Scheduling parallel batching machines in a sequence

Journal of Scheduling (2019) 22:335–357
https://doi.org/10.1007/s10951-018-0560-6

Scheduling parallel batching machines in a sequence

Ward Passchyn[1,2] · Frits C. R. Spieksma[1,3]

Published online: 21 March 2018

Abstract
Motivated by the application of scheduling a sequence of locks along a waterway, we consider a scheduling problem where multiple parallel batching machines are arranged in a sequence and process jobs that travel along this sequence. We investigate the computational complexity of this problem. More specifically, we show that minimizing the sum of completion times is strongly NP-hard, even for two identical machines and when all jobs travel in the same direction. A second NP-hardness result is obtained for a different special case where jobs all travel at an identical speed. Additionally, we introduce a class of so-called synchronized schedules and investigate special cases where the existence of an optimum solution which is synchronized can be guaranteed. Finally, we reinforce the claim that bidirectional travel contributes fundamentally to the computational complexity of this problem by describing a polynomial time procedure for a setting with identical machines and where all jobs travel in the same direction at equal speed.

Keywords Machine scheduling · Complexity · Parallel batching machine · Machine sequence

1 Introduction

Consider the following problem. Given is a set of $M \equiv \{1, 2, \ldots, m\}$ linearly ordered machines, each machine $i \in M$ being located at a given position x_i. Distances between machines follow from their position, e.g. the distance between machines $i, k \in M$ equals $|x_i - x_k|$. Each machine $i \in M$ is a parallel batching machine with capacity B_i, i.e. each machine is capable of processing up to B_i jobs simultaneously, processing a set of jobs takes T_i time units. Also given is a set of jobs $J \equiv \{1, 2, \ldots, n\}$; each job $j \in J$ is characterized by a release time r_j, a travelling speed v_j, a starting machine $s_j \in M$, and an ending machine $e_j \in M$. In addition, if for some job $j \in J$, $s_j = e_j$, then a parameter called direction is specified, with $\text{dir}_j \in \{\text{left}, \text{right}\}$. If, for job $j \in J$, $s_j < e_j$, job j is called right-travelling. If, for job $j \in J$, $s_j > e_j$, job j is called left-travelling. For the jobs where $s_j = e_j$, the direction 'left' or 'right' specifies whether the job is left-travelling or right-travelling, respectively. It follows that the set of jobs can be partitioned into two subsets R and L containing all right-travelling jobs and all left-travelling jobs, respectively. Clearly, we have $J = L \cup R$ and $L \cap R = \emptyset$. A right-travelling job j must be processed by machines $s_j, s_j + 1, \ldots, e_j$ in that order, while a left-travelling job $j \in J$ must be processed by machines $s_j, s_j - 1, \ldots, e_j$ in that order. Table 1 specifies the input data for an example instance; the corresponding layout of machines and the trajectory of the jobs is visualized in Fig. 1. We say that a job $j \in J$ is completed when it has been processed by its ending machine e_j; we refer to this moment as the completion time C_j of job j. (Clearly, C_j is not a parameter in this problem, instead C_j follows from a solution.) Each job $j \in J$ travels with speed v_j between consecutive machines. The travel time required for a job j to travel between two consecutive machines i and i' thus equals $|x_{i'} - x_i|/v_j$. There is a restriction on the set of jobs that can be processed simultaneously by any of the machines: no two jobs of different direction can be in the same batch. In other words, each batch must consist of only right-travelling jobs, or only left-travelling jobs; for convenience, we, respectively,

A preliminary version of this work has been published as part of the first author's Ph.D. dissertation (Passchyn 2016).

✉ Ward Passchyn
ward.passchyn@kuleuven.be; wpasschyn@ompartners.com

Frits C. R. Spieksma
f.c.r.spieksma@tue.nl

1 Faculty of Economics and Business, ORSTAT, KU Leuven, Leuven, Belgium

2 Present Address: OM Partners, Koralenhoeve 23, Wommelgem, Belgium

3 Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

Table 1 Input data for an example instance

Jobs						Machines			
j	r_j	s_j	e_j	dir_j	v_j	i	T_i	B_i	x_i
1	0	3	1	Left	12	1	1	1	0
2	1	3	2	Left	24	2	2	2	24
3	2	1	2	Right	4	3	2	2	72
4	3	1	1	Right	12				
5	4	3	3	Left	12				
6	4	1	3	Right	12				

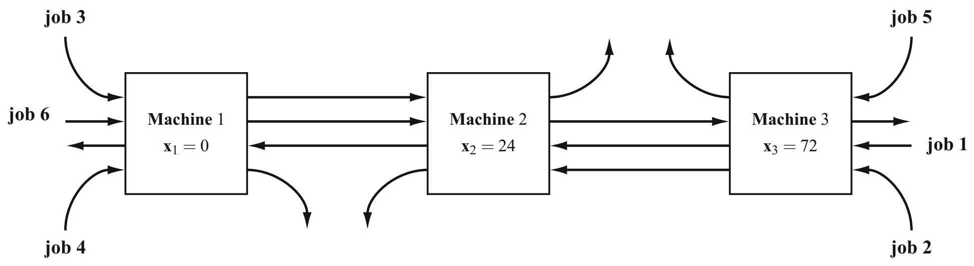


Fig. 1 Visualization of the machine layout and job trajectories for the instance described in Table 1

refer to such batches as right-travelling or left-travelling. Finally, once a machine has processed a set of jobs, it is immediately available to process a set of jobs travelling in the opposite direction. However, two batches containing jobs of the same direction must be separated by at least T_i time units. More precisely, for any machine $i \in M$, when considering two batches processing jobs with the same direction, their starting times should be at least $2T_i$ time units apart. We call this property the *separation* property; the absence of this property pertains to the situation where each machine is directly available to process any batch after a previous batch has finished processing.

Our goal is to find a feasible schedule that minimizes total flow time. We say that a job waits, or incurs waiting time, if it is not being processed by a machine, nor travelling between two machines. Since the values r_j, v_j, and the distance between the machines are fixed, it is clear that minimizing the total flow time is equivalent to minimizing the total waiting time and to minimizing the sum of completion times.

More precisely, we must specify

- for each machine at which moments in time the machine must start processing a batch, and
- the composition of each batch.

These moments in time, as well as the composition of the batches, must be such that each job $j \in J$ is processed by machines $s_j, \ldots, e_j$ in that order, and must be consistent with the given speeds and release times of the jobs; finally, $\sum_{j \in J}(C_j - r_j)$ should be minimum. We will refer to this problem as SPBM. The question in the corresponding decision-variant of SPBM asks whether there exists a solution with a total waiting time no more than a predefined value W; we refer to this decision problem as dec-SPBM. In case all jobs travel in the same direction, the resulting special case is called uni-directional SPBM.

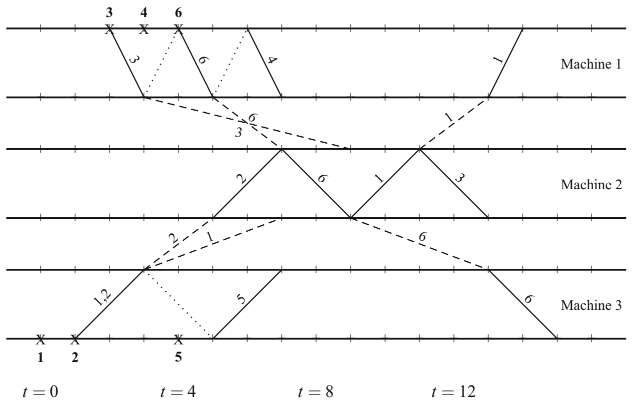


Fig. 2 Feasible solution and visualization for the instance of Table 1

When categorizing the SPBM problem, it becomes clear that this is not a ‘clean’ flow shop problem. Although there is quite a bit of structure in the set of machines that a job must visit, it is not true that each job visits the same set of machines in the same order. Even in the uni-directional case, not every job needs to be processed by the same machines; this remains true in the case of only two machines.

We represent an instance and a solution as illustrated in Fig. 2, which shows a solution for the instance specified in Table 1. Such representations, referred to as time-distance diagrams, are commonly used in the context of transportation, for example to track the movement of trains. As will be discussed in Sect. 1.1, our practical motivation underlying the SPBM problem also stems from transportation scheduling. In the figure, time passes from left to right and the vertical axis denotes the progress of a job j along its trajectory from s_j to e_j. Note that each machine, although located at a single coordinate, is represented by two separated horizontal lines, so that the processing of a batch on a machine i can be visualized as a tilted line where the horizontal component has a length equal to the processing time T_i. Each release time of a job is marked with an 'X'. The dashed lines correspond to a job travelling in between the machines, whereas solid lines correspond to the processing on the machines. Note that the dashed lines may intersect, i.e. jobs may overtake each other in between machines due to a difference in speed. On a machine, however, the batches may not overlap, although a batch may consist of multiple jobs on the condition that the machine capacity B_i is not exceeded. Further, the dotted lines correspond to the idle time interval required by the separation property in the event that a machine processes two consecutive batches containing jobs that travel in the same direction.

In Fig. 2, lines are labelled with the job to which they correspond. Recall that specifying a solution entails specifying the composition of each batch. Indeed, given the starting times of all batches and their corresponding direction, different solutions may still result from different assignments of jobs to these batches. One straightforward way of obtaining such an assignment is to consider the jobs in the order of their release time and to assign each job j, at each machine m, to the first batch following the time of arrival of job j at machine m for which the capacity bound B_m is not exceeded. Clearly, we can restrict ourselves to solutions where each job, immediately upon being processed by a machine, starts to travel immediately towards the next machine in its trajectory. In what follows, we will omit the labels whenever this straightforward assignment is implied. Note, however, that this assignment need not be optimal in general. Indeed, in the solution shown in Fig. 2, job 6 is processed on machine 1 before job 4; it can be verified that processing job 4 first, as would be the case with the straightforward assignment, yields a solution with a larger total waiting time.

Table 2 summarizes the completion time, flow time, and waiting time for the solution shown in Fig. 2. Recall that these three objective functions are equivalent since they differ by a constant.

1.1 Motivation and related literature

The introduction above phrases, in machine scheduling terminology, the situation that arises when operating a series of locks along a river or a canal. Figure 3 shows a single lock on a canal. Indeed, in such a situation, ships (the jobs) travel in one of two directions while locks (the machines) are present to control the water level of the waterway. A lock can transfer a set of ships simultaneously, but only when all ships in this set travel in the same direction; the time this operation takes is called the lockage time (T_i). This situation may reflect so-called staircase locks, consisting of multiple locks in immediate succession, i.e. where the travel distance between adjacent locks is equal to zero. Such a setting occurs, for example at the Three Gorges Dam in China and at Caen Hill in the United Kingdom, consisting of, respectively, 5 and 16 successive locks. With nonzero travel distance, however, the situation may also reflect an entire waterway where multiple locks are present over a longer distance. Several waterways of major economical importance in fact feature a more general setting where, in addition to the properties of the SPBM problem outlined above, each lock consists of multiple parallel chambers. We mention, for example the Kiel Canal and Danube river in Europe and the Panama Canal in Middle America. The SPBM problem clearly underlies the more general scheduling problem for these waterways. However, situations that correspond to SPBM, i.e. where each lock consists of a single chamber, also occur on several important rivers and waterways.

Table 2 Summary of the objective value for the solution of Fig. 2

j	Completion time	Flow time	Waiting time
1	14	14	3
2	7	6	0
3	13	11	2
4	7	4	3
5	7	3	1
6	15	11	0
Total	63	49	9

Fig. 3 Example of a single canal lock, corresponding to a machine in SPBM. Photograph by A. Fin, Wikimedia Commons. (In the public domain)

Different methods to solve the scheduling problem for locks in sequence have been proposed in the literature. Petersen and Taylor (1988) describe a problem setting for a sequence of 8 locks on the Welland Canal where batch processing is not possible, i.e. where $B_i = 1$ for all $i \in M$, and present a heuristic based on a dynamic programming algorithm for the single lock setting. Smith et al. (2009) considered a similar problem setting for a sequence of 29 locks on the Upper Mississippi River; they introduce a simulation model in order to evaluate the impact of operating policies and proposed infrastructural investments. Maximizing the throughput through a network of waterways is considered by Righini (2016), who models this setting as a multi-commodity flow problem while enforcing an upper bound on the throughput of locks, although making abstraction of the detailed lock operation schedules. Prandtstetter et al. (2015) give a formal definition for a problem involving the scheduling of a sequence of locks consisting of independent lock chambers and present a variable neighbourhood search procedure in order to obtain heuristic solutions.

While exact methods for the scheduling of a single lock have been proposed, see, e.g. Passchyn et al. (2016b), Verstichel et al. (2013), Smith et al. (2011), literature on exact methods for the integrated scheduling of a sequence of locks has remained scarce. Verstichel and Vanden Berghe (2016) identify this problem setting as an interesting direction for additional research, and mention the concept of so-called green waves, which can be seen as similar to the operation of traffic lights along important highways. In such an approach, the goal is to *synchronize* the operation of locks, meaning that each ship, upon exiting a lock, travels immediately towards the next lock and reaches it at a time where it is immediately available to start a lockage. Thus, in such a system, a ship incurs no waiting time after being served by the first lock it encounters. In Sect. 3, we give a rigorous treatment of synchronized solutions. Passchyn et al. (2016a) present and compare different mixed integer programming models that allow to minimize the total flow time for ships passing through locks while taking ship speed and emissions into account.

A related setting in the context of waterways is the scheduling of bidirectional traffic along a narrow river or canal, where a limited number of wider segments is available for the crossing of ships that travel in opposite directions or for the overtaking of ships that travel in the same direction. The complexity of the problem to minimize total waiting time for this setting is settled by Disser et al. (2015). We note that the results obtained by Disser et al. (2015) do not immediately extend to our setting for the scheduling of locks. One attempt to connect the two problems is to see the narrow canal sections in their problem setting as machines and the widened sections as the distance separating adjacent machines. A notable difference is then that in the SPBM setting, each machine must return to its initial position in between the processing of two jobs that travel in the same direction, i.e. the separation property must be satisfied. A narrow canal section, in contrast, is immediately available for jobs travelling in either direction. Another difference is that, in the SPBM setting, each job is allowed to overtake any other job, while both jobs are travelling.

A different problem related to SPBM concerns the flow of communication. Antoniadis et al. (2014) describe a problem concerning packet forwarding along a line network; in this setting, batch processing and the separation property are absent, and all packets travel from left to right. An online problem setting is considered and, for the objectives of minimizing total flow time and minimizing maximum flow time, a competitive analysis indicates bounds on the performance of different packet forwarding policies, with and without speed augmentation. Adler et al. (1998) consider the scheduling of messages to be routed across a line network. They show that maximizing the number of on-time messages is NP-hard. We point out that a notable difference with SPBM lies in the assumption that a machine can, in the network setting, simultaneously process jobs travelling in opposite directions. Further, the existence of deadlines may result in packets being dropped once their deadline is exceeded.

SPBM also resembles the well-known flow shop problem. A paper that is intimately related to a special case of SPBM is Brucker et al. (2004). They consider a traditional flow shop with two machines and arbitrary transportation times for a job. This situation is quite close to the special case of SPBM if we take (i) the uni-directional special case, (ii) $m = 2$, (iii) $B_1 = B_2 = 1$, and (iv) $s_j = 1, e_j = 2$ for each $j \in J$. However, due to the absence of the separation property in Brucker et al. (2004), it is not clear how the hardness of their flow shop problem carries over to SPBM. Moreover, we are interested in machines that are parallel batching machines, i.e. machines that have capacity $B_i > 1$.

A large body of related literature also exists on the scheduling of a single-track railway line with a limited number of segments or stations where overtaking is allowed. An early paper on this topic was presented by Frank (1966); since then, many papers have studied integer programming models and heuristics for this problem setting. While the single-track railway scheduling problem shows similarities to SPBM, a number of differences appear. First, a machine may serve multiple jobs simultaneously. Secondly, in SPBM, travel time follows from the distance between machines; in the railway setting, such a section corresponds to a single-track segment, which requires a train to stand still and incur waiting time in order to allow overtaking. Gafarov et al. (2015) prove NP-hardness for such a single-track railway problem. In their problem setting, each of the trains has a due date, so that the overtaking of trains may in fact be required, and the hardness result follows in part from the underlying sequencing problem.

Results on the generalization of the flow shop problem to a bidirectional processing order are scarce. One setting where this variant is considered is presented by Zhao et al. (2009); they consider a sequence of operations for a container loading and unloading problem. A bidirectional flow shop problem with additional constraints on the processing order is described, and the authors present integer programming formulations as well as a heuristic procedure based on a relaxation of the original problem.

1.2 Our results

SPBM is, in its generality, a difficult problem to solve. We analyse the complexity of SPBM for various special cases. The following options are considered: an arbitrary number of machines versus two machines, uni-directional versus bidirectional, and arbitrary speeds versus identical speeds. Our results are the following:

– We prove that SPBM is NP-hard even for $m = 2$ and in the uni-directional case where all jobs are processed by both machines. This result can be seen as the analogue of the result by Brucker et al. (2004) with unbounded capacity and with enforcement of the separation property. Further, we show that SPBM is NP-hard in the bidirectional setting, even if $v_j = v$ for all $j \in J$ (Sect. 2).
– We discuss a class of solutions which satisfy particular properties: so-called synchronized solutions. For a number of special cases, we investigate whether the existence of an optimum synchronized solution can be guaranteed. (Sect. 3)
– Finally, we prove that the uni-directional SPBM with $v_j = v$ for all $j \in J$ and with a common machine can be solved in polynomial time (Sect. 4).

2 Hardness results

We prove that SPBM is strongly NP-hard. In fact, we provide a reduction that implies strong NP-hardness for a more restricted uni-directional setting. The precise result is as follows.

Theorem 1 *Problem dec-SPBM is strongly NP-complete, even for two identical machines, with only right-travelling jobs, and when each job must be processed by both machines.*

The proof consists of a reduction starting from MAX CUT. For a detailed description of the proof, we refer to 'Appendix A'.

A different hardness result can be obtained for a special case of SPBM where all jobs have the same speed. Note that, in contrast to the setting covered by Theorem 1, uni-directional travel is not assumed in this setting. The precise result is the following.

Theorem 2 *Problem dec-SPBM is strongly NP-complete, even for jobs with equal speed and identical machines with unbounded capacity.*

For the proof, which also consists of a reduction from MAX CUT but which requires a significantly more involved construction, we refer to 'Appendix B'.

The results above indicate that finding an optimum solution for larger instances of the general SPBM problem is likely unrealistic. This, however, need not prevent the existence of polynomial time algorithms for special cases of the general SPBM problem. In the following section, we identify schedules with a specific structure that may guide the search for such algorithms. In Sect. 4, this structure is then used to present a polynomial time algorithm for a uni-directional special case.

3 Synchronized solutions

In this section, we introduce a class of solutions that posses a specific property: so-called *synchronized* solutions. We discuss a special case of SPBM involving two machines and show that in this special case the existence of an optimal solution which is synchronized is guaranteed. We also show by means of different examples that the existence of an optimum solution which is synchronized does not extend to generalizations of this two-machine setting.

We begin by stating the definition of a synchronized solution.

Definition 1 A solution to SPBM is *synchronized* if each job $j \in J$ only incurs waiting time before being processed in a batch by its arrival machine s_j.

Observe that if a solution is synchronized, it follows that each job $j \in R$ (respectively, $j \in L$), after having been served by a machine $m < e_j$ (respectively, $m > e_j$), travels to machine $m+1$ (respectively, $m-1$) and immediately enters a batch starting on machine $m+1$ ($m-1$) without incurring any waiting time. Figure 4 illustrates this definition; the input data corresponding to the instance shown in the figure are listed in Table 3.

Note that synchronized solutions correspond to so-called green waves in traffic scheduling; for example, when scheduling a series of traffic lights along an important road, it makes sense to adjust the timing of the lights so that once a car meets a first green light, it can keep travelling at the indicated maximum speed and arrive at all following traffic lights without encountering any red lights. When considering our bidirectional setting, a crucial difference with the scheduling

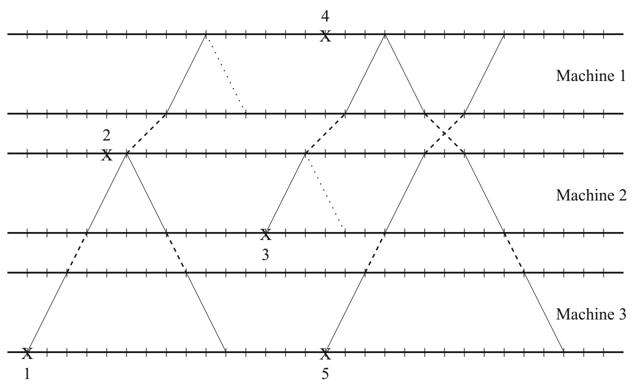


Fig. 4 A synchronized solution to the instance described in Table 3. The dotted lines correspond to the separation property

Table 3 Input data for an instance illustrating synchronized solutions

Jobs					Machines			
j	r_j	dir$_j$	s_j	e_j	i	T_i	B_i	x_i
1	0	Left	3	1	1	2	∞	0
2	4	Right	2	3	2	2	∞	2
3	12	Left	2	1	3	2	∞	3
4	15	Right	1	3				
5	15	Left	3	1				

of traffic lights appears: a green light simultaneously serves cars going in either direction, whereas a machine cannot.

3.1 Synchronized solutions for two machines

We now consider a special case of SPBM featuring two identical machines with processing time T. The travel distance between the machines is equal to zero (or, equivalently, the travel time is considered negligible) and each job must pass both machines. We consider the bidirectional setting. Figure 5 shows an instance of this special case, and a feasible synchronized solution illustrating this special case of SPBM. The solution shown in Fig. 5, with a total waiting time of 4 time units, is optimum. In fact, we show in this section that, for this problem setting, there always exists an optimum solution which is synchronized. Recall that in a synchronized solution, by definition, a job $j \in J$ incurs waiting time only before entering its arrival machine s_j.

Theorem 3 *For each instance of SPBM consisting of two identical machines where the travel distance equals zero and where all jobs must be served by both machines, there exists an optimum solution which is synchronized.*

Proof We prove the theorem by arguing that an arbitrary optimum solution $\mathcal{O}$ can be transformed into a synchronized solution $\mathcal{O}'$ without increasing the total waiting time. Let $\mathcal{I}$

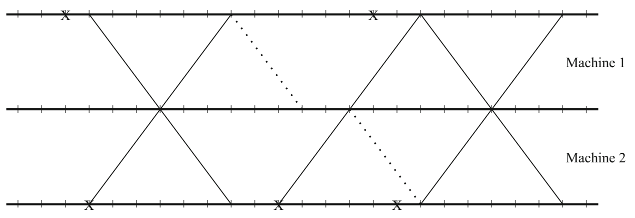


Fig. 5 An example of a synchronized schedule featuring two identical machines. The distance between the machines is equal to zero. The dotted lines correspond to the separation property

be an instance of the stated problem. We call the machines in this instance machine 1 and machine 2, with machine 2 positioned to the right of machine 1. Further, we index the batches scheduled on machine 1 by their starting time.

Consider an arbitrary optimum solution $\mathcal{O}$. Clearly, in an optimum solution, each non-empty right-travelling (left-travelling) batch is first processed by machine 1 (2), and then by machine 2 (1). For convenience, we restrict ourselves to solutions where all odd-numbered batches on machine 1 are right-travelling, all even-numbered batches on machine 1 are left-travelling, and where every idle period between consecutive batches on machine 1 has a duration strictly smaller than $2T$. Notice that any feasible solution is easily modified so that it satisfies these requirements, without increasing the total waiting time. Indeed, empty batches can be scheduled to start where needed since the separation property enforces that a machine remains idle for at least T time units in between subsequent batches that process jobs travelling in the same direction.

Now consider all odd-numbered batches i of machine 1 and their subsequent batch $i+1$. Let t_i and t_{i+1} be the respective starting time of these batches. Clearly, $t_{i+1}-(t_i+T) \geq 0$, since batch $i+1$ cannot start earlier than the completion of batch i. To build the synchronized solution $\mathcal{O}'$, we specify the starting times for the corresponding batches on machine 2. We use t'_i (t'_{i+1}) to denote the starting time of the right-travelling (left-travelling) batch on machine 2 that processes the jobs in batch i ($i+1$). We distinguish the following cases for the starting times of batches i and $i+1$ in solution $\mathcal{O}$. Recall that the case where $t_{i+1} - (t_i + T) \geq 2T$ does not occur since machine 1 is never idle for a period of $2T$ time units.

– Case 1: $t_{i+1} - (t_i + T) = 0$. This situation is shown in Fig. 6. We schedule the following batches in solution $\mathcal{O}'$. On machine 1, we schedule a right-travelling batch starting at time t_i and a left-travelling batch starting at time t_{i+1}, i.e. we copy the batches i and $i+1$ from solution $\mathcal{O}$ to $\mathcal{O}'$. On machine 2, we schedule a left-travelling batch starting at time $t_{i+1} - T$ processing all jobs that are processed in batch $i+1$, and a right-travelling batch starting

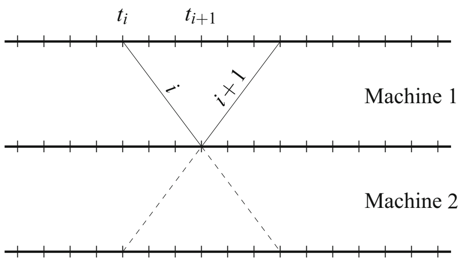


Fig. 6 Visualization of Case 1 in proving Theorem 3

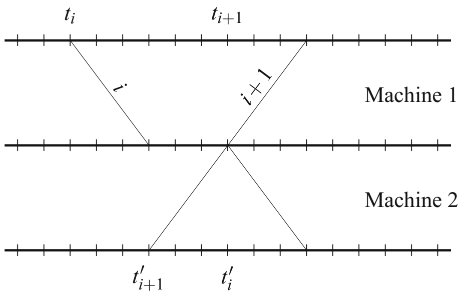


Fig. 7 Visualization of Case 2 in proving Theorem 3

at time $t_i + T$ processing all jobs that are processed in batch i. Notice that this does not occupy either machine outside of the interval $[t_i, t_{i+1} + T)$. Also notice that any job served by batch i in solution $\mathscr{O}'$ leaves machine 2 no later than it does in solution $\mathscr{O}$, and that any job served by batch $i + 1$ in solution $\mathscr{O}'$ leaves machine 1 no later than it does in solution $\mathscr{O}$.

– Case 2: $0 < t_{i+1} - (t_i + T) < 2T$. If either batch i or batch $i + 1$ is empty, we may reschedule it so that these two batches are consecutive without increasing the total waiting time. We can then schedule batches on machine 2 as described in Case 1 above. Assume, thus, that both i and $i+1$ are non-empty. The starting time of the batches on machine 2 satisfies $t'_{i+1} + T \le t_{i+1}$ and $t'_i \ge t_i + T$. Now, if $t'_i < t'_{i+1}$, it would mean that $t'_i \le t'_{i+1} - T$ since machine 2 cannot simultaneously process two batches. Then, the above inequalities imply that $t_i+T \le t_{i+1}-2T$ which signifies the existence of an idle period with a duration of at least $2T$, a contradiction. It follows that $t'_i \ge t'_{i+1}$, which actually means that $t'_i \ge t'_{i+1} + T$.
Consider now the moment $t = \max(t_i + T, t'_{i+1} + T)$. We claim that we have $t'_i = t'_{i+1} = t$. Indeed, since $\mathscr{O}$ is an optimum solution, it must be the case that the left-travelling batch $i+1$ as well as the right-travelling batch i start at time t; this follows from the fact that each of these batches is non-empty, and that starting later than t only increases total waiting time. Thus, we have a situation as depicted in Fig. 7.
To construct solution $\mathscr{O}'$ we schedule, on machine 1, a right-travelling batch starting at time $t - T$ and a left-travelling batch starting at time t. On machine 2, we schedule batches so that no job incurs waiting time in between the two machines, as described in Case 1. Again notice that each job served in either batch i or batch $i+1$ leaves its ending machine no later in solution $\mathscr{O}'$ than it does in solution $\mathscr{O}$, and that neither machine is occupied outside of the interval $[t_i, t_{i+1} + T)$.

For every pair of batches $(i, i+1)$ with i odd, solution $\mathscr{O}'$ thus consists of an interval $[t_i, t_{i+1}+T)$ containing the structure in Fig. 6. It is easily verified that such a solution is synchronized. Furthermore, since the intervals containing these structures do not overlap, solution $\mathscr{O}'$ is feasible. Finally, since the total waiting time in solution $\mathscr{O}'$ is not greater than the total waiting time of the solution $\mathscr{O}$ by construction, solution $\mathscr{O}'$ is also optimum. □

The proof of Theorem 3 implies that there exists an optimum synchronized solution in which each right-travelling (left-travelling) batch on machine 1 coincides with a simultaneous left-travelling (right-travelling) batch on machine 2. Recall that, since the solution is synchronized, no job waits in between the two machines. Informally stated, given the presence of this structure, the two machines operate as a single entity, both starting their batches at the same moments in time. Thus, when we set the processing time of a single machine to $2T$, ignore the direction of a job, and keep all other parameters the same, we have constructed a corresponding single-machine instance. This structure allows us to map each instance of this special case of SPBM to an instance of a single-machine scheduling problem that is addressed in Passchyn et al. (2016b). This discussion allows us to state the following corollary.

Corollary 1 *The special case of SPBM with two identical machines, a travel distance equalling zero, and each job to be processed by both machines, can be solved in polynomial time.*

Finally, we note that the proof of Theorem 3 can be seen to hold in two slightly more general settings. If the setting from Theorem 3 is modified so that it features non-identical processing times T_1 and T_2, the proof remains valid if the machine for which the schedule is copied from solution $\mathscr{O}$ to solution $\mathscr{O}'$ is the machine with the largest processing time. Similarly, if the setting from Theorem 3 is modified so that it features arbitrary capacity bounds B_1 and B_2, the proof remains valid if the chosen machine is the machine with the smallest capacity. So, the theorem continues to hold either for arbitrary processing times, or arbitrary capacity bounds. We note that it does not hold, however, for both of these extensions at the same time. This will be shown in the following section.

Table 4 Instance illustrating Observation 1

Jobs					Machines			
j	r_j	dir_j	s_j	e_j	i	T_i	B_i	x_i
1	0	Left	2	1	1	1	1	0
2	0	Left	2	1	2	2	2	0

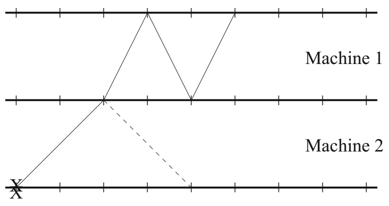


Fig. 8 Feasible solution for the instance of Table 4

3.2 Synchronized solutions in general

We show that, in general, there may not always exist an optimal solution that is synchronized. There are four conditions formulated in Theorem 3: (i) the machines are identical, (ii) there are two machines, (iii) the travel distance is equal to zero, and (iv) each job is served by each machine, i.e. for all $j \in R$: $s_j = 1$ and $e_j = 2$, and for all $j \in L$: $s_j = 2$ and $e_j = 1$. Each of these conditions is necessary, as shown by the following examples.

3.2.1 Two arbitrary machines

The first generalization we consider is the setting where the two machines are not identical. Let T_1, T_2 and B_1, B_2 denote the processing time and the capacity of the two machines, respectively. Consider the instance described in Table 4 and the solution (which is not synchronized) shown in Fig. 8. The total waiting time in this solution equals 2 time units. Observe that in any optimum solution, machine 2 starts a left-travelling batch at time 0 containing both jobs, otherwise the total waiting time equals at least 4 time units. Then, since both jobs cannot be served by machine 1 at time 2, there exists no feasible synchronized solution with a waiting time less than 4 time units. We can conclude the following.

Observation 1 *In case the two machines are not identical, there exist instances for which no optimum solution is synchronized.*

Recall that if either $T_1 \leq T_2$ and $B_1 \geq B_2$, or $T_1 \geq T_2$ and $B_1 \leq B_2$, i.e. if the 'fastest' machine is also the 'largest' machine, it is easily argued that we can restrict ourselves to solutions where the number of jobs in the machine with the highest capacity never exceeds the capacity of the other machine. We then obtain the setting with distinct processing times and equal capacity, for which an optimum synchronized solution exists, as shown in Sect. 3.1.

Table 5 Instance illustrating Observation 2

Jobs					Machines			
j	r_j	dir_j	s_j	e_j	i	T_i	B_i	x_i
1, 2, 3	0	Right	1	3	1	3	∞	0
4	3	Left	3	1	2	3	∞	0
5	7	Right	1	3	3	3	∞	0
6, 7, 8	11	Left	3	1				

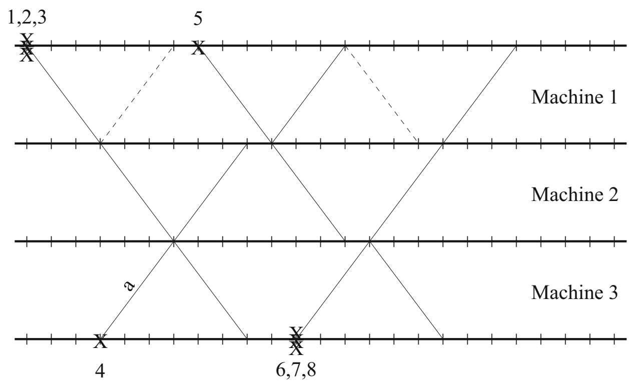


Fig. 9 Feasible solution for the instance of Table 5

3.2.2 Three machines

A different generalization extends the problem setting by including a third machine. We show that there may not exist an optimum synchronized solution, even when each job must be served by each of the three machines. Clearly, this result immediately generalizes to any setting with more than three machines.

Consider the instance described in Table 5 and the solution shown in Fig. 9. It is easily verified in Fig. 9 that the shown solution has a total waiting time of 2 time units. It follows that jobs 1, 2, 3, 6, 7, and 8 must not incur any waiting time in any optimum solution since the total waiting time would be no less than 3 time units otherwise. In order to obtain a total waiting time of at most 2 time units, it then follows that job 4 must be processed in a batch (denoted by a in the figure) starting immediately upon its arrival. Then, in any synchronized solution, machine 1 starts a left-travelling batch at time 9, so that job 5 incurs a total waiting time of at least 5 time units. Thus, for this instance, no synchronized solution with a total waiting time of at most 2 time units exists.

Observation 2 *In case there are three machines, there exist instances for which no optimum solution is synchronized.*

Table 6 Instance illustrating Observation 3

Jobs						Machines			
j	r_j	dir_j	s_j	e_j	v_j	i	T_i	B_i	x_i
1	0	Right	1	2	1	1	2	∞	0
2	2	Left	2	1	1	2	2	∞	2
3	5	Right	1	2	1				

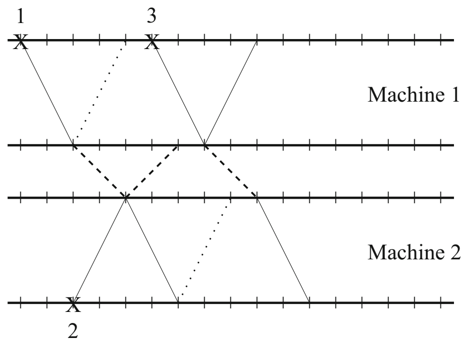


Fig. 10 Feasible solution for the instance of Table 6

3.2.3 Travel time

We now relax the assumption that the travel time in between the machines is equal to zero. Notice that the speed of jobs then becomes relevant. We show that there may not exist a synchronized schedule which is optimum. Perhaps surprisingly, this result also holds when all jobs travel at the same speed and thus spend the same travel time in between the machines, regardless of their direction of travel. The existence of an optimum solution which is synchronized thus highlights a noteworthy difference between the settings with and without travel time.

Consider the instance described in Table 6 and the feasible solution shown in Fig. 10. It is easily verified in Fig. 10 that, in the shown solution, job 2 incurs a waiting time of 1 time unit, whereas jobs 1 and 3 incur no waiting time. Also note that this solution is not synchronized, since the waiting time for job 2 occurs in between the machines. The total waiting time in this solution equals 1 time unit. Observe that in any feasible solution with a total waiting time of at most 1 time unit, at least two of the jobs incur no waiting time. Since, in any feasible solution, either job 2 or job 3 has at least one unit of waiting time, we conclude that there are only two cases to consider:

1. Jobs 1 and 2 have no waiting time. It immediately follows that the earliest time at which machine 1 can serve job 3 is time 8, so that the total waiting time in this solution is no less than 3 time units.
2. Jobs 1 and 3 have no waiting time. It is easily verified that, if job 2 enters a left-travelling batch in the time interval $[2, 6)$, the batches required to serve job 2 would overlap with one of the batches serving job 1 or job 3. The earliest time at which job 2 can enter machine 2 so that it incurs no waiting time in between the machines is time 6. The total waiting time of such a solution is then no less than 4 time units.

We conclude that a synchronized schedule for this instance must have a total waiting time of at least 3 time units, while a feasible solution exists with a total waiting time of 1 time unit.

Observation 3 *In case the travel time between the two machines is not equal to zero, there exist instances for which no optimum solution is synchronized.*

3.2.4 Not all jobs pass both machines

Let us now consider the generalized setting where jobs need not necessarily pass through each of the machines. We again show that there does not always exist an optimum synchronized solution. In fact, this holds even in a more restrictive setting where there is one machine that must serve all jobs; we refer to such a setting as a setting with a 'common machine'. The uni-directional variant of such a common machine setting is discussed in Sect. 4; in this case, it is shown that an optimum synchronized solution exists.

Consider the instance described in Table 7 and the feasible solution shown in Fig. 11. Clearly, the total waiting time in this solution equals 1 time unit. Observe that in any feasible solution with a total waiting time of no more than 1 time unit, at least two of the jobs have no waiting time. Since, in any feasible solution, either job 1 or job 3 has at least one unit of waiting time, we conclude that there are only two cases to consider:

1. Jobs 1 and 2 have no waiting time. It then immediately follows that machine 1 starts a left-travelling batch at time 2, and that job 3 can enter machine 1 no earlier than time 6. Any solution where this is the case thus has a total waiting time no less than 3 time units.
2. Jobs 2 and 3 have no waiting time. It follows that machine 1 processes a left-travelling batch starting at time 3, and machine 2 processes a right-travelling batch starting at time 2. It follows that the earliest time at which job 1 can enter machine 1 in a feasible synchronized solution is thus time 7, so that such a solution has a total waiting time no less than 5 time units.

It follows from the above that any feasible synchronized solution thus has a total waiting time no less than 3 time units.

Table 7 Instance illustrating Observation 4

Jobs					Machines			
j	r_j	dir_j	s_j	e_j	i	T_i	B_i	x_i
1	0	Left	2	1	1	2	∞	0
2	0	Right	1	2	2	2	∞	0
3	3	Left	1	1				

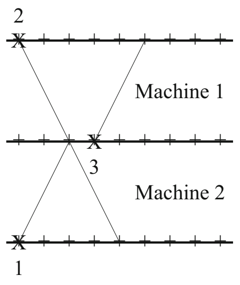


Fig. 11 Feasible solution for the instance of Table 7

Thus, there does not exist a synchronized solution which is optimum for this instance.

Observation 4 *In case not all jobs must be served by each of the machines, there exist instances for which no optimum solution is synchronized.*

4 Uni-directional traffic with a common machine

We prove that a uni-directional special case of SPBM can be solved in polynomial time. The problem setting discussed here considers identical machines with infinite capacity and jobs travelling in a single direction at identical speeds. We assume for simplicity that jobs travel at unit speed; it is easily seen that, by modifying the distance between machines, any result for unit speed extends immediately to the setting with arbitrary identical speeds. Jobs may arrive at arbitrary positions, i.e. a job need not be processed by each of the machines. A key difference with the uni-directional variant of the setting from Sect. 2, however, is that we assume the existence of at least one machine that must serve each of the jobs. We refer to this machine as the 'common machine'. That is, there exists a machine $m^* \in M$ satisfying $s_j \le m^* \le e_j$ for all jobs $j \in J$. Throughout this section, we assume that all machines are ordered from left to right and that all jobs are right-travelling.

The underlying idea of the proposed method for solving a given instance $\mathscr{I}$ of the problem setting stated above is as follows.

1. Construct a single-machine instance $\mathscr{I}'$ that is equivalent to instance $\mathscr{I}$. A formal definition of this equivalence relationship will be given below.
2. Solve this single-machine instance to optimality, for example by using a dynamic programming algorithm described by Passchyn et al. (2016b).
3. Extend the obtained solution to a synchronized solution for the original instance $\mathscr{I}$.

We prove that this procedure yields an optimal schedule for the original instance. In the following, we describe these steps in detail and argue that the obtained solution has minimum total waiting time.

Theorem 4 *Problem SPBM for identical machines with infinite capacity, jobs travelling in the same direction with equal speed, and a common machine, reduces to solving the uni-directional case of a single machine with infinite capacity.*

Proof Given an instance $\mathscr{I}$, we obtain an equivalent single-machine instance $\mathscr{I}'$ as follows. Let the processing time of the machines in $\mathscr{I}$ be equal to T. In instance $\mathscr{I}'$, we have a single machine with processing time T and infinite capacity. Let the set of jobs in $\mathscr{I}'$ be empty initially. For each job j in $\mathscr{I}$, we create a right-travelling job arriving at the machine. The release time r'_j of this job in the single-machine instance is equal to r_j minus the time needed to travel without any waiting time from the left-hand side of the first machine in $\mathscr{I}$ to the left-hand side of machine s_j:

$$r'_j = r_j - (s_j - 1)T - \sum_{i=1}^{s_j-1}(x_{i+1} - x_i).$$

This defines instance $\mathscr{I}'$. Informally, a release time r_j is thus 'traced back' to the first machine by subtracting a fictitious travel time assuming that this job j incurs no waiting time. This is visualized in Fig. 12. Observe that the lines that trace back these arrivals do not correspond to batches that are actually scheduled and thus need not be spaced without overlap. Also note that a job may be traced back along a machine on which it need not be served in instance $\mathscr{I}$. This completes the definition of instance $\mathscr{I}'$.

Next, we describe how to extend a solution for the single-machine instance $\mathscr{I}'$ to a solution for the original instance $\mathscr{I}$. In the solution for $\mathscr{I}$, we schedule batches for the first machine at the same starting times and in the same direction as the batches in the solution for $\mathscr{I}'$. We then extend this solution to a synchronized solution as if assuming that each right-travelling batch contains a job travelling to the last machine. More formally, let $\mathscr{T}$ be the set of starting times of the right-travelling batches of the single machine in the solution to $\mathscr{I}'$. We schedule, for each machine $i \in M$ in instance

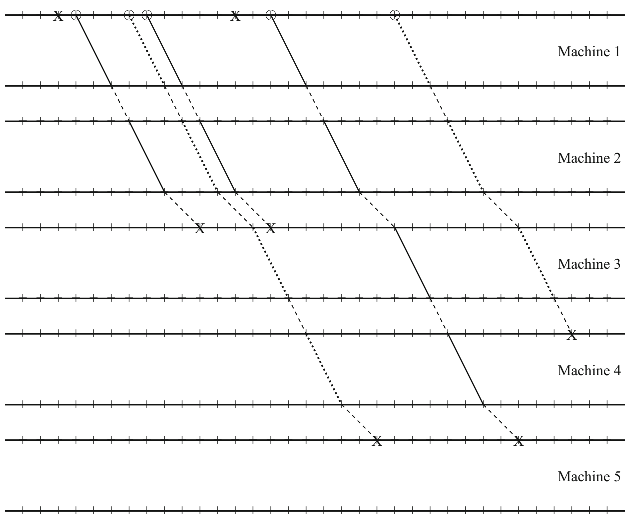


Fig. 12 Release times for instance $\mathscr{I}'$. The circles mark the times of arrival in $\mathscr{I}'$ corresponding to the original arrivals at machines $2, \ldots, 5$

$\mathscr{I}$ and for each $t \in \mathscr{T}$, a right-travelling batch starting at time t', with

$$t' = t + (i-1)T + \sum_{\ell=1}^{i-1}(x_{\ell+1} - x_\ell).$$

Observe that since all machines have identical processing time, the separation property is satisfied and the constructed solution is feasible. Further observe that, by construction, the result is a synchronized schedule.

In the solution to $\mathscr{I}$, we process each job in the first available right-travelling batch. Notice that a number of right-travelling batches may be empty. Indeed, since not all arrivals in $\mathscr{I}$ occur at the first machine, and since not all jobs in $\mathscr{I}$ travel to the last machine, right-travelling batches on a machine before a job's starting machine, or after a job's ending machine, may in fact remain empty. Clearly, we can simply remove these empty batches with no impact on the obtained solution.

This completes the description of the solution to $\mathscr{I}$ obtained by extending a solution to the single-machine instance $\mathscr{I}'$. Clearly, in any feasible solution to the single-machine instance, all jobs are processed by the machine. The constructed solution to $\mathscr{I}$ is thus feasible. We now argue that the single-machine instance $\mathscr{I}'$ is equivalent to the original instance $\mathscr{I}$:

Lemma 1 *A solution with a total waiting time of at most W for instance $\mathscr{I}'$ exists if and only if a solution with a total waiting of at most W exists for the given instance $\mathscr{I}$.*

Proof $\Rightarrow$ Consider a schedule with total waiting time W for the single-machine instance $\mathscr{I}'$, and the corresponding solution for instance $\mathscr{I}$ obtained by extending the single-machine solution as described above. For each job in $\mathscr{I}'$, the waiting time incurred at the machine is equal to the waiting time incurred at the corresponding job's starting position in instance $\mathscr{I}$. Indeed, this is trivial for all jobs arriving at the first machine, since both their time of arrival and the schedule of the first machine are identical in instances $\mathscr{I}$ and $\mathscr{I}'$. For all other jobs, it can be seen that the difference between the release time of a job $j \in J$ and the corresponding job in $\mathscr{I}'$, is equal to the difference between a right-travelling batch starting on the first machine and a corresponding right-travelling batch of machine s_j.

Thus, by construction, the total waiting time of a job in the solution for $\mathscr{I}'$ is equal to the waiting time of the corresponding job in the solution for $\mathscr{I}$. Since this holds for all jobs, it follows that the constructed solution for $\mathscr{I}$ has a total waiting time of exactly W.

$\Leftarrow$ Now consider a feasible solution $\mathscr{F}$ for instance $\mathscr{I}$ with a total waiting time of at most W. Let us first argue that there exists a synchronized schedule with a total waiting time of at most W. For each job $j \in J$, let t_j be the starting time of the right-travelling batch of machine m^* containing j. Next, we define r_j^* to be the latest possible time at which a job can enter a batch of machine s_j so that it reaches machine m^* at time t_j, for $j \in J$. Due to the unavoidable total processing time and the travel time between machines s_j and m^*, we have:

$$r_j^* = t_j - (m^* - s_j)T - \sum_{i=s_j}^{m^*-1}(x_{i+1} - x_i).$$

Clearly, since solution $\mathscr{F}$ is feasible, we have $r_j \leq r_j^*$ for all jobs $j \in J$. Further, we define C_j^* as the earliest possible time at which a job can leave its ending machine e_j if it enters machine m^* at time t_j. Due to the unavoidable total processing time and the travel time between machines m^* and e_j, we have:

$$C_j^* = t_j + (e_j - m^* + 1)T + \sum_{i=m^*}^{e_j-1}(x_{i+1} - x_i).$$

Consider the set of starting times of non-empty right-travelling batches of machine m^* in solution $\mathscr{F}$, and let us denote this set by $\mathscr{T}_{m^*}$. We thus have $\mathscr{T}_{m^*} = \{t_j \mid j \in J\}$. Based on this set of starting times we now construct a solution $\mathscr{F}^S$ by extending the batches scheduled on machine m^* at times t_j to a synchronized schedule for all machines $i \in M$, using the same procedure as illustrated in Fig. 13. Notice that, since it may be the case that $m^* > 1$, we extend the solution to machines on the left of machine m^* as well as to the right of machine m^*.

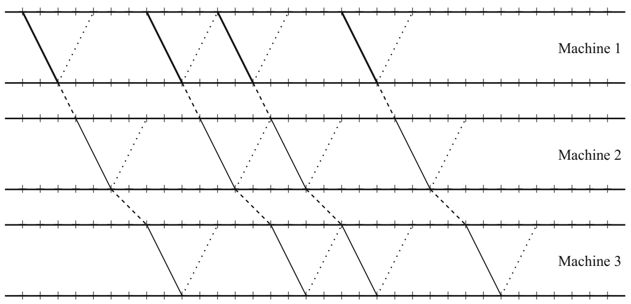


Fig. 13 Extending a solution for the single-machine instance $\mathscr{I}'$ (represented as thick solid lines) to a synchronized solution for an instance $\mathscr{I}$ featuring three machines. The dotted lines correspond to the separation property

More formally, let $\mathscr{T}_i^S$ represent the starting times of all right-travelling batches on machine i in solution $\mathscr{F}^S$. We then set the starting times on each machine $i \in M$ as follows:

- For machine m^*, we have $\mathscr{T}_{m^*}^S = \mathscr{T}_{m^*} = \{t_j \mid j \in J\}$.
- For all machines i with $i < m^*$, we have $\mathscr{T}_i^S = \{t_j - (m^* - i)T - \sum_{\ell=i}^{m^*-1}(x_{\ell+1} - x_\ell) \mid j \in J\}$.
- For all machines i with $i > m^*$, we have $\mathscr{T}_i^S = \{t_j + (i - m^*)T + \sum_{\ell=m^*}^{i-1}(x_{\ell+1} - x_\ell) \mid j \in J\}$.

Clearly, the solution $\mathscr{F}^S$ that is thus constructed is synchronized. Furthermore, since the given solution $\mathscr{F}$ is feasible and since all machines are identical, the scheduled batches in $\mathscr{F}^S$ do not overlap. Observe that, in $\mathscr{F}^S$, each job $j \in J$ that is served by machine m^* at time t_j, is served by its arrival machine s_j at time r_j^*. Since $r_j^* \geq r_j$ for all jobs $j \in J$, the constructed solution $\mathscr{F}^S$ is feasible. Finally observe that, in $\mathscr{F}^S$, each job $j \in J$ that is served by machine m^* at time t_j, leaves its departure machine at time C_j^*. It follows that each job $j \in J$ leaves its ending machine no later in solution $\mathscr{F}^S$ than it does in solution $\mathscr{F}$. Solution $\mathscr{F}^S$ is thus a synchronized solution with a total waiting time equal to at most W.

By the construction of instance $\mathscr{I}'$, the waiting time incurred by a job $j \in J$ in a synchronized solution for instance $\mathscr{I}$ is exactly equal to the waiting time incurred by the corresponding job in a solution for instance $\mathscr{I}'$. There thus exists a solution with a total waiting time of at most W for $\mathscr{I}'$. □

From the lemma establishing the equivalence between an instance $\mathscr{I}$ and the corresponding instance $\mathscr{I}'$, it immediately follows that minimizing the total waiting time for instance $\mathscr{I}'$ yields an optimum solution for the original instance $\mathscr{I}$, proving the theorem. □

We note that constructing the instance $\mathscr{I}'$, as well as extending the obtained solution for the single-machine instance to a solution for the original instance, can be achieved in polynomial time. To solve the single-machine instance, a $O(n^2)$ dynamic programming algorithm described by Passchyn et al. (2016b) can be used. In fact, since the single-machine instances in the setting above are restricted to the uni-directional setting, a more efficient algorithm may also exist. Note that, since the only instance that must be solved throughout this procedure features only a single machine, the computational complexity of the proposed algorithm does not depend on the number of machines.

5 Conclusion

In this work, we investigated the complexity of optimally scheduling a sequence of parallel batching machines with respect to the total waiting time. We showed that this problem is strongly NP-hard, both in the special case with two identical machines in the uni-directional setting, as well as in the special case with identical machines, identical job speed, in the bidirectional setting. Additionally, we introduced the notion of a synchronized solution, we identified two special cases for which every instance has an optimum solution which is synchronized, and we showed how to find these solutions in polynomial time.

We observe that the complexity of scheduling parallel batching machines in sequence is not fully characterized for all special cases. In order to close the gap between computationally easy and computationally hard problems, it may be interesting to investigate the complexity of the setting with identical machines, identical job speed, and uni-directional travel. Note that this case, where jobs may arrive and depart at arbitrary positions, separates the settings for which we obtained results in Sects. 2 and 4. Another problem setting that remains open is whether the bidirectional setting with identical machines, identical job speed, and a common machine, can be solved in polynomial time. Further, it may also be interesting to investigate whether SPBM with identical job speed can be solved in polynomial time for a fixed number of machines, for example by means of dynamic programming. Finally, motivated by the application in lock scheduling, it is interesting to consider a more sophisticated way of dealing with the capacity of the machines. It is conceivable to consider a problem where the jobs are partitioned into different (size-)classes and where the capacity of a machine is described in terms of numbers of jobs of each class that can jointly be processed.

Acknowledgements This research has been partially funded by the Interuniversity Attraction Poles Programme initiated by the Belgian Science Policy Office. This work was carried out when both authors were affiliated with KU Leuven. We thank the associate editor and the referees for their careful reading of the manuscript.

A Uni-directional traffic: proof of Theorem 1

Here, we provide a proof of Theorem 1, i.e. we show that problem SPBM is strongly NP-hard, even in the uni-directional setting with two identical machines and where each job must be processed by both machines. The proof is inspired by a hardness proof by Disser et al. (2015) for a problem featuring bidirectional traffic along a path where a limited number of spots are available to allow overtaking and crossing of traffic.

We start our reduction from MAX CUT, which consists of answering the following question: given a graph $G = (V, E)$, does there exist a cut consisting of at least K edges? This problem is shown to be NP-hard, see Garey et al. (1976). Notice that we consider the unweighted case, sometimes referred to as SIMPLE MAX CUT.

For a given instance of MAX CUT, we describe a corresponding instance of the decision version of SPBM. We will then argue that solving dec-SPBM for this instance corresponds to deciding the question of MAX CUT. We first turn G into a directed graph by choosing some ordering of the vertices in V, and next orienting every edge from the vertex with smaller index to the vertex with larger index.

The instance $\mathscr{I}$ of dec-SPBM is as follows. There are two identical machines, called machine 1 and machine 2 from left to right. Both machines have unit processing times (i.e. $T_1 = T_2 = 1$), the capacity of each machine is infinite (i.e. $B_1 = B_2 = \infty$), and the distance between the two machines is equal to 1 (i.e. $x_1 = 0$ and $x_2 = 1$). The set of jobs J consists of a total of $n = 2|V||E| + 2|V| + 2|E|$ jobs. All jobs are right-travelling and must be processed by both machines, i.e. $s_j = 1$ and $e_j = 2$ for all $j \in J$. For ease of exposition, we distinguish two types of jobs: *vertex jobs* and *edge jobs*. Note that the speed of the jobs need not be identical, as will be described in what follows.

We now specify the release times r_j for each job $j \in J$. Recall that we have imposed an arbitrary order on V: let $V = \{1, \ldots, |V|\}$. For each vertex $v \in V$, we have $|E| + 1$ vertex jobs arriving at time $5(v - 1)$ and $|E| + 1$ vertex jobs arriving at time $5(v - 1) + 1$. We thus have a total of $2|V|(|E| + 1)$ vertex jobs. For each vertex $v \in V$, we say that the time interval $[5(v - 1), 5v)$ on the first machine is the period corresponding to vertex v on the first machine, and the time interval $[5|V| + 5(v - 1), 5|V| + 5|V|)$ is the period corresponding to vertex v on the second machine. The speed of each vertex job equals $1/(5|V|-1)$, i.e. each vertex job needs $5|V| - 1$ time units to travel the distance between the two machines.

In addition, there are two edge jobs for each $(v_i, v_j) \in E$: one job arriving at time $5(v_i - 1)$, and one job arriving at time $5(v_i - 1) + 1$. We will refer to these arrivals as the *first job* and *second job* corresponding to edge (v_i, v_j), respectively. Clearly, we have $2|E|$ edge jobs in total. The speed of the first job corresponding to edge (v_i, v_j) equals $1/(5(|V|+v_j-v_i))$, while the speed of the second job equals $1/(5(|V|+v_j-v_i)-2)$.

The question is: does there exist a solution with total waiting time of at most $W \equiv |V||E| + |V| + 3|E| - 2K$? This completes the description of the instance of dec-SPBM. An overview of the constructed instance of dec-SLS is shown in Fig. 14. A detailed illustration of the vertex jobs and edge jobs follows in the remainder of this section.

Before we argue the equivalence between a yes-instance of MAX CUT and a yes-instance of dec-SPBM, let us first explicitly describe two particular ways of serving the jobs arriving in a given period. Figure 15 illustrates two possible ways of serving the jobs arriving in a period corresponding to some vertex v on the first machine.

In the first option, all jobs arriving at time $5(v - 1)$ constitute a batch, and start being processed immediately. At time $5(v - 1) + 1$, machine 1 has finished processing these jobs,

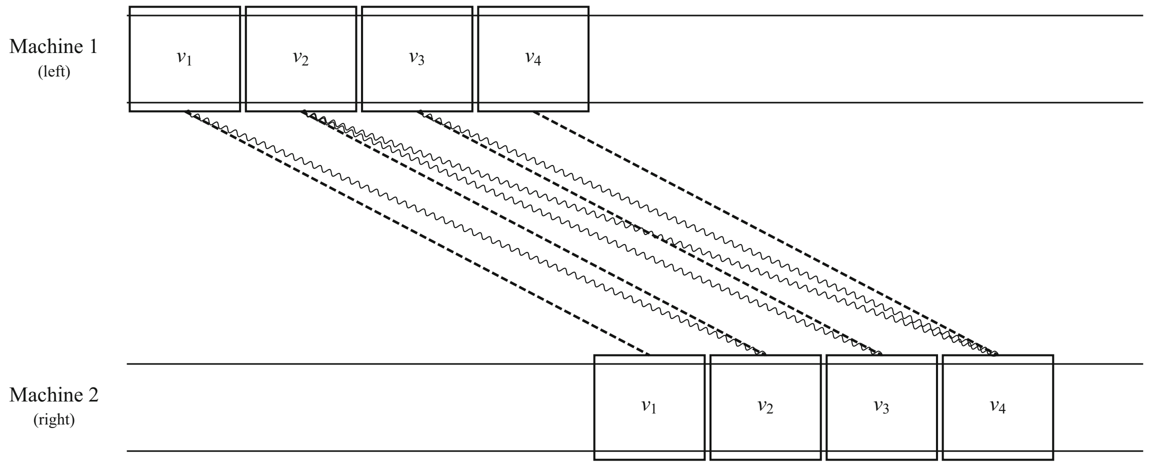


Fig. 14 Overview of a constructed instance of dec-SPBM. The blocks represent periods corresponding to the vertices in G. Dashed lines represent vertex jobs, while waved lines represent edge jobs

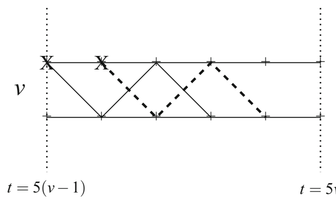


Fig. 15 Illustration of option 1 (solid lines) and option 2 (dashed lines) to serve the jobs arriving in a period on the first machine

which then travel to machine 2. Due to the separation property, machine 1 can start processing a new right-travelling batch not before $5(v-1)+2$. This batch contains all jobs that arrived at time $5(v-1)+1$. Notice that the latter jobs incur 1 unit of waiting time. We refer to this way of serving the jobs in this period as *option 1*. A second way to serve the jobs corresponding to vertex v on the first machine is for all of these jobs to constitute a single batch, and start processing at time $5(v-1)+1$. In this case, notice that the jobs arriving at time $5(v-1)$ incur 1 unit of waiting time. We refer to this way of serving the jobs that arrive in this period as *option 2*.

Given these two options for serving the arrivals at machine 1, we can define two similar options for each period on machine 2. Let option 1 on the second machine consist of right-travelling batches starting at times $5|V|+5(v-1)$ and $5|V|+5(v-1)+2$; let option 2 on the second machine consist of right-travelling batches starting at times $5|V|+5(v-1)+1$ and $5|V|+5(v-1)+3$.

Notice that, on either machine, options in different periods are independent of each other since there is enough time to ensure that the separation property is satisfied across periods, regardless of which option is chosen for the different periods. For example, selecting option 1 in the period [5, 10) does not prevent us from selecting either option 1 or option 2 in the periods [0, 5) or [10, 15).

We now argue the equivalence between a yes-instance of MAX CUT, and a yes-instance of dec-SPBM. Suppose that there exists a cut in the graph G with at least K edges; let the corresponding partition of the node-set V be indicated by V_1 and V_2. We build the following solution for the instance of dec-SPBM. If vertex $v \in V_1$, we use option 1 for the two periods corresponding to vertex v; if vertex $v \in V_2$, we use option 2 for the two periods corresponding to vertex v. We claim that a solution with total waiting time bounded by W then arises by (i) scheduling each job arriving at the first machine in the first possible batch; next, (ii) having each job immediately travel to the second machine after leaving the first machine; and finally (iii) scheduling in the first possible batch of the second machine.

We first revisit the vertex jobs. Observe that in each period on machine 1, no matter whether option 1 or option 2 is used, one half of the arriving vertex jobs in that period has no waiting time and the other half incurs a waiting time of 1 time unit. This accounts for a total waiting time of $|V|(|E|+1)$ for the vertex jobs at the first machine. Recall that these jobs require a travel time of $5|V|-1$ in between the two machines. Suppose that option 1 is selected for the period corresponding to v on machine 1. This means that at time $5(v-1)+1$ and at time $5(v-1)+3$, there are $|E|+1$ vertex jobs leaving machine 1 and travelling towards machine 2. It follows that these jobs are available for processing on machine 2 at times $5|V|+5(v-1)$ and $5|V|+5(v-1)+2$, respectively. Observe that this period on machine 2 also corresponds to vertex v and hence uses option 1. Thus, these jobs do not incur any waiting time at machine 2. In the case that $v \in V_1$, we thus have a waiting time of $|E|+1$ for the vertex jobs corresponding to v. In the case that $v \in V_2$, a similar argument can be made. Indeed, the $|E|+1$ jobs arriving at time $5(v-1)$ then each incur a waiting time of 1 time unit at the first machine; all vertex jobs arriving in the period corresponding to v leave the first machine at time $5(v-1)+2$ and are available for processing on the second machine at time $5|V|+5(v-1)+1$. Since option 2 was selected for this period, these jobs do not incur any additional waiting time at machine 2. It follows that the total waiting time due to vertex jobs equals $|V|(|E|+1)$. An illustration of this construction is provided in Fig. 16.

We now look at the edge jobs. Consider an edge $(v_i, v_j) \in E$ (with $v_i < v_j$). Recall that the first job corresponding to this edge requires a travel time of $5(|V|+v_j-v_i)$, and the second job requires a travel time of $5(|V|+v_j-v_i)-2$. For convenience, let $t_i = 5(v_i-1)$ and $t_j = 5|V|+5(v_j-1)$, so that t_i and t_j are the starting time of the period corresponding to v_i on machine 1 and the starting time of the period corresponding to v_j on machine 2, respectively. We now have the following four cases to consider, illustrated graphically in Fig. 17.

- Case 1: option 1 is used for the period corresponding to v_i on machine 1 and option 1 is used for the period corresponding to v_j on machine 2. Then, the first job is processed by machine 1 at its release time t_i, leaves machine 1 at t_i+1, and due to its speed, arrives at machine 2 at time t_j+1, where it has to wait 1 time unit in order to be processed. The second job waits 1 time unit in front of machine 1, leaves machine 1 at t_i+3, and arrives at machine 2 at time t_j+1. Hence, the second job also has to wait 1 time unit in front of machine 2. The total waiting time for these two jobs in this case equals 3 time units.
- Case 2: option 1 is used for period v_i on machine 1 and option 2 is used for period v_j on machine 2. Then, the first job leaves machine 1 at t_i+1 and arrives at machine 2 at time t_j+1. The second job incurs one unit of waiting time at machine 1, leaves machine 1 at time t_i+3, and arrives at machine 2 also at time t_j+1. Hence, neither job incurs any additional waiting time at machine 2; the

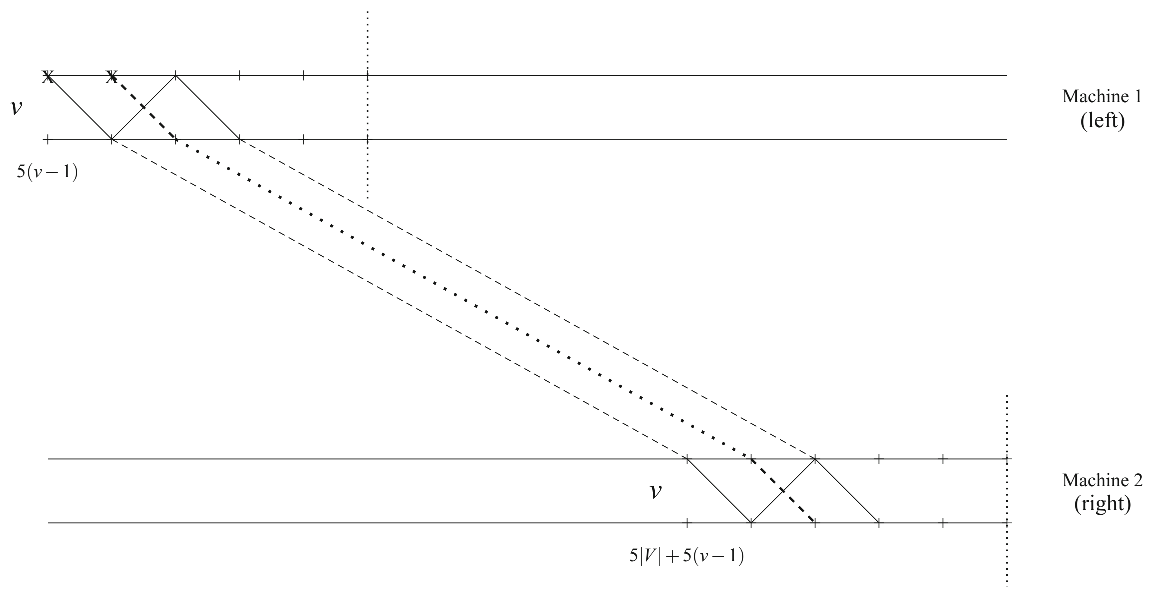
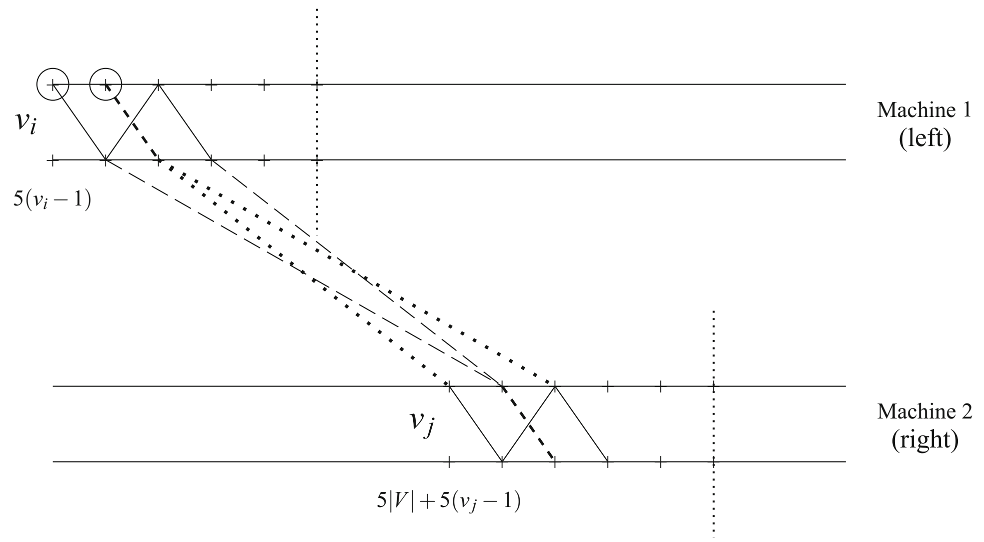


Fig. 16 Illustration of the vertex jobs in a period corresponding to a vertex v. Each 'X' marks the arrival of $|E| + 1$ vertex jobs

Fig. 17 Illustration of the edge jobs in a period corresponding to an edge (v_i, v_j). Each circle marks the arrival of a single edge job



total waiting time for these two jobs in this case equals 1 time unit.

– Case 3: option 2 is used for period v_i on machine 1, and option 1 is used for period v_j on machine 2. Then, both the first and second job are processed by machine 1 at time $t_i + 1$ and leave machine 1 at $t_i + 2$. Due to their speeds, the first job arrives at machine 2 at time $t_j + 2$, while the second job arrives at machine 2 at time t_j. It follows that neither job incurs any additional waiting time at machine 2; the total waiting time for these two jobs in this case equals 1 time unit.
– Case 4: option 2 is used for period v_i on machine 1, and option 2 is used for period v_j on machine 2. Then, both the first and second job are processed by machine 1 at time $t_i + 1$ and leave machine 1 at $t_i + 2$. Due to their speeds, the first job arrives at machine 2 at time $t_j + 2$, while the second job arrives at machine 2 at time t_j. Both jobs incur an additional waiting time of 1 time unit. The total waiting time for these two jobs in this case equals 3 time units.

We conclude this case analysis by observing that if the same option is selected for a period v_i on machine 1 and a period v_j on machine 2 that correspond to an edge $(v_i, v_j) \in E$, there is a waiting time of 3 time units corresponding to this edge. If the two selected options for the periods corresponding to this edge differ, there is a waiting time of 1 time unit.

Since the cut in G contains K edges, we infer that the edge jobs have a total waiting time of $K+3(|E|-K) = 3|E|-2K$. Indeed, observe that if edge (v_i, v_j) is in the cut, i.e. if $v_i \in V_1$ and $v_j \in V_2$, the two options used for periods v_i and v_j differ, resulting in a waiting time of 1 corresponding to this edge; otherwise, there is a waiting time of 3. Hence, the total waiting time for all jobs equals $|V|(|E|+1)+3|E|-2K = W$. A yes-instance of MAX CUT thus gives rise to a yes-instance of dec-SPBM.

Consider now a solution to the instance of SPBM with a total waiting time of at most W. First, we argue that we can assume that such a solution is so-called *sensible*. We say that a solution to dec-SPBM is sensible if

- Condition 1: each job enters the first right-travelling batch that occurs at or after its arrival at a machine,
- Condition 2: in each period on machine 1 and on machine 2, either option 1 or option 2 is used.

We argue that we can restrict ourselves to sensible solutions only.

Lemma 2 *For any feasible solution to SLS with total waiting time W', there exists a sensible solution to SLS with total waiting time at most W'.*

Proof It is easily argued that Condition 1 can be enforced without increasing the total waiting time. Indeed, given any solution that does not satisfy Condition 1, it is obvious that each job for which an earlier batch assignment is available can be immediately reassigned to that earlier batch; this does not increase the total waiting time. We note that an equivalent statement also holds when the capacity of each machine is bounded: each job then enters the first non-full batch starting at or after its arrival at a machine.

To see that Condition 2 can be guaranteed, observe that there exists an optimal solution where each batch of a machine either (i) starts at a moment in time where some job that is served by the batch arrives at this machine, or (ii) starts immediately upon the completion time of a preceding batch of this machine. If neither is the case, it is easily seen that such a batch can start earlier in time, which cannot increase the total waiting time. Consider now a period on the first machine, corresponding to some vertex v. It follows that we can restrict ourselves to solutions where the first right-travelling batch starts at time $5(v-1)$ or at time $5(v-1)+1$. This corresponds to either option 1 or option 2.

As a result, on the second machine, jobs arrive only at times $t, t+1$, or $t+2$, where t denotes the starting time of a period corresponding to some vertex $v \in V$. Thus, the same argument can be repeated to see that selecting either option 1 or option 2 yields minimum waiting time for any given period on the second machine. Since this holds independently for each period on each machine, the claim follows. □

We may now assume that the given solution for dec-SPBM is a sensible solution with a total waiting time bounded by W. Let us argue that the instance of MAX CUT is then a yes-instance. Since our solution for dec-SPBM is a sensible solution, it follows that in each period either option 1 or option 2 is used. We again consider the waiting time of the two types of jobs. For each vertex $v \in V$, observe that there are $2(|E|+1)$ vertex jobs arriving in the period corresponding to v on the first machine. These jobs incur a total waiting time of $|E|+1$ if both periods corresponding to v use the same option, and a total waiting time of $3(|E|+1)$ if the two periods use different options. For the edge jobs, recall that the two jobs corresponding to an edge (v_i, v_j) incur a total waiting time of 1 if different options are used for the period corresponding to v_i on the first machine and the period corresponding to v_j on the second machine; if the same option is used for these periods, the total waiting time equals 3 time units. Observe that, since there are $|E|$ edges and $|V|$ periods where vertex jobs arrive, a total waiting time of $|V|(|E|+1)+|E|$ cannot be avoided.

We claim that the two periods corresponding to any given vertex $v \in V$ must use the same option. We argue by contradiction. Recall that the total waiting time must equal at least $|V|(|E|+1)+|E|$ and that an additional waiting time of $2(|E|+1)$ is incurred for every vertex v for which the two corresponding periods are scheduled with different options. Assume that there is a single vertex for which this is the case. It follows that the total waiting time is then at least $|V|(|E|+1)+|E|+(2|E|+1) > |V||E|+|V|+3|E|-2K = W$. This contradicts that our solution for dec-SPBM has a waiting time of at most W. It follows that for each vertex $v \in V$, the two corresponding periods are scheduled using the same option. We can conclude that in any sensible schedule with a total waiting time no greater than W, the total waiting time equals $|V|(|E|+1)+|E|$ plus an additional waiting time of 2 time units for every edge $(v_i, v_j) \in E$ (with $v_i < v_j$) where the periods corresponding to v_i are scheduled in the same state as the periods corresponding to v_j.

Finally, we construct a solution to MAX CUT by assigning a vertex v to V_1 (V_2) if option 1 (option 2) is used for the two periods corresponding to vertex v. Then, since the solution to dec-SPBM has a total waiting time no greater than $W = |V||E|+|V|+|E|+2(|E|-K)$, it follows that there are at least K pairs of vertices (v_i, v_j) such that the period corresponding to v_i on machine 1 is scheduled with a different option than the period corresponding to v_j on machine 2. Indeed, if only $K-1$ pairs use different options, the total waiting time equals at least $|V||E|+|V|+(K-1)+3(|E|-(K-1)) = |V||E|+|V|+3|E|-2K+2$. By construction, there are thus at least K pairs of vertices v_i and v_j with $(v_i, v_j) \in E$ such that exactly one of these vertices is in V_1 and the other is in V_2. Thus, there are at least K edges in the resulting cut in G. A yes-instance of dec-SPBM thus gives rise to a yes-instance of MAX CUT, which completes our reduction.

As a remark, we note that the construction of this reduction can be modified so that each machine has unit capacity and all arrivals occur at distinct times. This can be achieved by extending the length of each period corresponding to a vertex and spreading all simultaneous arrivals out over time. Notice that Lemma 2 also holds in this more general setting. We omit a formal description of this proof. As a corollary, it then follows that SPBM is strongly NP-hard for each fixed B_i.

B Jobs with equal speed: proof of Theorem 2

We provide a proof for Theorem 2, i.e. we show that SPBM is strongly NP-hard even in the setting with identical machines and where the speed of all jobs is the same. Note that in the reduction outlined below, jobs travel in both directions. A crucial difference with the reduction provided in 'Appendix A' is that, here, the number of machines is not bounded by a constant. Furthermore, jobs need not be served by each of the machines: we specify a starting machine s_j and an ending machine e_j for all jobs $j \in J$.

The general outline of the reduction is inspired by a reduction for a problem involving bidirectional traffic on a path, described by Disser et al. (2015). This setting, however, does not correspond exactly to the machine scheduling setting, as mentioned in Sect. 1.1. The wording and presentation of the proof in this section also resemble the proof of Theorem 1.

We again start our reduction from MAX CUT; recall that an instance of MAX CUT consists of a graph $G = (V, E)$ and a non-negative integer K. To aid the exposition of the reduction, we first provide a general overview of the reduction before describing all details. The total number of machines in the instance of SPBM is not bounded by a constant, although we argue below that this number is bounded by $O(|E|)$. On the machines, we describe periods that correspond to the vertices in the instance of MAX CUT. A crucial part of the construction is that, on each machine that contains these periods, the order of the vertices corresponding to the periods is permuted. Similar to the argument in 'Appendix A', we first show that there are only two sensible scheduling options for each period; the option that is chosen in a given period reflects the assigned partition in the corresponding instance of MAX CUT.

Figure 18 illustrates a simplified overview of the fundamental part of the construction: a set of periods on a sequence of machines that represents the existence of an edge. In the figure, this is shown for an edge (v_1, v_4). Each square in

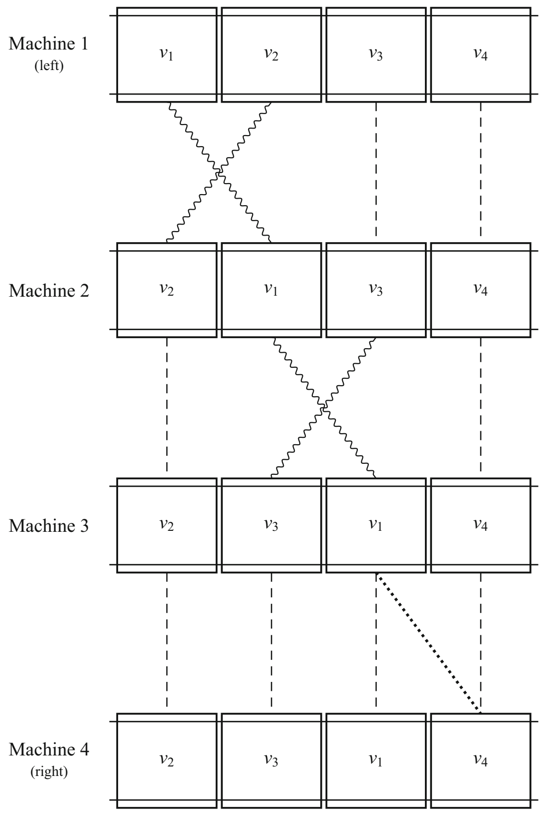


Fig. 18 Illustration of the construction for an edge (v_1, v_4). Only the odd-numbered machines are shown. Each box represents a period corresponding to a vertex. Dashed lines connect periods corresponding to the same vertex on the same time interval; waved lines connect adjacent periods for which the corresponding vertex is interchanged; dotted line models the existence of an edge between diagonally adjacent periods

the figure represents a period on one of the machines; each period corresponds to a vertex. We say that two periods are *adjacent* if they occupy consecutive time intervals on the same machine. We say that two periods are *diagonally adjacent* if they occupy consecutive time intervals on two consecutive odd-numbered machines. Note that in Fig. 18, only the odd-numbered machines are shown. The role of the even-numbered machines will be specified in the detailed description of the construction. The main components of the construction are (1) sets of jobs, represented in the figure by dashed lines, which ensure that two periods occupying the same time interval on different machines correspond to the same vertex, (2) sets of jobs, represented in the figure by waved lines, which ensure that the vertices to which two adjacent periods correspond are interchanged from one machine to the next, and (3) sets of jobs, represented in the figure by dotted lines, which correspond to the existence of an edge between the vertices corresponding to two diagonally adjacent periods. In what follows we argue that the constructed instance of SPBM corresponds to a given instance of MAX CUT and provide a detailed description of these three components.

To define the set of jobs in the SPBM instance, we distinguish jobs of different types. All jobs travel at unit speed, i.e. each job traverses one unit of distance per unit of time. The two types of jobs are:

1. jobs of *type 1*, arriving on the left side of a specified machine i and travelling towards the right side of this machine i. Each type 1 job thus only requires processing on a single machine. We have $s_j = e_j = i$ and $j \in R$ for all jobs j of type 1.
2. jobs of *type 2*, arriving at a specified machine i; each job of this type may be right-travelling or left-travelling, and requires processing by three machines. For a job $j \in J$ of type 2, we thus have either $s_j = i, e_j = i+2$ and $j \in R$, or $s_j = i, e_j = i-2$ and $j \in L$. In what follows, the travel direction and characteristics will be distinguished as needed.

To construct an instance of dec-SPBM, we use an algorithmic description. This algorithm runs a procedure for each edge $(v_i, v_j) \in E$ where $v_i < v_j$. We initialize by specifying the first machine and next we use the procedure described below. Each odd-numbered machine in the instance has n periods, each corresponding to a vertex in V. A period consists of a time interval on a machine i; the p'th period spans a time interval $[24(p-1), 24p)$. For convenience, we define $t_p = 24(p-1)$, which equals the starting time of the p'th period on each of the machines. Note that, as illustrated in Fig. 18, the p'th period on a machine i does not necessarily correspond to the vertex v_p, and the order in which periods correspond to vertices need not be the same on different machines. The processing time equals $T_i = 2$ and the distances between machines satisfy $x_{i+1} - x_i = 6$ for all machines i in the instance. The machines in M remain ordered from left to right, i.e. as new machines are added to M throughout the procedure, they are added to the right of the existing machines.

Initialization

We start with a single machine: $M = \{1\}$. At this machine 1 we have, for each vertex $v \in V$, $|E|+1$ arrivals of type 1 at each of the times $24(v-1), 24(v-1)+2, \ldots, 24(v-1)+18$.

We define the p'th period on machine 1, i.e. the period on the interval $[24(p-1), 24p)$, to be the period corresponding to vertex p, for all $p \in \{1, \ldots, |V|\}$. For convenience, we also define the value $m^* = 1$, representing the last machine in M at each step in the construction procedure where new machines are added to M. Figure 19 illustrates the arrivals of type 1 in such a period.

As in 'Appendix A', we first highlight two possible ways to schedule a machine to serve the arrivals that arrive within a period on some machine. Let t be the starting time of the period. *Option 1* consists of scheduling a series of consecutive batch operations starting with a right-travelling batch at time t and ending with a right-travelling batch that starts at time $t+20$; *option 2* consists of scheduling a series of consecutive batch operations starting with a right-travelling batch at time $t+2$ and ending with a right-travelling batch that starts at time $t+18$. Observe that if either of these options is used, the jobs of type 1 incur a total waiting time of $10(|E|+1)$. Also notice that options in different periods are independent of each other since there is enough time to ensure that the separation property is satisfied across periods, regardless of which option is chosen for the different periods. For example, selecting option 1 in the period $[24, 48)$ does not prevent us from selecting either option 1 or option 2 in the periods $[0, 24)$ or $[48, 72)$.

For each edge $(v_i, v_j) \in E$, we now describe a procedure that specifies a set of machines, and arrivals at these machines, to be added to the partial instance. This procedure is run for each edge once, and after the final edge the instance is complete.

Procedure repeated for each edge

Consider some edge $(v_i, v_j) \in E$ with $v_i < v_j$. Let i and j be the periods corresponding to v_i and v_j on machine m^*, respectively. We assume that $t_i < t_j$; if this does not hold, we simply swap i and j below. The procedure is as follows.

1. While the periods corresponding to vertices v_i and v_j on machine m^* are not adjacent, repeat
   (a) We add two new machines: let $M \leftarrow M \cup \{m^*+1, m^*+2\}$.

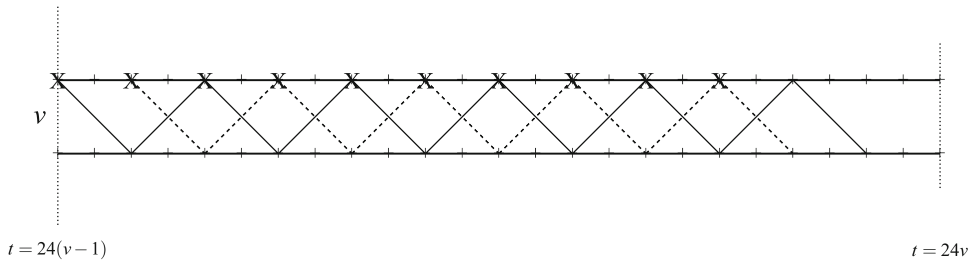


Fig. 19 Illustration of a period on some machine, corresponding to a vertex. Each 'X' marks the arrival of $|E|+1$ type 1 jobs. Solid lines correspond to option 1; dashed lines correspond to option 2

(b) On machine $m^* + 2$, we have periods corresponding to vertices: for each $p \in \{1, \dots, n\}$, we add $|E| + 1$ arrivals of type 1 at each of the times t_p, $t_p + 2$, …, $t_p + 18$.

(c) Let v_k be the vertex to which the period $[t_i + 24, t_i + 48)$ on machine m^* corresponds; note that this is the period following the period corresponding to vertex v_i on machine m^*. Clearly, $v_k \neq v_i$ and $v_k \neq v_j$. We say that, on machine $m^* + 2$, vertex v_k corresponds to period $[t_i, t_i + 24)$, and vertex v_i corresponds to period $[t_i + 24, t_i + 48)$. Notice that, compared to machine m^*, the vertices corresponding to these two periods are interchanged on machine m^*+2. All other vertex-period correspondences remain equal to those on machine m^*.

(d) On machine m^*, we add $|E| + 1$ arrivals of type 2 at each of the times $t_i + 16$ and $t_i + 18$. These jobs are right-travelling and are thus served by machines m^*, $m^* + 1$, and $m^* + 2$. Additionally, on machine $m^* + 2$, we add $|E| + 1$ arrivals of type 2 at each of the times $t_i + 10$ and $t_i + 12$. These jobs are left-travelling and are thus served by machines $m^* + 2$, $m^* + 1$, and m^*. For convenience, we will refer to the jobs added in this step as jobs of type 2a. Notice that these jobs are added in the periods for which the vertex-period correspondence changes from machine m^* to machine $m^* + 2$.

(e) On all remaining periods on machine m^*, i.e. all periods $[t_k, t_k+24)$ for which $t_k \neq t_i$ and $t_k \neq t_i+24$, we add $|E|+1$ arrivals of type 2 at both time t_k and time t_k+2. For convenience, we will refer to the jobs added in this step as jobs of type 2b. Notice that these jobs are added in the periods for which the vertex-period correspondence remains the same from machine m^* to machine $m^* + 2$.

(f) We update m^* so that it again refers to the latest added machine in the instance. That is, we set $m^* \leftarrow m^*+2$. Further, we redefine i and j to be the periods corresponding to v_i and v_j on the new last machine m^*; we adjust t_i and t_j accordingly.

2. Observe that the periods corresponding to v_i and v_j are now adjacent on machine m^*. We add two additional machines: let $M = M \cup \{m^* + 1, m^* + 2\}$. On machine m^*+2, we again have periods corresponding to vertices: for each $p \in \{1, \dots, n\}$, we add $|E| + 1$ arrivals of type 1 at each of the times t_p, $t_p + 2$, …, $t_p + 18$. The vertex-period correspondence on machine $m^* + 2$ is the same as the vertex-period correspondence on machine m^*.
3. For each $p \in \{1, \dots, n\}$, we add $|E| + 1$ right-travelling jobs of type 2 at each of the times t_p and t_p+2 on machine m^*. We refer to the jobs added in this steps as jobs of type 2b.
4. Finally, we add arrivals corresponding to the edge (v_i, v_j). We add a single right-travelling job of type 2 on machine m^* at each of the times $t_i + 12$ and $t_i + 14$. We refer to these arrivals as jobs of type 2c. Additionally, we add $|E| + 1$ arrivals of type 1 on machine $m^* + 1$ at each of the times $t_i + 21$ and time $t_i + 23$.

This concludes the formal description of an instance of SPBM corresponding to a given instance of MAX CUT. Figure 18 shows the structure of an example where the procedure is applied for an edge (v_1, v_4). In the figure, release times of jobs and the even-numbered machines are not shown. Upon completing the construction of this instance, let N_1, N_{2a}, and N_{2b} equal the total number of jobs of type 1, type $2a$, and type $2b$, respectively. Note that both the total number of jobs and the number of machines are polynomial in the size of the original instance G. Indeed, for each edge in E, at most $|V|$ interchange operations are performed: we extend the construction with $O(|V|)$ machines for each edge. On each machine, at most $O(|V||E|)$ jobs are added. The total number of machines is thus bounded by $O(|V||E|)$; the total number of jobs is bounded by $O(|V|^2|E|^2)$. The decision question to be answered in the corresponding instance of dec-SPBM is the following. 'Does there exist a solution with a total waiting time of at most $W \equiv N_1 + N_{2a} + N_{2b} + 10|E| - 4K$?'

Correspondence of MAX CUT to dec-SPBM

We first state the foundations of the argument which shows the correspondence between the given instance of MAX CUT and the constructed instance of dec-SPBM. Notice that, on each odd-numbered machine, we have a period corresponding to each vertex. We will argue that we can restrict ourselves to solutions of SPBM where all periods are scheduled using either option 1 or option 2 and, moreover, all periods corresponding to the same vertex are scheduled using the same option. The selected option then indicates one of two possible partitions of V to which a vertex is assigned, thus defining a cut in the given graph G. We now proceed by providing a detailed overview of the different arrivals added throughout the construction, and the waiting time incurred by these arrivals depending on the chosen option for the different periods.

Figure 19 gives a detailed representation of a period $[24(v - 1), 24v)$ on some machine, corresponding to some vertex. Recall that, in each such period, we have $10(|E| + 1)$ arrivals of type 1 and that, if either option 1 or option 2 is used in this period, the total waiting time incurred by these jobs equals $10(|E| + 1)$.

Step 1(d) adds jobs of type 2a, corresponding to the waved lines in Fig. 18; these arrivals are added where the vertex-period correspondence of two adjacent periods is interchanged from some machine m^* to machine $m^* + 2$.

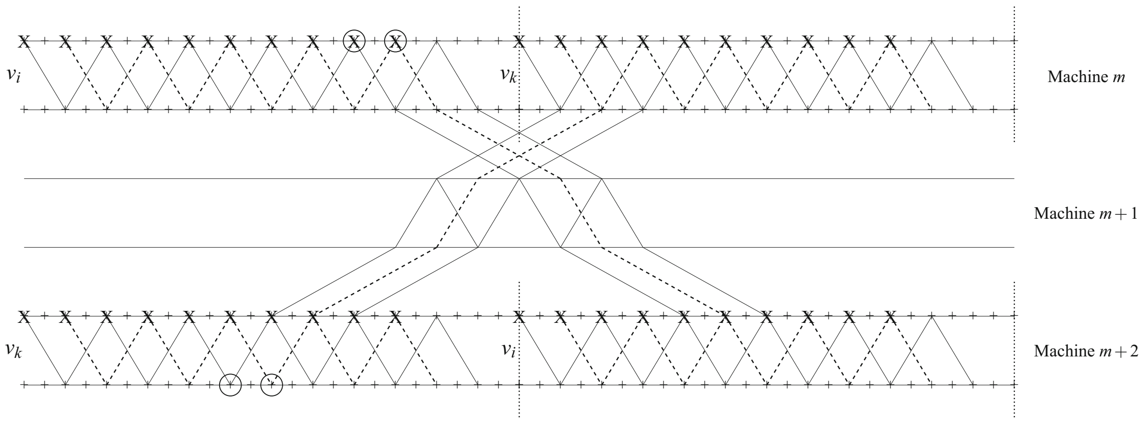


Fig. 20 Construction corresponding to the interchange of two adjacent periods. Each 'X' marks the arrival of $|E| + 1$ jobs of type 1. Each circle marks the arrival of $|E| + 1$ jobs of type 2a

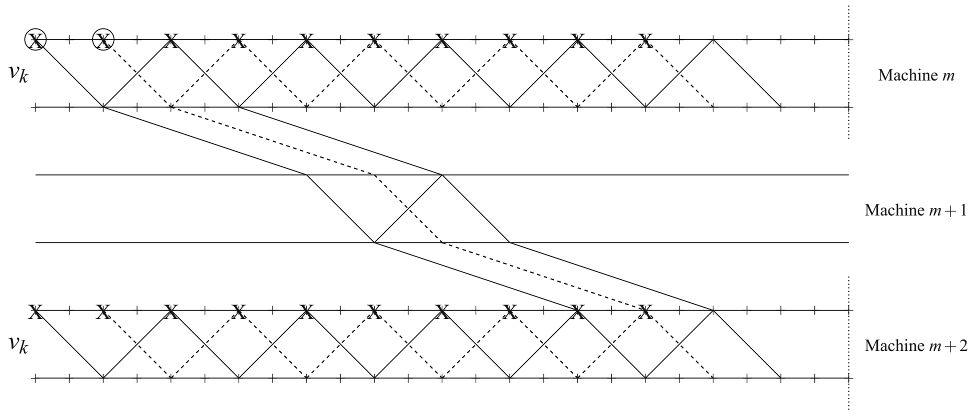


Fig. 21 Construction corresponding to two periods that occupy the same time interval and correspond to the same vertex. Each 'X' marks the arrival of $|E| + 1$ type 1 jobs. Each circle marks the arrival of $|E| + 1$ type 2b jobs

A detailed representation depicting all arrivals in the corresponding periods is shown in Fig. 20. Observe that if either option 1 or option 2 is chosen for each period, although jobs travelling in opposite direction cross in between machines m^* and $m^* + 1$, no batches overlap regardless of the chosen option for the periods. Further observe that, if the same option is chosen for the two periods corresponding to v_i, a total waiting time of $2(|E|+1)$ is incurred by the $2(|E|+1)$ jobs of type 2a arriving on machine m^*; similarly, if the same option is chosen for the two periods corresponding to v_k, a total waiting time of $2(|E|+1)$ is incurred by the $2(|E|+1)$ jobs of type 2a arriving on machine $m^* + 2$. If different options are used for either the two periods corresponding to v_i (respectively, v_k), the total waiting time for the jobs of type 2a arriving on machine m^* (respectively, m^*+2) equals at least $6(|E|+1)$.

Step 1(e) and step 3 add jobs of type 2b, corresponding to the dashed lines in Fig. 18; these jobs travel between a period on a machine m^* and a period on machine m^*+2 that occupy the same time interval and correspond to the same vertex. Figure 21 shows a detailed representation depicting all arrivals in the corresponding periods. Observe that if the same option is chosen for the two periods corresponding to v_k, a total waiting time of $2(|E| + 1)$ is incurred by the jobs of type 2b; if different options are used for the two periods, the waiting time equals at least $6(|E| + 1)$.

Step 4 adds jobs of type 2c, corresponding to the dotted lines in Fig. 18. A detailed representation of this construction is shown in Fig. 22. Observe that if two different options are used for the periods corresponding to v_i and v_j on machine m^* and $m^* + 2$, the jobs of type 2c incur a total waiting time of at least 6 time units. In contrast, if the same option is used for both periods, the total waiting time for the jobs of type 2c equals at least 10 time units. Also notice that on the even-numbered machine $m^* + 1$, in order to achieve this waiting time, a right-travelling batch must be scheduled at times $t_i + 21$ and $t_i + 25$ if option 1 is selected for the period corresponding to v_i, and a right-travelling batch must be scheduled at time $t_i + 23$ if option 2 is selected for the period corresponding to v_i. The arrivals of type 1 on the intermediate machine then incur a total waiting time of $2(|E|+1)$. Note that whenever these arrivals of type 2c are added to a period, this period also has arrivals of type 2b as described by step 5 of the construction. There is, however, no overlap in the trajectory of the jobs of types 2b and 2c, nor is there

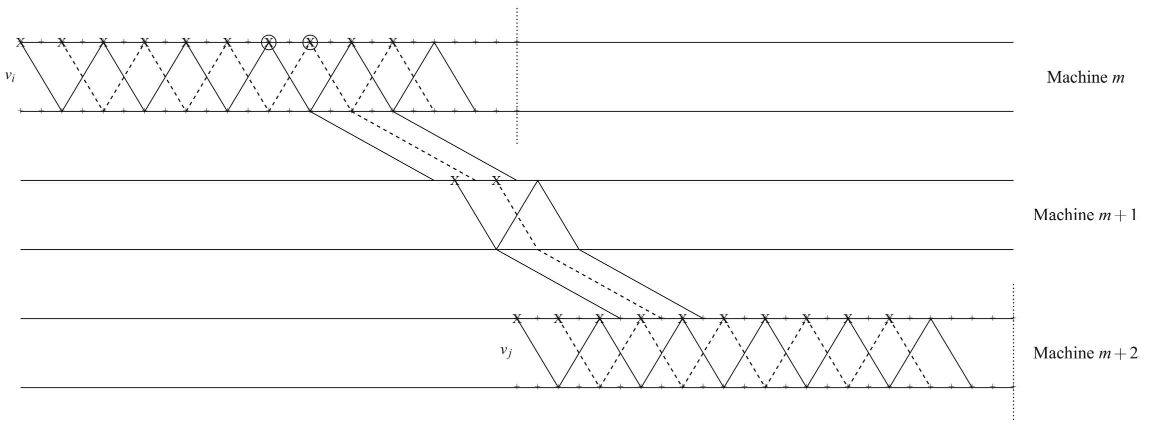


Fig. 22 Construction corresponding to an edge (v_i, v_j). Each 'X' marks the arrival of $|E|+1$ type 1 jobs. Each circle marks the arrival of a single type 2c job

an overlap between the batches serving these jobs in Figs. 21 and 22.

We are now ready to argue that a cut in graph G containing at least K edges exists if and only if a solution exists for the instance of dec-SPBM with a waiting time of at most W.

$\Rightarrow$ Assume that there exists a cut in G that contains at least K edges; the corresponding partition of the vertices is indicated by V_1 and V_2. We build the following solution for the instance of dec-SPBM. If vertex $v \in V_1$, then we use option 1 for all periods corresponding to vertex v; if vertex $v \in V_2$, then we use option 2 for all periods corresponding to vertex v; each job enters the first available batch corresponding to its direction of travel, and the batches for all even-numbered machines are scheduled such that jobs of type 2c incur a waiting time of 1 time unit at these machines, as illustrated in Fig. 22. We claim that the resulting waiting time of all jobs is bounded by W.

We first identify the total waiting time for the jobs of types 1, 2a, and 2b. Recall that either option 1 or option 2 is chosen for each period in the instance.

- The total waiting time for jobs of type 1 equals the total number of type 1 arrivals. Indeed, arrivals of type 1 arrive either (i) in a period corresponding to a vertex (Fig. 19), or (ii) on an even-numbered machine where two adjacent periods are connected to model an edge in E (Fig. 22). In both of these cases, the total waiting time incurred by type 1 jobs equals the number of type 1 arrivals.
- The total waiting time for jobs of type 2a equals the total number of type 2a arrivals. Indeed, arrivals of type 2a are added only where two periods occupy the same time interval and correspond to the same vertex (Fig. 21). As a result, the same option is chosen for these two periods, and the total waiting time incurred by type 2a jobs equals the number of type 2a arrivals.
- The total waiting time for jobs of type 2b equals the total number of type 2b arrivals. Indeed, arrivals of type 2b are added only where the vertices to which two periods correspond on a machine m^* are interchanged on machine m^*+2 (Fig. 20). As a result, the same option is chosen for each pair of periods corresponding to the same vertex, and the total waiting time incurred by type 2b jobs equals the number of type 2b arrivals.

For these jobs, this yields a total waiting time of $N_1 + N_{2a} + N_{2b}$.

Now consider the jobs of type 2c. Observe that for each edge $(v_i, v_j) \in E$, there are exactly two jobs of type 2c. If edge (v_i, v_j) is in the cut, i.e. if $v_i \in V_1$ and $v_j \in V_2$ or vice versa, the options used for periods v_i and v_j differ on the machines traversed by these jobs. This results in a waiting time of 6 corresponding to this edge; otherwise, there is a waiting time of 10. Since there exists a cut of K edges, we thus obtain a total waiting time of $6K + 10(|E| - K) = 10|E| - 4K$ for the jobs of type 2c. The total waiting time for the corresponding solution of SLS equals $N_1 + N_{2a} + N_{2b} + 10|E| - 4K = W$. A yes-instance of MAX CUT thus gives rise to a yes-instance of dec-SPBM.

$\Leftarrow$ Consider now a solution to an instance of SLS with a total waiting time of at most W. First, we argue that we can assume that such a solution is so-called sensible. We say that a solution is *sensible* if:

- Condition 1: after arriving at a machine, each job enters the first available batch corresponding to its direction of travel,
- Condition 2: in each period corresponding to a vertex, either option 1 or option 2 is used,
- Condition 3: each even-numbered machine is scheduled such that jobs of type 2c traversing that machine incur a waiting time of 1.

We argue that we can restrict ourselves to considering sensible solutions only.

Lemma 3 *For any feasible solution to SPBM with waiting time W', there exists a sensible solution to SPBM with waiting time at most W'.*

Proof Notice that Conditions 1 and 2 are identical to the definition a sensible solution used in Lemma 2 in the context of jobs with arbitrary speed. The argument that these conditions can be guaranteed, without increasing the total waiting time of a solution, can be repeated from the proof of Lemma 2.

To see that Condition 3 can be enforced without increasing the total waiting time, we again make use of the fact that there is an optimum solution where, for each machine m^*, each batch operation starts at a point in time where a job arrives at m^* or follows immediately upon the completion of an earlier batch operation of that machine m^*. It is then clear that for the corresponding time t_i in the step in the construction where jobs of type 2c are added, visualized in Fig. 22, the type 1 jobs that arrive at the even-numbered machine $m^* + 1$ can be served either by (i) a right-travelling batch starting at time $t_i + 20$ and a right-travelling batch starting at time $t_i + 24$, (ii) a right-travelling batch starting at time $t_i + 21$ and a right-travelling batch starting at time $t_i + 25$, (iii) a single right-travelling batch starting at time $t_i + 22$, or (iv) a single right-travelling batch starting at time $t_i + 23$. Observe that the schedule of the even-numbered machine only determines the total waiting time of jobs of type 1 arriving on the machine and the total waiting time of jobs of type 2c. It can be seen that if option 1 is selected for the period corresponding to v_i, case (i) or (ii) must hold in any optimum solution; if option 2 is selected, case (iii) or (iv) must hold in any optimum solution. If case (i) or (iii) holds, it is not difficult to see that, by selecting option (ii) or (iv), respectively, the total waiting time for jobs of type 1 reduces by $2(|E| + 1)$ time units, whereas it increases for the jobs of type 2c by at most 6 time units. Clearly we may assume $|E| \geq 2$ since instances with $|E| = 1$ are trivial to solve. Case (ii) or case (iv) above must then hold in any optimum schedule; it follows that all jobs of type 2c then incur a single unit of waiting time at an even-numbered machine. □

We may thus assume that the given solution for SPBM is a sensible solution with a total waiting time bounded by W. We argue that the instance of MAX CUT is then a yes-instance. We first claim that all periods corresponding to any given vertex $v \in V$ must use the same option. We argue by contradiction. Recall that the total waiting time in any sensible schedule equals at least $N_1 + N_{2a} + N_{2b} + 6|E|$ and that an additional waiting time of $4(|E| + 1)$ is incurred for every vertex v for which two corresponding periods are scheduled with different options. Assume that there is a single vertex for which this is the case. It follows that the total waiting time must be equal to at least $N_1 + N_{2a} + N_{2b} + 10|E| + 4 > N_1 + N_{2a} + N_{2b} + 10|E| - 4K = W$. This contradicts the fact that our solution for dec-SPBM has a waiting time of at most W. Thus, all periods corresponding to some vertex $v \in V$ are scheduled using the same option. For any sensible schedule with a total waiting time no greater than W, it then follows that we have a solution where the total waiting time consists of:

1. $N_1 + N_{2a} + N_{2b}$, incurred by jobs of type 1, type 2a, and type 2b,
2. a waiting time of 6 time units for every edge $(v_i, v_j) \in E$ where different options are chosen for the periods corresponding to v_i and v_j, incurred by jobs of type 2c,
3. a waiting time of 10 time units for every edge $(v_i, v_j) \in E$ where the same option is chosen for the periods corresponding to v_i and v_j, incurred by the remaining jobs of type 2c.

Given a solution to SPBM with a total waiting time of at most $W = N_1 + N_{2a} + N_{2b} + 10|E| - 4K$, we construct a solution to MAX CUT by assigning a vertex v to V_1 (respectively, V_2) if option 1 (option 2) is used for the periods corresponding to vertex v. It follows that there are at least K pairs of vertices (v_i, v_j) such that the periods corresponding to v_i are scheduled with a different option than the periods corresponding to v_j. Indeed, if at most $K - 1$ pairs use different options, the total waiting time must equal at least $N_1 + N_{2a} + N_{2b} + 6(K-1) + 10(|E| - (K-1)) > W$. By construction, there are thus at least K pairs of vertices v_i and v_j with $v_i < v_j$ and $(v_i, v_j) \in E$ such that exactly one of these vertices is in V_1 and the other is in V_2. Thus, there are at least K edges in the resulting cut in G. A yes-instance of dec-SPBM thus gives rise to a yes-instance of MAX CUT, which completes our reduction.

We again remark that this construction, like the proof described in 'Appendix A', can be modified so that each machine has unit capacity and arrivals occur at distinct times. We omit a formal description for this modified setting.

References

Adler, M., Sitaraman, R. K., Rosenberg, A. L., & Unger, W. (1998). Scheduling time-constrained communication in linear networks. In *Proceedings of the tenth annual ACM symposium on parallel algorithms and architectures, SPAA '98* (pp. 269–278). New York, NY: ACM.

Antoniadis, A., Barcelo, N., Cole, D., Fox, K., Moseley, B., Nugent, M., & Pruhs, K. (2014). Packet forwarding algorithms in a line network. In A. Pardo, & A. Viola (Eds.), *LATIN 2014: Theoretical informatics: 11th Latin American symposium, Montevideo, Uruguay, March 31–April 4, 2014. Proceedings* (pp. 610–621). Berlin: Springer.

Brucker, P., Knust, S., Cheng, T. E., & Shakhlevich, N. V. (2004). Complexity results for flow-shop and open-shop scheduling problems with transportation delays. *Annals of Operations Research*, *129*(1), 81–106.

Disser, Y., Klimm, M., & Lübbecke, E. (2015). Scheduling bidirectional traffic on a path. *Computing Research Repository* abs/1504.07129.

Frank, O. (1966). Two-way traffic on a single line of railway. *Operations Research*, *14*(5), 801–811.

Gafarov, E. R., Dolgui, A., & Lazarev, A. A. (2015). Two-station single-track railway scheduling problem with trains of equal speed. *Computers and Industrial Engineering*, *85*, 260–267.

Garey, M. R., Johnson, D. S., & Stockmeyer, L. (1976). Some simplified NP-complete graph problems. *Theoretical Computer Science*, *1*(3), 237–267.

Passchyn, W. (2016). *Scheduling locks on inland waterways*. Ph.D. thesis, KU Leuven.

Passchyn, W., Briskorn, D., & Spieksma, F. C. R. (2016a). Mathematical programming models for lock scheduling with an emission objective. *European Journal of Operational Research*, *248*(3), 802–814.

Passchyn, W., Coene, S., Briskorn, D., Hurink, J. L., Spieksma, F. C. R., & Vanden Berghe, G. (2016b). The lockmaster's problem. *European Journal of Operational Research*, *251*(2), 432–441.

Petersen, E. R., & Taylor, A. J. (1988). An optimal scheduling system for the Welland Canal. *Transportation Science*, *22*, 173–185.

Prandtstetter, M., Ritzinger, U., Schmidt, P., & Ruthmair, M. (2015). A variable neighborhood search approach for the interdependent lock scheduling problem. In G. Ochoa, & F. Chicano (Eds.), *Evolutionary Computation in Combinatorial Optimization, volume 9026 of Lecture Notes in Computer Science* (pp. 36–47). Berlin: Springer.

Righini, G. (2016). A network flow model of the northern Italy waterway system. *EURO Journal on Transportation and Logistics*, *5*(2), 99–122.

Smith, L. D., Nauss, R. M., Mattfeld, D. C., Li, J., Ehmke, J. F., & Reindl, M. (2011). Scheduling operations at system choke points with sequence-dependent delays and processing times. *Transportation Research Part E*, *47*, 669–680.

Smith, L. D., Sweeney, D. C., & Campbell, J. F. (2009). Simulation of alternative approaches to relieving congestion at locks in a river transportation system. *Journal of the Operational Research Society*, *60*, 519–533.

Verstichel, J. (2013). *The lock scheduling problem*. Ph.D. thesis, KU Leuven.

Verstichel, J., & Vanden Berghe, G. (2016). Scheduling serial locks: A green wave for waterbound logistics. In M. Lu & J. De Bock (Eds.), *Sustainable logistics and supply chains: Innovations and integral approaches* (pp. 91–109). Berlin: Springer.

Zhao, Z. J., Lau, H. C., & Ge, S. S. (2009). Integrated resource allocation and scheduling in a bidirectional flowshop with multimachine and COS constraints. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, *39*(2), 190–200.