding to x_i, must both start at their release.

If the leftbound job t^{l}_i is scheduled within part P_1 it must be scheduled at its release and hence $f^{\mathrm{r}}_{i,1}$ and $f^{\mathrm{r}}_{i,2}$ must be postponed to the next parts. In this case, also the second blocking job b^{f}_i as well as the first two dummy jobs h^{rt}_i and h^{lt}_i are forced to start at their release, consequently also b^{t}_i. In this case it is not possible anymore to schedule q^{f}_i within part P_1. For this reason, the counter part q^{t}_i must be scheduled at its release time and the leftbound f^{l}_i must be postponed. With this, there is exactly one free slot for $t^{\mathrm{r}}_{i,2}$ and one for $t^{\mathrm{r}}_{i,1}$.

If, on the other hand, the leftbound job t^{l}_i is scheduled after part P_1, we have to schedule f^{l}_i within part P_1. Due to the conflicts with h^{rf}_i, the start time of f^{l}_i and the blocking and dummy jobs in between must in particular be scheduled at

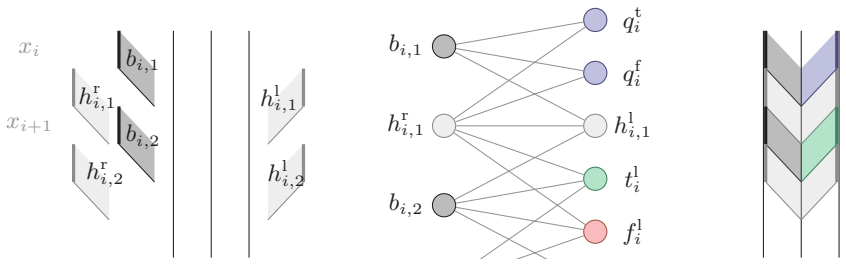


Fig. 12: Part P_2 creates a structure of blocking and dummy jobs with respective compatibilities that create space for exactly one indefinite job per variable x_i.

their release. For that reason q^{t}_i must be postponed and q^{f}_i must be scheduled at its release. Hence, also the rightbound true jobs t^{r}_i and t^{r}_i must be postponed and there are exactly two slots for the two false jobs.

In both cases, the scheduled leftbound jobs ensure that no earlier released variable job can start after $6(i-1)+5$. Hence, it can be concluded by induction that, for each variable, either all corresponding false jobs or all corresponding true jobs must be scheduled after part P_1. And since, by Lemmas 15 and 16, at least $2n$ rightbound variable jobs must be scheduled within P_1 the free spots ensure that exactly the two counter parts are scheduled within P_1. □

We can conclude the following claim and hence, Theorem 7.

Claim. There is a satisfying assignment for the given $(\leq 3, 3) - \textsc{Sat}$ instance if and only if there is a feasible schedule for the constructed scheduling instance with makespan $C_{\max} = A_5 + 1$.

Proof (Proof of Theorem 7). If there is a schedule with makespan $C_{\max}$ we can apply Lemmas 14 to 16. Within the resulting schedule we can therefore be sure that $|C|$ rightbound variable jobs are scheduled within the clause part. Since by Lemma 14 the assignment of each variable is well defined we get by Lemma 15 a satisfying truth assignment for the clauses.

If on the other hand a satisfying truth assignment is given, the described schedule with demanded makespan can be created in straight-forward manner, by postponing the assignment jobs corresponding to the truth assignment and scheduling all other jobs within the part they are released in (or in part P_2 in the case of leftbound variable jobs or indefinite jobs). □

C.2 NP-Hardness of Total Completion Time Minimization

Theorem 3. *The bidirectional scheduling problem on a single segment and with an arbitrary compatibility graph is* NP*-hard even if* $p_j = \tau_1 = 1$ *for each* $j \in J$.

We give an analogous reduction as for Theorem 7. Note, that solutions optimal for the total completion time and those optimal for the total waiting time are equivalent. Hence, it is sufficient to prove the hardness for the latter. The goal is to enforce the same structure as for makespan minimization when minimizing the total waiting time. To do so, we start by calculating an upper bound of the resulting waiting time.

We can trivially bound the total waiting time of a schedule that achieves a makespan of $C_{\max}$ by $W = |J| \cdot C_{\max} = |J| \cdot (A_5 + 1)$, where J is the set of all jobs in

our construction. With this polynomial bound we can extend the construction of a scheduling instance for a given $(\leq 3, 3)$-Sat instance by part P_5 with $W+1$ further leftbound blocking jobs $B_5 = \{b_i \mid i = 0, \ldots, W-1\}$ with release date $A_5 + i + 1$ for each $b_i^5 \in B_5$ that are not compatible to any of the previous jobs.

Claim. There is a satisfying truth assignment for the given $(\leq 3, 3)$-Sat instance if and only if there is a feasible schedule for the constructed scheduling instance with total waiting time of at most W.

Proof (Proof of Theorem 3). Assume first that there is a satisfying assignment for the $(\leq 3, 3)$-Sat instance. In this case, there is a schedule where no job released in the first four parts starts processing after A_5 and hence the resulting total waiting time does not exceed W.

Assume on the other hand, that there is a solution for the constructed scheduling instance whose objective does not exceed W. For such a solution, either all jobs released in the first four parts start before A_5 or their is at least one starting later. In the first case, we get, by Lemmas 16 to 14, a schedule together with a satisfying truth assignment with waiting time bounded by W.

In the second case each postponed job j with starting time S'_j increases the already existing waiting time by at least an amount of $(S'_j - A_5) + W + 1 - (S'_j - A_5) = W + 1$. Hence, the first case applies. □

C.3 APX-Hardness of Makespan Minimization

In this section, we show the APX-hardness of bidirectional scheduling. As for the NP-hardness proof, it is convenient to first prove the APX-hardness for minimizing the makespan before turning to the minimization of the total completion time.

Theorem 8. *Minimizing the makespan for $m = 1$ with an arbitrary compatibility graph G_1 is* APX*-hard even if $p_j = \tau_1 = 1$ for each $j \in J$.*

Proof. We reduce from a specific variant of Max-3-Sat which is NP-hard to approximate to within a factor of 1016/1015, see Berman et al. [3]. An instance of Symm-4-Occ-Max-3-Sat is given by a Boolean formula with a set C of clauses of size three over a set of variables X, where both the positive and the negative literal of each variable $x_i \in X$ appears in exactly two clauses. Berman et al. [3] construct a family of instances of Symm-4-Occ-Max-3-Sat with $1016n$ clauses, where $n \in \mathbb{N}$. They show that for any $\delta \in (0, 1/2)$, it is NP-hard to distinguish between the "bad" instances where at most $(1015 + \delta)n$ clauses can be satisfied and the "good" instances where at least $(1016 - \delta)n$ instances can be satisfied.

Let ϕ be a formula of the above family. Based on ϕ, we use the same construction as in Theorem 7 with one small adaption: In the first part, for each variable x_i, $i = 0, \ldots, |X| - 1$, we release additionally two virtual jobs $v_{i,1}$ and $v_{i,2}$ at times $6i$ and $6i + 1$, respectively. Both jobs are compatible with all leftbound blocking, dummy and variable jobs of the same variable. We claim that for this bidirectional scheduling instance the optimal makespan is $12|X| + |C| + 1 + \tilde{c}$ if and only if the minimum number of unsatisfied clauses of ϕ is $\tilde{c}$. Assuming the correctness of the claim, we derive that for a good instance with $1016n$ clauses, the makespan is at most $12|X| + 1016n + 1 + \delta n$. Using the identity $|X| = 3|C|/4 = 3 \cdot 1016n/4$, we can bound the makespan from above by $(10160 + \delta)n + 1$. For bad instances, on the other hand, the makespan is at least $(10161 - \delta)n$, i.e., the optimal makespan cannot be approximated by a factor of $10161/10160 \approx 1.000098$.

It is left to prove the correctness of the claim. It is easy to see that the optimal makespan is bounded from above by $12|X| + |C| + 1 + \tilde{c}$ by a small adaption of

the arguments of the proof of Theorem 7. To see this, fix a variable assignment satisfying all but $\tilde{c}$ clauses. In parts one and two (where the variable assignments are fixed) we schedule all jobs as in the proof of Theorem 7 with respect to the variable assignment. Additionally, the leftbound variable jobs not scheduled in the first part, leave a gap in the schedule that is a perfect fit for the additional virtual jobs, see also the right illustration in Figure 11. In the third part, we schedule one satisfying variable for each clause that is satisfied. In the forth part, we schedule any $2|X| - |C|$ variable jobs left over from previous parts. By construction, at the end of the forth part, we are left with $\tilde{c}$ variable jobs (that could not be matched to any clause job in the third part). Scheduling them one after another, we obtain the claimed makespan of $12|X| + |C| + 1 + \tilde{c}$.

To see that $12|X| + |C| + 1 + \tilde{c}$ is lower bound on the optimal makespan, we argue using the concept of matched jobs. First, note that there is always an optimal schedule in which all jobs are processed at an integral point in time. Otherwise, we could move the first job scheduled at a non-integral point in time to the previous integral point in time without violating any constraints. Iterating this process, we obtain a schedule in which all jobs are processed at integral times, as claimed. Given such an integral schedule, we call a job processed at time t *matched*, if it is leftbound and there is another rightbound job processed at time t, or vice versa. Otherwise the job is called unmatched.

For the following arguments, fix an integral schedule. We proceed to argue that there are at least $\tilde{c}$ unmatched jobs that are mutually incompatible.

First consider the (clause) blocking jobs released in part three. For $k, l \in \{0, 1, 2\}$, let $X_{k,l}$ be the set of variables x_i such that k rightbound true variable jobs $t^{\mathrm{r}}_{i,1}$, $t^{\mathrm{r}}_{i,2}$ are matched to a (clause) blocking job and l rightbound false variable jobs $f^{\mathrm{r}}_{i,1}$, $f^{\mathrm{r}}_{i,2}$ are matched to a (clause) blocking job. Intuitively, the sets $X_{1,1}, X_{2,1}, X_{1,2}, X_{2,2}$ contain the variables that are not set consistently according to a well-defined truth assignment. Using that at most $|C| - \tilde{c}$ clauses of ϕ can be satisfied, we derive that at least

$$\tilde{c} - |X_{1,1}| - |X_{2,1}| - |X_{1,2}| - 2|X_{2,2}| \tag{2}$$

(clause) blocking jobs (or rightbound dummy jobs) are unmatched.

For any variable $x_i \in X_{2,2}$, the leftbound blocking jobs b^{t}_i, b^{f}_i, dummy jobs h^{lt}_i, h^{lf}_i, indefinite jobs q^{t}_i, q^{f}_i, and variable jobs f^{l}_i, t^{r}_i are matched by at most the two rightbound dummy jobs h^{rt}_i, h^{rt}_i and the two virtual jobs $v_{i,1}$, $v_{i,2}$ released in part 1 as well as the two blocking jobs $b_{i,1}$, $b_{i,2}$ released in part 2, so that in the end, at least two rightbound jobs are left unmatched. Equivalently, for any variable $x_i \in X_{1,2} \cup X_{2,1}$ at least one of the rightbound jobs above is left unmatched.

For any variable $x_i \in X_{1,1}$, consider the leftbound variable jobs f^{l}_i and t^{l}_i as well as the leftbound indefinite jobs q^{t}_i and q^{f}_i. At most one indefinite job and one variable job most can be matched with the blocking jobs $b_{i,1}$ and $b_{i,2}$ released in part 2. The other two jobs, say the true variable job t^{l}_i and the indefinite job q^{t}_i, are only compatible with the rightbound variable jobs, the virtual jobs and the dummy jobs jobs, leaving at least one job unmatched. Using (2), we may conclude that the total number of unmatched jobs is at least $\tilde{c}$.

As argued above, the unmatched jobs are either (clause) blocking jobs released in part three or remainders of the different types of leftbound jobs associated with variables and released in the first part. As none of them are compatible, the makespan is at least $12|X| + |C| + 1 + \tilde{c}$, as claimed. □

We are now ready to prove the APX-hardness of the minimization of the total completion time.

Theorem 4. *The bidirectional scheduling problem on a single segment and with an arbitrary compatibility graph is* APX*-hard even if* $p_j = \tau_1 = 1$ *for each* $j \in J$.

Proof (Sketch). Let ϕ be a formula with $1016n$ clauses for some $n \in \mathbb{N}$ with $\tilde{c}$ unsatisfiable clauses, as in Berman et al. [3] (cf. proof of Theorem 8). We use a similar idea as in the proof of Theorem 3, i.e., we use the same construction as in the reduction for the makespan but add an additional set of M leftbound blocking jobs $B_5 = \{b_i | i = 0, \ldots, M-1\}$ with release date $M = 12|X| + |C| + 1 = 10160n + 1$. With similar arguments as before, we can show that there is an optimum schedule in which exactly $\tilde{c}$ (clause) jobs are unmatched before time M, with only exactly $\tilde{c}$ incompatible variable jobs remaining unscheduled after time $2M$. The sum of completion times of this schedule is $an^2 + bn\tilde{c} + \tilde{c}(\tilde{c}+1)/2 + \mathcal{O}(n)$ for some constants $a, b \in \mathbb{N}$.

Now consider a "good" instance with at most δn unsatisfiable clauses. The optimum schedule has a sum of completion times of at most

$$an^2 + b\delta n^2 + n^2\delta^2/2 + \mathcal{O}(n).$$

On the other hand, a "bad" instance with at least $(1-\delta)n$ unsatisfiable clauses leads to a sum of completion times of at least

$$an^2 + b(1-\delta)n^2 + n^2(1-\delta)^2/2 + \mathcal{O}(n).$$

Since, for $n \to \infty$, good and bad instance cannot be distinguished in polynomial time unless $\mathsf{P} = \mathsf{NP}$ (cf. [3]), no algorithm can approximate the sum of completion times by a factor better than

$$\frac{a + b(1-\delta) + (1-\delta)^2}{a + b\delta + \delta^2} \underset{\delta \to 0}{\to} \frac{a+b+1}{a},$$

which is constant. □