Line Side Placement for Shorter Assembly Line Worker Paths

Helmut A. Sedding

Institute of Theoretical Computer Science, Ulm University, Ulm, Germany

Abstract

Placing material containers at moving assembly lines is an intriguing problem because each container position influences worker paths. This optimization is relevant in practice as worker walking time accounts for about 10–15% of total work time. Nonetheless, we find few computational approaches in the literature. We address this gap and model walking time to containers; then optimize their placement. Our findings suggest this reduces walking time of intuitive solutions by an average of 20%, with considerable estimated savings. To investigate the subject, we formulate a quintessential optimization model for basic sequential container placement along the line side. However, even this core problem turns out as strongly NP-complete. Nonetheless, it possesses several polynomial cases that allow to construct a lower bound on the walking time. Moreover, we discover exact and heuristic dominance conditions between partial placements. This facilitates an exact and a truncated branch and bound solution algorithm. In extensive tests, they consistently deliver superior performance compared to several mixed integer programming and metaheuristic approaches. To aid practitioners in quickly recognizing instances with high optimization potential even before performing a full optimization, we provide a criterion to estimate it with just few measurements.

KEYWORDS

Assembly Line; Line Side Placement; Walking Time; Moving Conveyor; Time-dependent Scheduling

1. Introduction

A key component for improving the productivity of manufacturing plants is the elimination of nonproductive time. In automotive final assembly, worker walking times are a significant contributive factor: Scholl *et al.* (2013) indicate that about 10–15% of total production time at a major German car manufacturer is spent on fetching parts from the line side. We aim to minimize this nonproductive work time by optimizing the line side placement of parts. The main difficulty arises from a moving conveyor which induces time-dependency for walk distances to the line side.

In this work, we approach this with a model that captures the quintessence of planning the line side part placement such that it takes time-dependent walking time into account.

This is an Accepted Manuscript of an article published by Taylor & Francis in IISE Transactions 52(2), on Feb 1, 2020, available at http://wwww.tandfonline.com/10.1080/24725854.2018.1508929

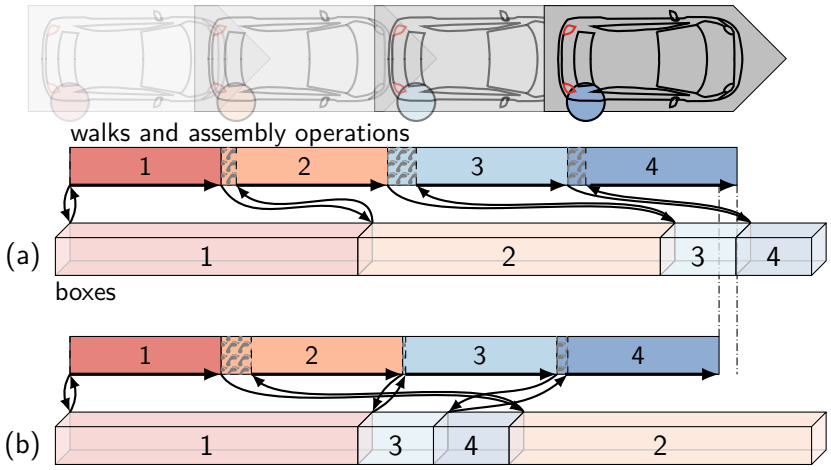

Figure 1. This figure shows an example assembly station for the right rear wheel. The conveyor line is shown at four points in time; each is at the start of an assembly operation (marked with a dashed line). Before each operation, the worker walks to pick up required parts from the corresponding box at the line side. An intuitive placement strategy is to order the boxes in job sequence $\langle 1, 2, 3, 4\rangle$, as shown in (a). However, the resulting walking time is over 40% higher compared to the optimum placement in (b) with box sequence $\langle 1, 3, 4, 2\rangle$. Clearly, it is better to accept a longer distance for operation 2; it results in much shorter distances for 3 and 4.

We are given n part boxes, each with a certain width. They are placed side-to-side in a single row along the assembly line. Each box contains all the parts required for exactly one assembly operation. Hence, we are also given a list of n assembly operations, each with a specific assembly time. The operations are processed by one assembly worker in a fixed sequence. For each assembly operation, he or she leaves the workpiece, walks along the assembly line to the corresponding box, picks up the required parts, and returns to the workpiece. We call this a *walk*. Then, the worker performs the assembly operation. The workpiece is situated on a conveyor that moves with constant velocity along a line. Thus, walk distances are time-dependent, but deterministic. To minimize *walking time* (the sum of all walk times), each box should be placed close to the point where its operation is performed. Intuitively, it should suffice to order the boxes in the same sequence as the jobs. Although this is a common practice, it can be better to place certain boxes further away which yields a longer walking time locally, but reduces it globally, like for instance in Figure 1.

For devising a method that optimizes the box sequence, one may turn to existing scheduling literature. However, we are not aware of any existing model that matches the given setting, Therefore, let us at least identify related scheduling problems and their discrepancy to our setting in the following. To minimize the walking time, each box ought to be placed close to the location of its assembly operation. This resembles the classic scheduling objective of completing each job close to its due date, while any deviation increases costs linearly. The corresponding minimization problem is known as the early/tardy scheduling problem (Baker and Scudder, 1990; Józefowska, 2007). It is NP-hard already in its simplest form with cost slopes all equal and one common

due date (Hall *et al.*, 1991). With distinct due dates, it becomes NP-hard in the strong sense (Wan and Yuan, 2013). However, there is a clear shortcoming of this model because, in our given application, it is infeasible to assume constant due dates: As each walk influences the position of succeeding operations, the due dates need to change accordingly. Hence, due dates are impacted by the job sequence. There is literature on variable due dates, also in combination with the early/tardy objective (Cheng and Gupta, 1989; Gordon *et al.*, 2002a,b, 2004; Kaminsky and Hochbaum, 2004; Shabtay, 2016). Nonetheless, the closest relation we see is the quotation of due dates in dependence of job processing times. In our case however, each due date should depend both on the actual start time of the job, and the preceding due date (to leave enough space for placing the box), and we are not aware of any literature taking this into account.

A second perspective on the placement problem is to regard assembly operations as jobs with processing times that depend on start time (Alidaee and Womer, 1999; Cheng *et al.*, 2004a; Gawiejnowicz, 2008). In particular, there are studies on processing time functions that depend on the absolute difference between start time and due date (Farahani and Hosseini, 2013; Jaehn and Sedding, 2016). However, their due date value is fixed. This is relieved with variable due dates (Cheng *et al.*, 2004b; Gordon *et al.*, 2012; Yin *et al.*, 2013). But in this stream of work, processing time functions are independent of due dates. Furthermore, it is not ensured that there is enough space after each due date for placing the corresponding box. Moreover, these models allow to permute the jobs as well, although their sequence is fixed in our case. Hence, this literature is of limited use for the problem at hand. Nonetheless, we find use of their mindset in modeling and solving it.

On the applied side, literature on assembly line optimization provides a rich body of work leading up to line side placement optimization. To begin with, most notable is the assembly line balancing problem as it belongs to one of the first described mathematical optimization problems (Salveson, 1955). Its objective is to distribute work equally among workers. As each worker's available time is limited, it generalizes the bin-packing problem (Wee and Magazine, 1982). Surveys on this topic are found in Battaïa and Dolgui (2013); Baybars (1986); Scholl and Becker (2006). In practice, a multitude of additional aspects need to be considered. Assembly of a workpiece often requires parts from containers at the line side. However, available space is scarce. Therefore, Bautista and Pereira (2007) introduce space constraints for each workstation, while space utilization is optimized in Bukchin and Meller (2005) to reduce line stoppage from late part replenishments. As stops also happen if an assembly worker's workload repeatedly exceeds the available time, it is essential to anticipate and plan each worker's workload properly. To this end, the models in Andrés *et al.* (2008) and Scholl *et al.* (2013) factor in sequence-dependent work times for, e.g., fetching parts. However, these models assume fixed walking distances. These are inaccurate for moving assembly lines, which transport a workpiece continuously on a conveyor. The resulting time-dependency of walk distance is modeled in Jaehn and Sedding (2016). They minimize each worker's

total walking time by sequencing assembly operations according to a given line side part placement. However, an optimization of the latter allows to attain a significant reduction of walking time as well. A walk distance reduction by 52% is, e.g., achieved through manual optimization in the case study in Finnsgård *et al.* (2011). A first operational research approach for changing the line side part placement to minimize walking time is described in Klampfl *et al.* (2006). However, they report rather long computation times even for a small case study of just five part containers. Although manufacturers demand sophisticated, intelligent decision support systems in this area, we are not aware of further computational approaches in the literature. This research gap is moreover explicitly identified in Boysen *et al.*'s (2015) recent review on automotive part logistics.

In this work, we address this gap by introducing a core model for line side part placement with time-dependent walk times, analyze properties and construct tailored optimization methods. In Section 2, we describe the setting and our model assumptions that lead to a formal definition of the placement problem. Its applicability is described for several walking strategies of walking atop and beside the conveyor. We highlight polynomial cases of this model in Section 3 and study its computational complexity in Section 4. Section 5 exploits these properties and introduces, for partially solved problems, a lower bound on the objective. An exact, and a heuristic dominance rule allow to compare partial solutions and eliminate dominated ones in Section 6. These results are combined in a branch and bound algorithm in Section 7, from which we deduce a heuristic version. We empirically evaluate their performance against several heuristics and an integer programming approach in a numerical experiment in Section 8. In Section 9, we evaluate the average savings potential compared to using the intuitive placement strategy of ordering boxes in job sequence, and give a criterion that allows practitioners to easily spot low performing instances. In Section 10, we conclude with a research outlook to spark further work in the problem area.

2. Modeling

2.1. Formal Definition

In our formal definition, we use the term *job* for referring to an assembly operation in conjunction with its preceding walk. Accordingly, the job's processing time function sums a time-dependent walking time and an assembly time. However in contrast to most scheduling problems, the given job sequence is fixed. Instead, the box sequence is variable. It places the boxes in a row side-to-side. Then, walk times to the boxes can be calculated, although we defer their precise formula until Section 2.3. To minimize total walking time or, equivalently, the last job's completion time, we seek for an optimum box sequence.

Definition 1 (Problem $\mathcal{P}$)**.** We are given a set of n jobs $J = \{1, \dots, n\}$. Each job $j \in J$ is given the assembly time $l_j \in \mathbb{Q}_{\geq 0}$ and, for its corresponding box, a box width $w_j \in \mathbb{Q}_{>0}$.

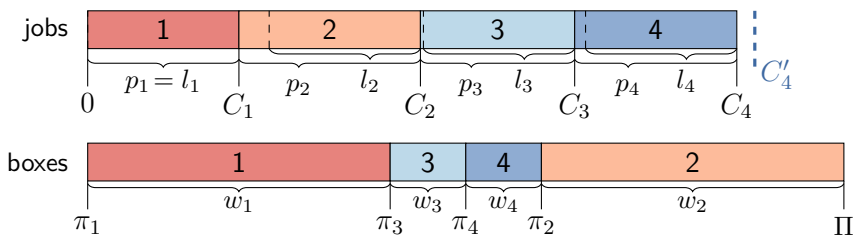

Figure 2. Example instance of Figure 1 in terms of Definition 1: factors $a = b = 0.1$ and $n = 4$ jobs, assembly times $l_1 = \cdots = l_4 = 2$ all equal, $a = b = 0.1$ and $n = 4$ jobs, assembly times $l_1 = \cdots = l_4 = 2$ all equal, and box widths $w_1 = w_2 = 4, w_3 = w_4 = 1$. Optimum box sequence $S = \langle 1, 3, 4, 2 \rangle$ reaches $C_4 = 8.584$, the intuitive $S' = \langle 1, 2, 3, 4 \rangle$ yields $C'_4 = 8.822$.

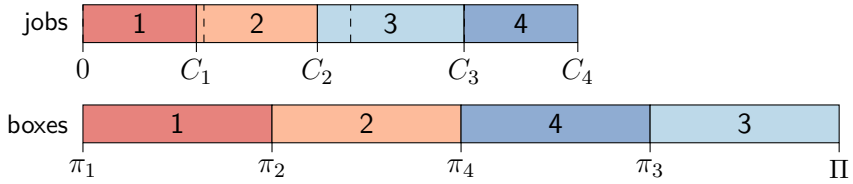

Figure 3. Example instance with $a = b = 0.1$ and $n = 4$ jobs, assembly times $l_1 = \cdots = l_4 = 3$ all equal, and box widths $w_1 = \cdots = w_4 = 5$ all equal, has the optimum box sequence $S = \langle 1, 2, 4, 3 \rangle$.

Total box width is $\Pi = \sum_{j \in J} w_j$. We need to find a placement for the boxes which is defined by a box sequence $S : J \to \{1, \ldots, n\}$ that places the box of a job $j \in J$ at position $\pi_j = \sum_{k \in J,\, S(k) < S(j)} w_k$. Iteratively, for each job $j \in J$, we calculate start time $t_j = C_{j-1}$, with $C_0 = 0$, and completion time $C_j = C_j(t_j)$ with $C_j(t) = t + p_j(t)$, start time dependent processing time $p_j(t) = l_j + \varpi_j(t)$, and time-dependent walk time $\varpi_j(t)$. This yields the last completion time C_n. The objective is to find a box sequence S that minimizes $\phi = C_n$.

In Figure 2, we visualize the example instance from Figure 1 in terms of the formal definition. In each job, the dashed line indicates the return from walking. As box 1 is placed first, the walk time for job 1 is zero, thus $p_1 = l_1$. All other jobs $j > 1$ begin with a nonzero walk time, hence $p_j > l_j$. A second example in Figure 3 has all-equal assembly times and all-equal box widths. Finding an optimum box sequence is, however, difficult because repositioning a box j impacts the processing time of all the jobs $k \geq j$ in the job sequence. In turn, new start times of these jobs require a reoptimization of all the corresponding boxes $k > j$. Although the problem setting might appear basic, it adequately models a real world problem, see Section 2.2. Nonetheless, the underlying structure is nontrivial and the problem is NP-hard in the strong sense as we show in Section 4.

2.2. Assumptions

With our model, we aim for a close depiction of reality, as well as for a subset of further practice settings in order to gain generic insights. In the following, we dissect the practice situation into several aspects, and denote how they are depicted in our model.

A1 (Single station)**.** In practice, an assembly line is divided into a row of equal-sized, separately operated stations. As the station's length equals the distance between succeeding workpieces, it contains exactly one workpiece at all times. Moreover, the station's line side is not shared with other stations. Therefore, we focus on one work station and one workpiece.

A2 (Single worker)**.** In practice, usually one worker is assigned to the same station, although this can be extended to multiple workers (Becker and Scholl, 2009). Therefore, our model considers operations and placement area of one worker. However, our model can even depict the multiple worker case by assuming no interference occurs and each worker's material is placed on a separate, contiguous area.

A3 (Single product variant)**.** In practice, a production-mix of multiple variants is nowadays the norm. This is either achieved by placing a separate part container for each variant, or by kitting parts of several variants in the same container. Kitting effectively requires to consider a single model only, because the same part container is used in every variant. As kitting is a common strategy in practice, we decide to focus on this variant for our model. Hence, there is only one list of operations and no further containers.

A4 (Fixed operation sequence)**.** In practice, workers may autonomously change the order of some operations. We abstain from predicting this and assume an immutable list of operations.

A5 (Single cycle)**.** A cycle starts when the workpiece enters, and ends when it leaves the station. In practice, a worker might finish a cycle early or late depending on product variations and his or her current condition. Thus, he or she might float up- or downstream the conveyor line. However, this is hard to predict. Therefore, we focus on average assembly times and disregard floating. Hence, we only model a single cycle. Then, each operation always happens at the same time and takes the same time.

A6 (Single work point)**.** In practice, larger workpieces have several work points with significant walking distance in between. However, such operations are usually marked incompatible during line balancing and thus, they get assigned to different workers anyhow (Becker and Scholl, 2009). Therefore, in most cases all assembly work happens at one spatial point of workpiece on the moving conveyor, which is depicted in our model.

A7 (One box per operation)**.** In practice, an operation sometimes requires parts of several containers. However, even then it is advisable to store them side-by-side by to avoid further walking time. We assume that this is the case and therefore subsume these containers by an imaginary box that encompasses them. This also comprises stacked containers in a rack or on a dolly.

A8 (One-dimensional box placement)**.** In practice, larger containers are placed on pallets or dollies on the floor, while smaller containers are placed in racks along the line. Our model indeed allows stacking of containers within a box as long as they correspond to the same operation. Then, placement reduces to a single row of boxes at the line side.

A9 (No space between boxes)**.** In practice, there is commonly a lack of space at the line side. Therefore, we disallow space between boxes entirely in our model and place boxes side-to-side without spacing.

A10 (Stationary box positions)**.** In practice, container positions are not changed during production, even if larger containers are sometimes on wheels or mounted on dollies or overhead cranes. Accordingly, we assume fixed positions, and optimize them offline.

A11 (Uniform walking velocity)**.** In practice, a worker requires time for acceleration and deceleration. Moreover, heavier parts reduce the velocity. In this model, we approximate walk times by using a constant average velocity for all parts. Then, walking time is linear function of the distance.

A12 (One-dimensional walking)**.** In practice, containers are placed as close to the conveyor as possible to reduce walking time. Therefore, we assume that the containers are within gripping distance. As a result, the worker only walks in parallel to the conveyor belt. Hence, we calculate walking time by measuring distance in just one dimension along the line.

A13 (One walk per operation)**.** In practice, parts of an operation are fetched in one walk just before the operation's start. Hence, we model a walk at the start of each operation. After that, each assembly time is constant. Additionally, the worker may bring along parts for other operations as well. These parts are placed close to the other parts by to avoid further walking, hence they can be subsumed by the same box. If, following that, one of the corresponding operations no longer requires a separate walk, we join it with its preceding operation into one longer operation. Hence, it again suffices to model one walk per operation. With this approach, it is also possible to depict gathering all small parts like screws in the walk of, e.g., the first operation, and additional larger parts in later walks. If an operation requires no parts at all, we also append it to its preceding operation, or, if it is the first, adjust the worker's start time.

A14 (No pick time)**.** In practice, the case study in Finnsgård *et al.* (2011) reports that the picking time accounts for only 6% of the nonproductive time (mean picking time 1.6 seconds, mean walking time 26.4 seconds). Therefore, we ignore pick times.

A15 (Picking at upstream side)**.** In practice, depending on the spatial arrangement of parts within a box, the pick point at larger boxes may vary. We eschew modeling such detail and thus settle on picking at a single point, the upstream (left) side of the box.

2.3. Walking Strategies

A one-dimensional walk distance measurement suffices to model walking time along a moving conveyor line (see assumption A12). Moreover, we show in the following that walking time can be calculated by just a piecewise linear function of two pieces.

The conveyor moves linearly along the assembly line with constant velocity. Distance is a size that can be measured by the amount of time it takes the conveyor to travel it. Indeed, we let all measurements base on conveyor velocity $v_{\text{conv}} = 1$, and scale all distance measures accordingly to this unit (for example, box widths). Moreover, we equate the workpiece's position with time: at time t, the workpiece is at $v_{\text{conv}}t = t$. Furthermore, we express a box position by the time it is passed by the wokpiece.

For the worker, we assume a constant walking velocity $v > v_{\text{conv}}$ (see assumption A11). Let us estimate, for some job $j \in J$, the worker's walk time from the workpiece to box position π_j and back if the walk starts at t. If $t = \pi_j$, the walk time is zero. Else, we distinguish if

(a) the workpiece moves toward the box: $t < \pi_j$; or
(b) the workpiece moves away from the box: $t > \pi_j$.

In each case, walk time is proportional to distance $\pi_j - t$. Thus, walk time is calculated by

$$\varpi_j(t) = \max\{a\,(\pi_j - t)\,,\, b\,(t - \pi_j)\}, \tag{1}$$

with a linear factor $a \in (0, 1)$ for case (a), and a linear factor $b \in (0, \infty)$ for case (b).

We show in the following that by choosing a, b accordingly, it is possible to cover all common walking strategies from reality. If the floor is fixed and just the workpiece moves (as in Klampfl *et al.* (2006)), strategy (A) applies. If the floor plates move together with the workpiece (as in Jaehn and Sedding (2016)), strategy (B) applies. Their combination (C) applies if the worker can freely alternate between a fixed floor and moving floor plates. These strategies are listed below. Additionally, Figure 4 depicts the displacement of the worker, the workpiece, and a box for an example job.

Walking strategy (A). The worker walks beside the conveyor or underneath an overhead conveyor. Here, the walk time if starting at time t is $\varpi_j(t)$ from Equation (1) with factor $a = 2/(v + 1) \in (0, 1)$ and $b = 2/(v - 1) \in (0, \infty)$ for worker velocity $v > 1$.

Proof. Here, we fix the coordinate system on the floor. The target box has the constant position $f(t) = \pi_j$. Let us consider case (a). To walk to the box, starting at time $\hat{t}$ at position $\hat{t}v_{\text{conv}} = \hat{t}$, the worker's position is calculated by function $g(t) = (t - \hat{t})v + \hat{t} = tv + \hat{t}\,(1 - v)$ for $t \geq \hat{t}$. Then, the box visit time t' is $g(t') = f(t') \iff t'v + \hat{t}\,(1 - v) = \pi_j \iff t' = \pi_j/v + \hat{t}\,(1 - 1/v)$. For returning, the worker movement function is $h(t) = -(t - t')v + f(t')$, for $t \geq t'$. Then, it meets the conveyor, which is described by $q(t) = t$, at return time t'', which is $h(t'') = q(t'') \iff -(t'' - t')v + \pi_j = t'' \iff t'' =$

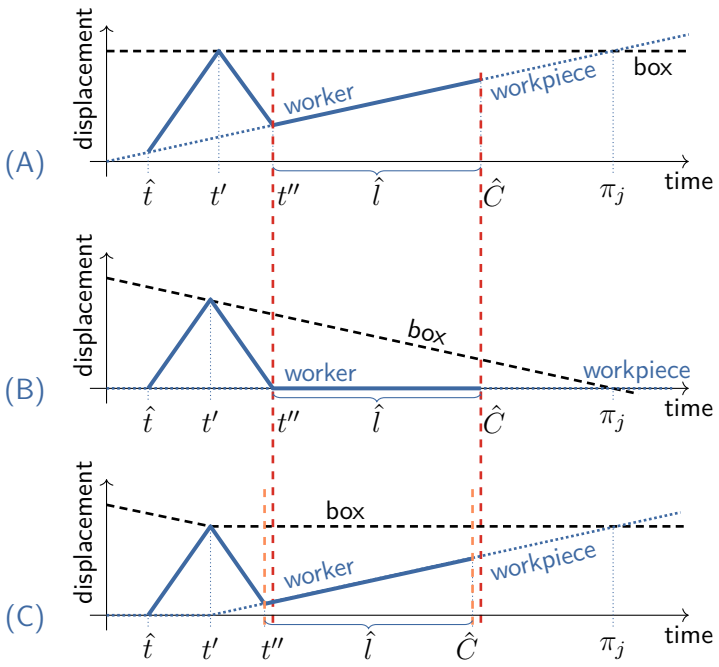

Figure 4. In this example, we show the displacement of the worker during one job (drawn with a solid thick line) in walking strategy (A), (B), and (C). The worker commences with the job by leaving the workpiece (dotted line) at time $\hat{t}$, visits the box at time t', returns to the workpiece at time t'', remains there for assembly time $\hat{l}$, and completes the job at time $\hat{C}$. Additionally, the diagram shows the displacement of the corresponding box (dashed line) to the workpiece on the moving conveyor in the course the job; they pass each other at time $\hat{\pi}$. It is visible that walk time $t''-\hat{t}$ is the same in (A) and (B). However in (C), the walk time is smaller, hence t'' and $\hat{C}$ are smaller as well (as can be seen from the vertical, dashed reference lines).

$(\pi_j + t'v)/(1+v)$. Substituting t', the walk time then is $t''-\hat{t} = (\pi_j-\hat{t})\cdot 2/(1+v) = (\pi_j-\hat{t})\,a$. The walk time thus proportionally depends on the distance between the box and the work point. Case (b) is calculated similarly. Here, the worker's movement is $g(t) = -(t-\hat{t})v+\hat{t} = -tv+\hat{t}\,(1+v)$. Then, t' is $g(t') = f(t') \iff t' = -\pi_j/v+\hat{t}\,(1+1/v)$. Returning, the worker is at $h(t) = (t-t')v + f(t')$, and meets the conveyor at t'', which is $h(t'') = q(t'') \iff t'' = (\pi_j - t'v)/(1-v)$. The walk time in this second case is $t''-\hat{t} = (\pi_j-\hat{t})\cdot 2/(1-v) = (\pi_j-\hat{t})\,b$. Both cases combined yield the stated walk time. □

Walking strategy (B). The worker walks upon the conveyor, which has mounted floor plates. As in strategy (A), walk time is $\varpi_j(t)$ from (1) with $a = 2/(v+1) \in (0,1)$ and $b = 2/(v-1) \in (0,\infty)$ for $v > 1$.

Proof. Here, we fix the coordinate system on the product (moved by the conveyor), for easing the calculation of the worker's walk time. Again, this is depicted in Figure 4. We begin with case (a). The box movement function then is $f(t) = -t+\pi_j$. The worker starts walking at time $\hat{t}$. The worker's forward movement is described by $g(t) = (t-\hat{t})v$, for $t \geq \hat{t}$. The worker visits the box at time t', thus $g(t') = f(t') \iff (t'-\hat{t})v = -t'+\pi_j \iff t' = (\pi_j+\hat{t}v)/(1+v)$. After that, the worker returns to the product by walking the same path backward. Therefore, the walk time is $2(t'-\hat{t}) = (\pi_j-\hat{t})\cdot 2/(1+v)$. Case (b) is calculated similarly. The worker's backward movement function is $g(t) = -(t-\hat{t})v$, for $t \geq \hat{t}$. Then, $g(t') = f(t') \iff t' = (\pi_j-\hat{t}v)/(1-v)$. Again doubling the distance for including the return, the walk time is $2(t'-\hat{t}) = (\pi_j-\hat{t})\cdot 2/(1-v)$. Combining both cases yields the stated walk time. □

Table 1. Numeric a, b values for walking strategies (A), (B), (C) and several v.

	(A) and (B)		(C)	
v	a	b	a	b
2	0.667	2.000	0.556	1.250
4	0.400	0.667	0.360	0.562
8	0.222	0.286	0.210	0.266
16	0.118	0.133	0.114	0.129
32	0.061	0.065	0.060	0.063

Walking strategy (C). When walking forwards, the worker velocity adds to the conveyor velocity. Therefore, we let the worker walk atop the moving conveyor floor plates in the forward direction (strategy (B)). Backwards instead, we rather let the worker walk beside the conveyor on the stationary floor, such that the opposing conveyor velocity does not reduce his or her velocity (strategy (A)). In summary, the forward velocity is $1 + v$, and the backward velocity is v. Then, walk time at t is $\varpi_j(t)$ from (1) with $a = (2v + 1)/(1 + v)^2$ and $b = (2v + 1)/v^2$ for $v > 1$.

Proof. Shown by combining the proofs of strategy (A) and (B). □

These results show that both walking strategy (A) and (B) yield the same walk time. Therefore, neither walking beside nor walking atop the conveyor is superior. However, their combination (C) allows to reduce the walk time significantly: for worker velocity $v = 13.6$ as in Klampfl *et al.* (2006), factor a reduces by 3.5% and b by 4.1%. Hence, it is advantageous to install moving conveyor floor plates and stationary edges at assembly lines, as it enables workers to apply walking strategy (C).

All three described walking strategies are covered by the same piecewise linear function in Equation (1). Moreover, as $a < b$ in each strategy, the walk time is shorter if the workpiece moves toward the box (case (a)). Exemplary a, b values for different worker velocities v are shown in Table 1.

3. Polynomial Cases

In this section, we introduce polynomial cases of $\mathcal{P}$. They are later used for analyzing the complexity of $\mathcal{P}$ and to derive lower bounds.

Given an instance, let us sort the boxes by $w_j(1 - a)^j$, for each $j \in J$. If this yields a placement where each box is at or behind their job's start time, this placement is optimum. Hence, the instance is polynomially solvable. An example is depicted in Figure 5.

Lemma 2. *Given an instance of $\mathcal{P}$ with n jobs. Also given a start time t, and a value $F \geq t$. We require that the box of job 1 is placed at $\pi_1 = F$. The box sequence is given by S. We further require that by S, each box gets placed at or behind its job start: $\pi_j \geq t_j$*

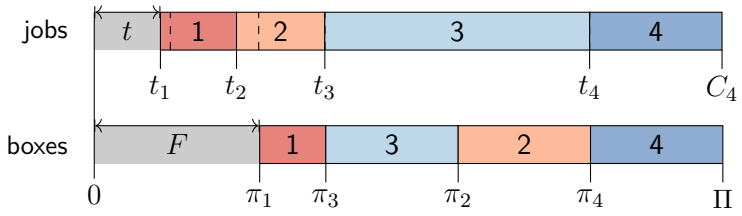

Figure 5. Given is an instance with $n = 4$ jobs. The first job starts at $t_1 = t$. Here, the boxes may only be placed at and behind F. The depicted placement sorts the boxes nondecreasingly by $w_j(1-a)^j$. Furthermore, each box $j \in J$ is placed at or behind its start time ($\pi_j \geq t_j$). Then, by Lemma 2, this placement results in a minimum last completion time, here C_4.

for all $j \in J$. If S is sorted nondecreasingly by $w_j(1-a)^j$, the last completion time C_n is minimum.

Proof. First, we derive a closed formula for the completion time C_n of the last job n by induction as follows. For ease of description, we express time duration t by a virtual job 0, which starts at 0 and completes at t, with processing time p_0. For this, we use $w_0 = F, \pi_0 = 0, l_0 = p_0$. Thus, the job set extends to $J' = \{0\} \cup J = \{0, 1, \ldots, n\}$.

Given some box sequence S with $S(0) = 0$ (the virtual job being the first). For the given S, the box positions then are $\pi_j = \sum_{k=0}^{S(j)-1} w_{S^{-1}(k)}$ for each $j \in J'$.

We like to calculate $C_n = \sum_{j=0}^{n} p_j$. For this, we need to know the value of p_j for all $j \in J'$. We begin with the definition $p_j = l_j + \max\{a\,(\pi_j - t_j),\, b\,(t_j - \pi_j)\}$. Knowing that $\pi_j \geq t_j$ for all $j \in J'$, we simplify it to $p_j = l_j + a\,(\pi_j - t_j)$. Then,

$$p_j = l_j + a\,(\pi_j - t_j) \iff p_j = l_j + a\left(\sum_{k=0}^{S(j)-1} w_{S^{-1}(k)} - \sum_{k=0}^{j-1} p_k\right).$$

For $j \in J$, we define $\Delta(j) = \sum_{k=0}^{S(j)-1} w_{S^{-1}(k)} - \sum_{k=0}^{S(j-1)-1} w_{S^{-1}(k)}$. Then,

$$\begin{aligned} p_j - p_{j-1} &= l_j - l_{j-1} \\ &\quad - a\sum_{k=0}^{j-1} p_k + a\sum_{k=0}^{j-2} p_k \\ &\quad + a\sum_{k=0}^{S(j)-1} w_{S^{-1}(k)} - a\sum_{k=0}^{S(j-1)-1} w_{S^{-1}(k)} \\ \iff p_j - p_{j-1} &= l_j - l_{j-1} - a p_{j-1} + a\,\Delta(j) \\ \iff p_j - (1-a)p_{j-1} &= l_j - l_{j-1} + a\,\Delta(j). \end{aligned} \tag{2}$$

This is a recurrence relation for p_j, starting with p_0. We reformulate this as a closed-form expression. For $j \in J$, we define $\Phi_j = p_j/(1-a)^j$. Then, from (2),

$$\begin{aligned} & \frac{p_j}{(1-a)^j} - \frac{(1-a)p_{j-1}}{(1-a)^j} = \frac{l_j - l_{j-1} + a\,\Delta(j)}{(1-a)^j} \\ \iff & \frac{p_j}{(1-a)^j} - \frac{p_{j-1}}{(1-a)^{j-1}} = \frac{l_j - l_{j-1} + a\,\Delta(j)}{(1-a)^j} \\ \iff & \Phi_j - \Phi_{j-1} = \frac{l_j - l_{j-1} + a\,\Delta(j)}{(1-a)^j}. \end{aligned}$$

The base case is $\Phi_0 = p_0 = t$, therefore

$$\begin{aligned} & \Phi_j - \Phi_0 = \sum_{k=1}^{j} \Phi_k - \Phi_{k-1} \\ \iff & \Phi_j - \Phi_0 = \sum_{k=1}^{j} \frac{l_k - l_{k-1} + a\,\Delta(k)}{(1-a)^k} \\ \iff & \frac{p_j}{(1-a)^j} = p_0 + \sum_{k=1}^{j} \frac{l_k - l_{k-1} + a\,\Delta(k)}{(1-a)^k} \\ \iff & p_j = t\,(1-a)^j + \sum_{k=1}^{j} (l_k - l_{k-1} + a\,\Delta(k))\,(1-a)^{j-k} \end{aligned}$$

We use this closed form expression for p_j to calculate C_n:

$$\begin{aligned} C_n &= \sum_{j=0}^{n} p_j \\ &= \sum_{j=0}^{n} \left(t\,(1-a)^j + \sum_{k=1}^{j} (l_k - l_{k-1} + a\,\Delta(k))\,(1-a)^{j-k} \right) \\ &= \sum_{j=0}^{n} \left(t\,(1-a)^j + \sum_{k=1}^{j} (l_k - l_{k-1})\,(1-a)^{j-k} \right) + \sum_{j=0}^{n} \sum_{k=1}^{j} a\,\Delta(k)\,(1-a)^{j-k}. \end{aligned} \tag{3}$$

Changing the box sequence S only influences the last term in line (3), which we denote by μ. Using equation $\sum_{j=k}^{n}(1-a)^{j-k} = (1-(1-a)^{n-k+1})/a$, $S(0) = 0$, $\left(1-(1-a)^1\right) = a$, and $\left(1-(1-a)^{n-j}\right) - \left(1-(1-a)^{n-j-1}\right) = \left((1-a)^{-1}-1\right)(1-a)^{n-j} = a\,(1-a)^{n-j-1}$, we reformulate μ:

$$\begin{aligned} \mu &= \sum_{j=0}^{n} \sum_{k=1}^{j} a\,\Delta(k)\,(1-a)^{j-k} \\ &= \sum_{k=1}^{n} \Delta(k)\,a \sum_{j=k}^{n} (1-a)^{j-k} \\ &= \sum_{k=1}^{n} \Delta(k)\,\left(1-(1-a)^{n+1-k}\right) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{j=1}^{n} \left(1-(1-a)^{n+1-j}\right) \left(\sum_{k=0}^{S(j)-1} w_{S^{-1}(k)} - \sum_{k=0}^{S(j-1)-1} w_{S^{-1}(k)} \right) \\
&= -\left(1-(1-a)^{n+1-1}\right) \sum_{k=0}^{S(1-1)-1} w_{S^{-1}(k)} \\
&\quad + \sum_{j=2}^{n} \Big(\left(1-(1-a)^{n+1-(j-1)}\right) - \left(1-(1-a)^{n+1-j}\right) \Big) \sum_{k=0}^{S(j-1)-1} w_{S^{-1}(k)} \\
&\quad + \left(1-(1-a)^{n+1-n}\right) \sum_{k=0}^{S(n)-1} w_{S^{-1}(k)} \\
&= a \sum_{k=0}^{S(n)-1} w_{S^{-1}(k)} + \sum_{j=1}^{n-1} \Big(\left(1-(1-a)^{n+1-j}\right) - \left(1-(1-a)^{n-j}\right) \Big) \sum_{k=0}^{S(j)-1} w_{S^{-1}(k)} \\
&= a \sum_{k=0}^{S(n)-1} w_{S^{-1}(k)} + a \sum_{j=1}^{n-1} (1-a)^{n-j} \sum_{k=0}^{S(j)-1} w_{S^{-1}(k)} \\
&= a \sum_{j=1}^{n} (1-a)^{n-j} \sum_{k=0}^{S(j)-1} w_{S^{-1}(k)} \\
&= a \sum_{j=0}^{n} w_j \sum_{k=S(j)+1}^{n} (1-a)^{n-S^{-1}(k)}.
\end{aligned}
$$

Term μ is minimized if S is sorted nondecreasingly by $w_j(1-a)^j$. We show this by contradiction, using an adjacent job interchange argument. Let j, j' be two jobs in a schedule $\tilde{S}$ with $\tilde{S}(j)+1 = \tilde{S}(j')$ and $w_j(1-a)^j > w_{j'}(1-a)^{j'}$. Suppose, for a contradiction, that term $\tilde{\mu}$ in $\tilde{S}$ is minimal. Construct a second schedule $\dot{S}$, identical to $\tilde{S}$ with the exception that $\dot{S}(j) = \tilde{S}(j')$ and $\dot{S}(j') = \tilde{S}(j)$, hence j and j' are swapped. Then, the corresponding $\tilde{\mu}$ and $\dot{\mu}$ are almost identical, except that $\tilde{\mu}$ contains summand $\tilde{x} = aw_j(1-a)^{n-j'}$, and $\dot{\mu}$ contains $\dot{x} = aw_{j'}(1-a)^{n-j}$. That is, $\tilde{\mu} - \tilde{x} = \dot{\mu} - \dot{x}$, and $w_j(1-a)^j > w_{j'}(1-a)^{j'} \iff w_j(1-a)^{-j'} > w_{j'}(1-a)^{-j} \iff aw_j(1-a)^{n-j'} > aw_{j'}(1-a)^{n-j} \iff \tilde{x} > \dot{x} \iff \tilde{\mu} > \dot{\mu}$. It follows that $\tilde{\mu}$ is not minimum. This completes the contradiction: any schedule that is not sorted nondecreasingly by $w_j(1-a)^j$ does not possess a minimum term μ. □

A symmetric polynomial case follows if sorting the boxes nonincreasingly by $w_j(1+b)^j$ results in start times that all are at or behind their box.

Lemma 3. *Given an instance of $\mathcal{P}$ with n jobs. Also given a start time t, and a value $F \leq t$. We require that the box of job 1 is placed at $\pi_1 = F$. The box sequence is given by S. We further require that by S, each box gets placed at or before its job start: $\pi_j \leq t_j$ for all $j \in J$. If S is sorted nonincreasingly by $w_j(1+b)^j$, the last completion time C_n is minimum.*

Proof. The proof is analogous to the proof of Lemma 2. Here instead, p_j can be simplified to $p_j = l_j + b\,(t_j - \pi_j)$. The remaining steps are similar. Therefore, we omit a full proof here. □

4. Computational Complexity

For analyzing the computational complexity of $\mathcal{P}$, we need to specify a decision version of $\mathcal{P}$. As usual for minimization problems, this is done by setting a threshold τ for the objective ϕ. If, for a given instance, there exists a solution with objective $\phi \leq \tau$, the instance is called a Yes-instance. Else, it is called a No-instance. Then, it is possible to polynomially reduce from, e.g., the NP-hard Partition Problem (Garey and Johnson, 1979, p. 47) to the decision version of $\mathcal{P}$. However, a stronger result is a reduction from the strongly NP-hard Three Partition Problem (Garey and Johnson, 1979, p. 96), which follows.

Definition 4 (Three Partition Problem (3P) (Garey and Johnson, 1979))**.** Given a bound $B \in \mathbb{N}$ and $3z$ elements in multiset $X = \{x_1, \ldots, x_{3z}\} \subset \mathbb{N}$ with $B/4 < x_j < B/2$, $j = 1, \ldots, 3z$, and $\sum_{x \in X} x = zB$. The question is: does there exists a partition of X into disjoint multisets $A^{(i)}$, $i = 1, \ldots, z$ with $\sum_{x \in A^{(i)}} x = B$?

In such a partition, each multiset consists of three elements. For any 3P instance, denoted by $3\mathrm{P}^{\mathcal{I}}$, we introduce a corresponding instance of $\mathcal{P}$'s decision version:

Definition 5 ($\mathcal{P}^{\mathcal{I}}$)**.** Given a 3P instance $3\mathrm{P}^{\mathcal{I}}$. Then, the corresponding $\mathcal{P}$ instance $\mathcal{P}^{\mathcal{I}}$ is as follows. Set $a = b = 1/(3z)$. Define $q = \lceil \log_{1+b} 2(z + b + zB)/b \rceil$ and $r = b^2/(1 - (1+b)^{-q})$. Note that r is polynomial in the input size as q is a rounded up logarithm of input sizes. Then, we construct an instance of $n = z + q + 3z$ jobs that are in a sequence of three parts:

(1) z filler jobs $j = 1, \ldots, z$ with box width $w_j = 1$ and assembly time $l_j = B + 1$,
(2) q enforcer jobs $j = z + 1, \ldots, z + q$ with equal box width and assembly time $w_j = l_j = r/(1+b)^{j-z}$,
(3) $3z$ partition jobs $j = z + q + 2, \ldots, n$ with box width $w_j = x_j$ and assembly time $l_j = 0$.

The total box width of enforcer jobs is $\sum_{j=z+1,\ldots,z+q} w_j = r \sum_{i=1,\ldots,q} (1+b)^{-i} = r \cdot (1 - (1+b)^{-q})/b = b$. Of all jobs, total box width is $\Pi = z + b + zB$. The sum of assembly times is $z(B+1) + a = \Pi$ as well. At last, we set $\tau = \lceil \mathrm{e} \cdot \Pi \rceil$ as threshold (e is Euler's number). As $z \geq 1$ and $B \geq 3$, there is $\Pi \geq 4$ and $\tau < 3 \cdot \Pi$.

An example of a Yes-instance is given and visualized in Figure 6.

Lemma 6. *Given a* 3P *instance* $3\mathrm{P}^{\mathcal{I}}$ *and the corresponding* $\mathcal{P}$ *instance* $\mathcal{P}^{\mathcal{I}}$*. If* $3\mathrm{P}^{\mathcal{I}}$ *is a Yes-instance, then there exists a solution of* $\mathcal{P}^{\mathcal{I}}$ *with* $\phi < \tau$*.*

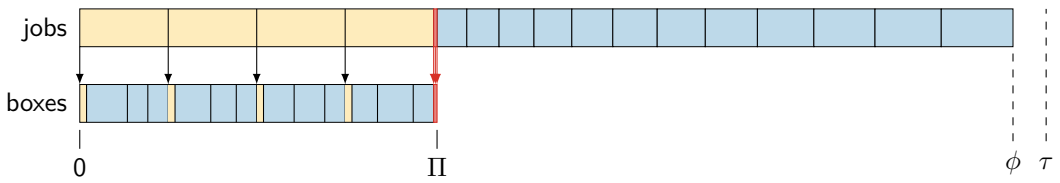

Figure 6. Given is an instance $\mathcal{P}^{\mathcal{I}}$ of $\mathcal{P}$ that corresponds to a 3P Yes-instance $3\mathrm{P}^{\mathcal{I}}$ with $B = 12$ and $z = 4$. Thus, $a = b = 1/12$, $q = 90$, $n = 106$, $\Pi = \sum_{j=1}^{n} w_j = \sum_{j=1}^{n} l_j = 52 + 1/12$, and $\tau = 142$. The job sequence begins with z filler jobs, then has q enforcer jobs, and finishes with $3z$ partition jobs. The partition jobs start behind Π. The box sequence alternates between a filler job box and three partition job boxes, then ends with all q enforcer job boxes. For the partition jobs, we just depicted their upper bound processing time, which emerges if assuming $\pi_j = 0$ for each partition job j. This visualizes that for any Yes-instance, there exists a solution with $\phi < \tau$ in $\mathcal{P}^{\mathcal{I}}$, see Lemma 6.

Proof. Given a Yes-instance $3\mathrm{P}^{\mathcal{I}}$ and a solution $A^{(i)}$, $i = 1, \ldots, z$. Let us construct a placement S for the corresponding instance $\mathcal{P}^{\mathcal{I}}$. For each filler job $j = 1, \ldots, z$, set $\pi_j = (j-1)B$. Then, for $i = 1, \ldots, q$, set $\pi_{z+i} = z + zB + r \sum_{k=1,\ldots,i} (1+b)^{-i}$. The partition jobs are placed between the filler jobs: For each $i = 1, \ldots, z$ and $x_j \in A^{(i)}$, we set $\pi_{z+q+i} = (i-1)(B+1) + \sum_{\{x_{j'} \in A^{(i)} | j' < j\}} x_{j'}$. There is no gap and no overlap in the constructed placement, therefore it is feasible. All filler and enforcer jobs $j = 1, \ldots, z+q$ start at $t_j = \pi_j$. Hence, the first partition job, $z+q+1$, starts at $C_{z+q} = \Pi$. By this, each partition job $j = z+q+1, \ldots, n$ is late, as it needs to be placed before Π. Still, π_j is positive. As a result, $C_j = C_{j-1} + b\,(C_{j-1} - \pi_j) > C_{j-1} + bC_{j-1}$. Solving this recurrence for all $3z$ partition jobs yields $C_n < (1+b)^{3z}\, C_r$ where $C_r = \Pi$. Therefore, the objective $\phi = C_n$ of this placement is bounded from above by $\phi < \Pi \cdot (1+b)^{3z} < \Pi \cdot \mathrm{e} < \tau$ with the known inequality $(1 + 1/x)^x < \mathrm{e}$. □

Lemma 7. *Given a* 3P *instance* $3\mathrm{P}^{\mathcal{I}}$ *and the corresponding* $\mathcal{P}$ *instance* $\mathcal{P}^{\mathcal{I}}$. *If* $3\mathrm{P}^{\mathcal{I}}$ *is a No-instance, then all solutions of* $\mathcal{P}^{\mathcal{I}}$ *have* $\phi > \tau$.

Proof. Given a No-instance $3\mathrm{P}^{\mathcal{I}}$ and the corresponding instance $\mathcal{P}^{\mathcal{I}}$. Consider a placement S with minimum objective ϕ^*. As $3\mathrm{P}^{\mathcal{I}}$ is a No-instance, there exists no partition of the elements into triple-sets $A^{(i)}$, $i = 1, \ldots, z$, of equal size B. Hence, in placement S, there is least one filler job $j \leq z$ with $|t_j - \pi_j| \geq 1$, thus $p_j \geq l_j + a$. Therefore, $C_z \geq \Pi - a + a = \Pi$. Now, all enforcer jobs are late. By Lemma 3, sorting the boxes by $w_{j-z}\beta(j-z) = r$ minimizes C_{z+q}. As the sort criterion has equal value r for each of the enforcer jobs, any order yields the same C_{z+q} value. Then, the recurrence for $j = z+1, \ldots, z+q$ is

$$\begin{aligned} C_j &= C_{j-1} + l_j + b\,(C_{j-1} - (\Pi - \sum_{k=z+1,\ldots,j} w_k)) \\ &= (1+b)\, C_{j-1} - b\Pi + \frac{r}{(1+b)^{j-z}} + \sum_{k=1,\ldots,j-z} \frac{br}{(1+b)^k} \end{aligned}$$

$$
\begin{aligned}
&= (1+b)\,C_{j-1} - b\Pi + r\left(\frac{1+b}{(1+b)^{j-z}} + \sum_{k=1,\dots,j-z-1}\frac{b}{(1+b)^k}\right)\\
&= (1+b)\,C_{j-1} - b\Pi + r\left((1+b)^{-(j-z-1)} + 1 - (1+b)^{-(j-z-1)}\right)\\
&= (1+b)\,C_{j-1} - b\Pi + r.
\end{aligned}
$$

Dividing both sides with $(1+b)^j$ yields

$$\frac{C_j}{(1+b)^j} = \frac{C_{j-1}}{(1+b)^{j-1}} + \frac{r-b\Pi}{(1+b)^j}.$$

Then, let $S_j = C_j/(1+b)^{j-z}$. Thus, $S_z \geq \Pi/(1+b)^0 = \Pi$ and, for $j = z+1,\dots,z+q$, there is $S_j = S_{j-1} + (r-b\Pi)/(1+b)^{j-z}$. Rewriting S_{z+q} as a sum, we have

$$S_{z+q} = S_z + \sum_{j=z+1,\dots,z+q}\frac{r-b\Pi}{(1+b)^{j-z}} = S_z + (r-b\Pi)\frac{1-(1+b)^{-q}}{b}.$$

Returning to C_{z+q}, we obtain the closed form

$$
\begin{aligned}
C_{z+q} &= S_{z+q}\cdot(1+b)^q\\
&= S_z\cdot(1+b)^q + (r-b\Pi)\cdot\left(1-(1+b)^{-q}\right)\cdot(1+b)^q/b\\
&= S_z\cdot(1+b)^q + b(1+b)^q - \Pi\cdot\left(1-(1+b)^{-q}\right)\cdot(1+b)^q\\
&= S_z\cdot(1+b)^q + b(1+b)^q - \Pi\cdot\left((1+b)^q - 1\right)\\
&\geq \Pi\cdot(1+b)^q + \Pi\cdot(1-(1+b)^q) + b(1+b)^q\\
&= \Pi + b(1+b)^q = \Pi + b(1+b)^{\log_{1+b} 2\Pi/b}\\
&= 3\cdot\Pi > \tau.
\end{aligned}
$$

Processing times for jobs $j > z+q$ are not negative, thus $C_n > \tau$. □

Theorem 8. *The decision version of $\mathcal{P}$ is strongly NP-complete.*

Proof. For any $3\mathrm{P}^{\mathcal{I}}$, there is a corresponding $\mathcal{P}^{\mathcal{I}}$. By Lemmata 6 and 7, instance $\mathcal{P}^{\mathcal{I}}$ is a Yes-instance if and only if $3\mathrm{P}^{\mathcal{I}}$ is a Yes-instance. Therefore, we constructed a reduction from 3P to $\mathcal{P}$. Testing for $\phi < \tau$ is done in polynomial time, thus $\mathcal{P}$ is in NP. As 3P is strongly NP-complete, and the reduction is pseudopolynomial, we conclude that $\mathcal{P}$ is strongly NP-complete. □

5. Lower Bound

To construct a branch and bound algorithm for $\mathcal{P}$, it is necessary to calculate a lower bound on the minimum attainable objective value of a possibly empty partial solution.

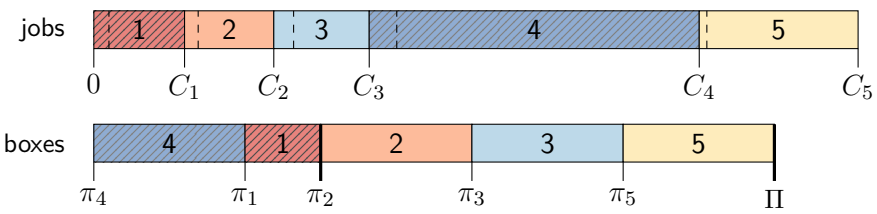

Figure 7. In the depicted partial schedule, jobs 1 and 4 are fixed: $J_\mathrm{F} = \{1, 4\}$. The lower bound places open boxes 2 and 3 behind the fixed boxes, while it places box 5 just before Π.

We let such a *partial schedule* be expressed by a possibly empty box sequence S. The set of jobs placed in S is called the *fixed* job set $J_\mathrm{F} \subseteq J$. Their boxes are placed in the order of S beginning from 0. Thus, they occupy a contiguous space that ends at $F = \sum_{j \in J_\mathrm{F}} w_j$. The partial schedule is extended to a full schedule by appending the remaining *open* jobs $J_\mathrm{O} = J \setminus J_\mathrm{F}$ to S in some order. As this order is undetermined for a partial schedule, we construct a lower bound on the objective value attained by S and any order of appending the remaining open boxes. Note that in any of these solutions, the box of an open job $j \in J_\mathrm{O}$ is placed in the interval $[F, \Pi - w_j]$.

We begin with the construction of two preliminary lower bounds derived from Lemma 2 and 3, respectively. In the first, we select a job subset $J' \subseteq J_\mathrm{O}$ and place their boxes in nondecreasing order of $w_j(1-a)^j$ at and behind F. Then, we set for each remaining job $j \in J_\mathrm{O} \setminus J'$ a box position $\pi_j = \max\{F, t_j\}$, which is as near to t_j as it may occur in a solution. Finally, we compute the start and completion times of all jobs, which results in some objective value. If and only if $\pi_j \geq t_j$ for all $j \in J'$, by Lemma 2, this objective value is a lower bound on the given partial schedule.

The second preliminary lower bound is constructed symmetrically, using Lemma 3. Here, we place the boxes of J' in nonincreasing order of $w_j\beta(j)$ directly before Π. For the remaining jobs $j \in J_\mathrm{O} \setminus J'$, we set the box positions to $\pi_j = \min\{t_j, \Pi - w_j\} \leq t_j$. Again, we compute the start and completion times of all jobs, resulting in some objective value. If and only if $\pi_j \leq t_j$ for all $j \in J''$, by Lemma 3, this objective is a lower bound on the given partial schedule.

Sorting the boxes is the runtime bottleneck of both preliminary lower bounds. However, these orders only rely on constant sorting criteria. Therefore, we calculate both orders in advance for all jobs J. Then, the bounding step only needs to select the relevant boxes from the sorted lists, merely taking $O(n)$ time.

Let us then integrate the two preliminary lower bounds into one. This is summarized in Algorithm 1. Here, the jobs $J = \{1, \ldots, n\}$ are iteratively visited, from first to last. After each iteration, i.e., after having visited jobs $1, \ldots, j$, the value t is a lower bound on their completion time. Also, t is a lower bound on the start time of the succeeding job $j + 1$. If a job j is in J_F, its box position π_j is known. Then, we increase t by the according processing time $p_j(t)$. If a job j is in J_O, its box is preliminarily placed to the nearest possible place, $\pi_j = \min\{\Pi - w_j, \max\{F, t\}\}$. However, if there are jobs after j that are also in J_O, we group them into set J', and try to place their boxes according

Algorithm 1. Combinatorial lower bound for $\mathcal{P}$

function LowerBound($J = \{1, \ldots, n\}$, $J_\mathrm{F} \subseteq J$, π_j for all $j \in J_\mathrm{F}$)
  $t \leftarrow 0$, $j \leftarrow 0$, $F := \sum_{j \in J_\mathrm{F}} w_j$ ▷ t is current start time, j is current job
  while $j < n$ **do**
    $j \leftarrow j + 1$ ▷ iterate job sequence
    if $j \in J_\mathrm{F}$ **then** ▷ j is a fixed job
      $t \leftarrow C_j(t)$
    else ▷ j is an open job
      for $j_\mathrm{max} \leftarrow \max\{j' \mid (\{j, j+1, \ldots, j'\} \cap J_\mathrm{F}) = \{\}\}$ **to** j **step** -1 **do**
        ▷ find large J' set
        if $j_\mathrm{max} = j$ **then** ▷ fallback case if only $J' = \{j\}$ is possible
          $\pi_j \leftarrow$ Nearest-GCD-Multiple(t) ▷ round box position
          $\pi_j \leftarrow \min\{\Pi - w_j, \max\{F, \pi_j\}\}$ ▷ constrain box position to limits
          $t \leftarrow C_j(t)$
        else
          $J' = \{j, \ldots, j_\mathrm{max}\}$
          if $t \leq F$ **then** ▷ place J' according to polynomial cases
            place J' nonincreasingly by $w_j(1-a)^j$ at F and behind
          else
            place J' nondecreasingly by $w_j(1+b)^j$ before Π
          $t' \leftarrow t$ ▷ temporarily calculate next start time
          for $j' \leftarrow j$ **to** j_max **do** ▷ test if jobs in J' fulfill polynomial conditions
            if $(t \leq F \wedge t' \leq \pi_{j'}) \vee (t \geq F \wedge t' \geq \pi_{j'})$ **then**
              $t' \leftarrow C_{j'}(t')$
            else
              $t' \leftarrow \infty$
          if $t' \neq \infty$ **then**
            $t \leftarrow t'$, $j \leftarrow j_\mathrm{max}$ ▷ conditions are fulfilled, now continue after j_max
            exit for
    $C_j^{(LB)} \leftarrow t$
  return t

to the two preliminary lower bounds described above. The grouping step heuristically maximizes the J' set of open jobs that meet the conditions. For this, it tests subsets of the smallest k elements in J_O. A linear search for maximizing k delivers a J' set in reasonably small, quadratic time. Then, the total runtime for checking a partial schedule is $O(n \cdot |J_\mathrm{O}|)$. The resulting lower bound is illustrated by an example in Figure 7.

The lower bound is further improved by an additional step in Algorithm 1, which is dedicated to open jobs that are not covered by either of the above rules. In the description above, we place such a job j at its lower bound start time $\pi_j = t_j$. However, box widths often follow a certain scheme in practice, e.g., divisions of ISO1- or EUR-pallets. Therefore, we can improve this placement by rounding it to a multiple of the greatest common divisor of all box widths. As rounding step needs to ensure a lower bound on C_j, hence, we round to the nearest multiple with the smallest possible walk distance. For example, with a greatest common divisor of 1, the nearest multiples are $\lceil t \rceil$ and $\lfloor t \rfloor$, hence the job is either early or late. If $(\lceil t \rceil - t)\, a < (t - \lfloor t \rfloor)\, b$, the rounded up value leads to a smaller walk distance, hence we set $\pi_j = \lceil t \rceil$. Else, we set $\pi_j = \lfloor t \rfloor$.

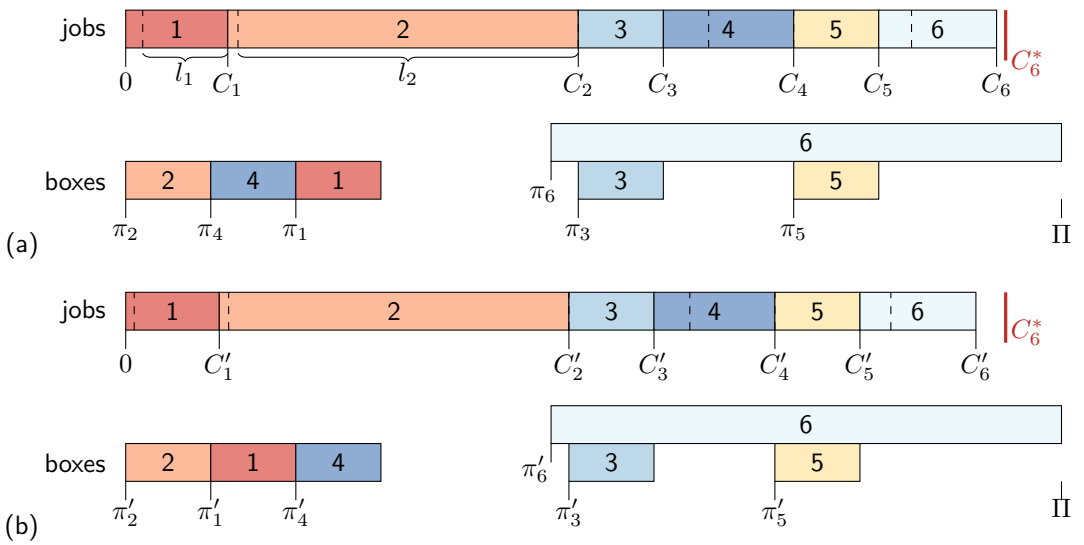

Figure 8. The depicted instance has $a = b = 0.1$ and $n = 6$ jobs with assembly times $l_1 = 1, l_2 = 4, l_3 = \cdots = l_6 = 1$, box widths $w_1 = \cdots = w_5 = 1, w_6 = 6$. Displayed is (a) partial schedule $S = \langle 2, 4, 1\rangle$ and (b) partial schedule $S' = \langle 2, 1, 4\rangle$ that swaps boxes 4 and 1. The open jobs $3, 5, 6$ are arranged by Algorithm 1, but neither S nor S' is pruned by bounding because the optimum value C_6^* is larger than the depicted lower bound placement. Nonetheless, the dominance rule shows that S' dominates S for all possible placements of the open jobs $3, 5, 6$. Therefore, it allows us to prune S.

6. Dominance Rule

6.1. Exact Dominance Rule

Dominance conditions are a common strategy for speeding up search algorithms like branch and bound by allowing for a comparison of partial solutions. We say that a partial schedule S is *dominated* by partial schedule S' if the objective for S' is smaller than the objective for S for any possible placement of the open jobs. The corresponding search branch for S can then be eliminated, thereby speeding up the search. In the following, we introduce a dominance rule that checks if a partial schedule is dominated by a partial schedule that swaps the boxes of the last two fixed jobs, or swaps the last job's box with any other fixed box of the same width. Note that any such swap operation allows to keep all other fixed boxes at the same place, which we require for proving correctness of the dominance rule. Moreover, we only consider swapping the last fixed box with some other box as, e.g., if used in a Branch and Bound search, any preceding box is already tested earlier in the search tree. An example with a fulfilled dominance condition on a swap of the last two fixed boxes is shown in Figure 8.

To compare partial schedules S and S', we need to take all possible placements for each open job into account. Say that S and S' have the same set of fixed jobs. Then, we only need to compare S and S' at equal placements of the open jobs. For this, we need to relate each job's completion times in both schedules. If the job is started at time t in S, it is started at time $t - \delta$ in S' for a given pair $\langle t, \delta\rangle_i$, where δ denotes

the difference $t - t'$ of start times. In any such pair, the two values are a result of an equal placement of all preceding open jobs. This gives a recurrence relation for $\langle t, \delta \rangle_i$. The corresponding pair of the next job $i+1$ if placing the boxes at π_i in S and at π'_i in S' is $\langle t`, \delta` \rangle_{i+1} = \langle C_i(t),\, C_i(t) - C'_i(t-\delta) \rangle_{i+1}$. If a job i is fixed, π_i and π'_i are determined. If i is an open job, the placement value is undetermined, but situated at equal positions $\pi_i = \pi'_i \in [F, \Pi - w_i]$ in the remaining space. Hence, we commence with $\langle t, 0 \rangle_1$ and continue inductively until reaching the virtual job $n+1$ with $\langle t, \delta \rangle_n$. If this δ is positive in all pairs $\langle t, \delta \rangle_{n+1}$, then S' dominates S. However, generating this pair for all possible placements requires an exhaustive search. In the following, we constrain the search and merely generate relevant pairs.

We define $\mathbb{T}_j = [t_j^{\min}, t_j^{\max}]$ as the range of possible start times of a job j. Start and end of the interval describe a lower and an upper bound on start time of job j for all possible placements of the open jobs J_O. A lower bound is obtained for some jobs during execution of Algorithm 1: Let $q \le j$ be the last job for which Algorithm 1 set a $C_q^{(LB)}$ value. Then, we can use $t_j^{\min} = C_q^{(LB)} + \sum_{r=q+1,\dots,j-1} l_r$. An upper bound value is obtained by subtracting assembly times of succeeding jobs from a global upper bound UB on C_n, i.e., $t_j^{\max} = UB - \sum_{k=j,\dots,n} l_k$. Note these bounds relate $t_j^{\min} + l_j = t_{j+1}^{\min}$ and $t_j^{\max} + l_j = t_{j+1}^{\max}$ for all $j = 1, \dots, n-1$.

We are given a partial schedule S and two fixed jobs j, k, $j < k$, for which either $\pi_k = \pi_j + w_j$, $\pi_j = \pi_k + w_k$, or $w_j = w_k$ hold. Consider partial schedule S' which places the boxes equally except for swapping the box placement of j and k. Note that a swap of j and k affects no other box position. Therefore, completion times of jobs $q < j$ remain the same. Moreover, job j starts at the same time in S and in S'. Also, $\mathbb{T}'_j = \mathbb{T}_j$ and $\langle t, 0 \rangle_j$ for all $t \in \mathbb{T}_j$.

Property 9. *Given pair $\langle t, 0 \rangle_j$ for start time $t \in \mathbb{T}_j$ of job j and $\pi_j + \omega = \pi'_j$. Let $\langle t`, \delta` \rangle_{j+1}$ be a corresponding pair for job $j+1$. If $\omega \ge 0$, there is $-a\omega \le \delta` \le b\omega$, else $b\omega \le \delta` \le -a\omega$. Moreover, $\delta`$ is extremal if it equals $\delta^{(a)}$ or $\delta^{(b)}$ in $\langle t_{j+1}^{\min}, \delta^{(a)} \rangle_{j+1}$, $\langle t_{j+1}^{\max}, \delta^{(b)} \rangle_{j+1}$.*

Proof. Given start time t, there is

$$
\begin{aligned}
\delta` &= C_j(t) - C'_j(t) \\
&= \max\{a(\pi_j - t),\, b(t - \pi_j)\} - \max\{a(\pi'_j - t),\, b(t - \pi'_j)\} \\
&= \max\{a(\pi_j - t),\, b(t - \pi_j)\} + \min\{-a(\pi'_j - t),\, b(\pi'_j - t)\} \\
&= \max\{a(\pi_j - t) + \min\{-a(\pi_j + \omega - t),\, b(\pi_j + \omega - t)\}, \\
&\qquad\qquad b(t - \pi_j) + \min\{-a(\pi_j + \omega - t),\, b(\pi_j + \omega - t)\}\} \\
&= \max\{\min\{-a\omega,\, (a+b)(\pi_j - t) + b\omega\}, \\
&\qquad\qquad \min\{-(a+b)(\pi_j - t) - a\omega,\, b\omega\}\}.
\end{aligned}
$$

If $(a+b)(\pi_j - t) + b\omega \leq -a\omega$, there is $(a+b)(\pi_j - t + \omega) \leq 0$. However, if $\omega \geq 0$, there is $(a+b)(\pi_j - t + \omega) \leq -(a+b)(\pi_j - t)$ and $(a+b)(\pi_j - t) + b\omega \leq -(a+b)(\pi_j - t) - a\omega$. Thus, for $\omega \geq 0$, there is $\delta\text{`} = \max\{-a\omega,\, \min\{(a+b)(t-\pi_j) - a\omega,\, b\omega\}\}$. Case $\omega \leq 0$ is analogous, there is $\delta\text{`} = \max\{\min\{-a\omega,\, (a+b)(\pi_j - t) + b\omega\},\, b\omega\}$. Moreover, $\delta\text{`}$ is a monotonic function of $t \in \mathbb{T}_j$ in both cases. Hence, it is extremal for extreme values of t. □

Depending on π_j and π_k, the value of ω is either positive or negative. We denote extremal value pairs by $\langle t^{(a)}, \delta^{\min}\rangle_{j+1}$ and $\langle t^{(b)}, \delta^{\max}\rangle_{j+1}$, obtained from $\langle t_j^{\min}, 0\rangle_j$ and $\langle t_j^{\max}, 0\rangle_j$. Note that $t^{(a)}$ is the larger value if ω is negative.

The box of job q, with $j \neq q \neq k$, is placed at the same position $\pi_q = \pi'_q$ in both S and S'. The difference of its completion time between S and S' is both influenced by box positions $\pi_q = \pi'_q$ and by the start times as defined by $\langle t, \delta\rangle_q$. Furthermore, if $q \in J_{\mathrm{O}}$, its box position is undetermined. Even then, we can, however, state bounds for the start time difference if we restrict our considerations to same signs of all difference values δ of job q, i.e., require that $\delta_q^{\min} \cdot \delta_q^{\max} \geq 0$.

Property 10. *Given a job $q \in J \setminus \{j,k\}$ with $\pi_q = \pi'_q$ and $\langle t, \delta\rangle_q$. For this, let $\langle t\text{`}, \delta\text{`}\rangle_{q+1}$ be the corresponding pair of job $q+1$. If $\delta\text{`} \geq 0$, then $(1-a)\delta \leq \hat{\delta}\text{`} \leq (1+b)\delta$. Else, $(1+b)\delta \leq \delta\text{`} \leq (1-a)\delta$. Moreover, $\delta\text{`}$ is extremal for this placement if $\langle t, \delta\rangle_q \in \{\langle t^{(a)}, \delta^{\min}\rangle_q,\, \langle t^{(b)}, \delta^{\max}\rangle_q\}$.*

Proof. Let $t' = t - \delta$. Then,

$$
\begin{aligned}
\delta\text{`} &= C_q(t) - C_q(t') \\
&= \delta + \max\{a(\pi_q - t),\, b(t - \pi_q)\} - \max\{a(\pi_q - t'),\, b(t' - \pi_q)\} \\
&= \delta + \max\{a(\pi_q - t),\, b(t - \pi_q)\} + \min\{-a(\pi_q - t'),\, -b(t' - \pi_q)\} \\
&= \delta + \max\{\, \min\{a(\pi_q - t) - a(\pi_q - t'),\, a(\pi_q - t) - b(t' - \pi_q)\}, \\
&\qquad\qquad \min\{b(t - \pi_q) - a(\pi_q - t'),\, b(t - \pi_q) - b(t' - \pi_q)\}\} \\
&= \delta + \max\{\, \min\{-a\delta,\, -a\delta + (a+b)(\pi_q - t')\}, \min\{b\delta + (a+b)(t' - \pi_q),\, b\delta\}\} \\
&= \delta + \max\{-a\delta + \min\{0,\, (a+b)(\pi_q - t')\}, b\delta + \min\{(a+b)(t' - \pi_q),\, 0\}\} \\
&= \delta + \max\{-a\delta + \min\{0,\, (a+b)(\pi_q - t + \delta)\}, b\delta + \min\{(a+b)(t - \delta - \pi_q),\, 0\}\}.
\end{aligned}
$$

As $\min\{0,\, x\} \leq 0$ for any number x, we derive the stated bounds for $\delta\text{`}$. Furthermore, $\delta\text{`}$ is a monotonic function of t and δ. Therefore, the extrema of $\delta\text{`}$ are obtained for extremal values of both t and δ. □

Finding extremal values for the difference is similar for job k.

Property 11. *For job k, we are given placements π_k, π'_k and $\langle t, \delta\rangle_k$. Let $\langle t\text{`}, \delta\text{`}\rangle_{k+1}$ be the corresponding $k+1$ pair. Then, $\delta\text{`}$ is extremal if $\langle t, \delta\rangle_k \in \{\langle t^{(a)}, \delta^{\min}\rangle_k,\, \langle t^{(b)}, \delta^{\max}\rangle_k\}$.*

Proof. Let $t' = t - \delta$ and $\gamma = \pi_k - \pi'_k$. Then,

$$\begin{aligned}\delta\text{`} &= C_k(t) - C'_k(t') \\ &= \delta + \max\{-a(t-\pi_k),\, b(t-\pi_k)\} - \max\{-a(t'-\pi'_k),\, b(t'-\pi'_k)\} \\ &= \delta + \max\{-a(t-\pi_k),\, b(t-\pi_k)\} - \max\{-a(t'+\gamma-\pi_k),\, b(t'+\gamma-\pi_k)\} \\ &= C_k(t) - C_k(t'+\gamma).\end{aligned}$$

Adding γ to t', we establish difference $\bar{\delta} = \delta - \gamma$. As the offset to δ is constant, Property 10 applies for π_k and $\langle t, \bar{\delta}\rangle_k$. □

In Properties 9, 10, and 11, the function for δ` is monotonic. Therefore, a larger range for t (which we have) only extends the extremal value range of δ.

If $q \in J_\mathrm{O}$, its box position π_q is undetermined. Thus, we cannot find precise extremal values for $\hat{\delta}_q$. Nonetheless, given difference δ, a lower bound on the succeeding δ` is determined by multiplying $\delta$ with $1 - a$ if $\delta$ is nonnegative, else with $1 + b$. The sign of $\delta$` equals the sign of δ. Therefore, we can inductively state for all q, where $j < q < k$, that the sign of δ` for job q equals the sign of the difference $\hat{\delta}$ after job j, and

$$\delta\text{`} \geq \hat{\delta} \cdot \begin{cases} (1-a)^{q-j}, & \hat{\delta} \geq 0, \\ (1+b)^{q-j}, & \text{else.} \end{cases} \tag{4}$$

Similarly, we can inductively state for all $q > k$ that the sign of δ` for job q equals the sign of $\check{\delta}$, the difference after job k. Hence, we barely need to compute $\hat{\delta}$ for $t_j^{\min}$ and $t_j^{\max}$, multiply it for a lower bound as in (4), and calculate $\check{\delta}$ for $t_k^{\min}$ and $t_k^{\max}$. If $\check{\delta} > 0$ for both start times, we now can say that $C_n > C'_n$ for all possible open job placements. In this case, partial schedule S' dominates S.

The runtime of testing the described dominance criterion on S, S' is $O(n)$ in a naïve implementation. We can precompute a list of cumulated assembly times, size n. Furthermore, Algorithm 1 can store, for each job $j \in J$, a pointer to the last job $q \leq j$ for which it calculated a lower bound on the completion time, i.e., $C_q^{(LB)} \geq 0$. The resulting steps are described in Algorithm 2. This reduces the runtime of the dominance criterion to constant time $O(1)$ for comparing S to S'.

6.2. Heuristic Dominance Rule

Integrating more than an adjacent box swap results in a change of more than two box positions. Then, upper and lower bounds of the difference between two partial schedules are not only harder to obtain, but of inferior quality. Nonetheless, it is still of interest to conduct a comparison of two partial schedules S, S' with the same set of fixed jobs but placing more than two boxes at different positions. Of course, appending the same sequence of open jobs to both S and S' allows comparing objective values. Testing

Algorithm 2. Dominance rule for $\mathcal{P}$

function CHECKDOMINANCE($J = \{1, \ldots, n\}$, UB, $J_\mathrm{F} \subseteq J$, $j, k \in J_\mathrm{F}$ for $j < k$, sequence S with box positions π_j, π_k (either adjacent or with $w_j = w_k$), and sequence S' with swapped π'_j, π'_k)
  $t_\mathrm{min} \leftarrow t'_\mathrm{min} \leftarrow \max\{C_q^{(LB)} + \sum_{r=q+1,\ldots,j-1} l_r \mid q \le j\}$ ▷ initialize earliest start time of j
  $t_\mathrm{max} \leftarrow t'_\mathrm{max} \leftarrow UB - \sum_{k=j,\ldots,n} l_k$ ▷ initialize latest start time of job j
  $t_\mathrm{min} \leftarrow C_j(t_\mathrm{min})$, $t'_\mathrm{min} \leftarrow C'_j(t_\mathrm{min})$ ▷ process job j
  $t_\mathrm{max} \leftarrow C_j(t_\mathrm{max})$, $t'_\mathrm{max} \leftarrow C'_j(t_\mathrm{max})$
  $\hat{\delta}_j^\mathrm{min} \leftarrow t_\mathrm{min} - t'_\mathrm{min}$ ▷ calculate difference
  $\hat{\delta}_j^\mathrm{max} \leftarrow t_\mathrm{max} - t'_\mathrm{max}$
  if $\hat{\delta}^\mathrm{min} \cdot \hat{\delta}^\mathrm{max} \ge 0$ **then** ▷ constrain to differences of same sign
    if $\hat{\delta}^\mathrm{min} \ge 0$ and $\hat{\delta}^\mathrm{max} \ge 0$ **then**
      $\hat{\delta}_{k-1}^{(LB)\ \mathrm{min}} \leftarrow \hat{\delta}_j^\mathrm{min} \cdot (1-a)^{j-k}$ ▷ calculate lower bound on difference
      $\hat{\delta}_{k-1}^{(LB)\ \mathrm{max}} \leftarrow \hat{\delta}_j^\mathrm{max} \cdot (1-a)^{j-k}$
    else
      $\hat{\delta}_{k-1}^{(LB)\ \mathrm{min}} \leftarrow \hat{\delta}_j^\mathrm{min} \cdot (1+b)^{j-k}$
      $\hat{\delta}_{k-1}^{(LB)\ \mathrm{max}} \leftarrow \hat{\delta}_j^\mathrm{max} \cdot (1+b)^{j-k}$
    $t_\mathrm{min} \leftarrow t_\mathrm{min} + \sum_{q=j+1,\ldots,k-1} l_q$ ▷ calculate earliest start of job k
    $t_\mathrm{max} \leftarrow t_\mathrm{max} + \sum_{q=j+1,\ldots,k-1} l_q$
    $t'_\mathrm{min} \leftarrow t_\mathrm{min} - \hat{\delta}_{k-1}^{(LB)\ \mathrm{min}}$ ▷ calculate latest start of job k
    $t'_\mathrm{max} \leftarrow t_\mathrm{max} - \hat{\delta}_{k-1}^{(LB)\ \mathrm{max}}$
    $t_\mathrm{min} \leftarrow C_k(t_\mathrm{min})$, $t'_\mathrm{min} \leftarrow C'_k(t_\mathrm{min})$ ▷ process job k
    $t_\mathrm{max} \leftarrow C_k(t_\mathrm{max})$, $t'_\mathrm{max} \leftarrow C'_k(t_\mathrm{max})$
    if $t_\mathrm{min} > t'_\mathrm{min} \wedge t_\mathrm{max} > t'_\mathrm{max}$ **then** ▷ check if swapped completes earlier
      return swapped-is-dominant
  return swapped-is-*non*-dominant

dominance with all possible sequences of the open jobs requires, however, an exponential search in the worst case. To avoid this, we pick out corner cases of placing the open jobs. Although this yields only a heuristic dominance rule, it is relatively effective if these corner cases are representative of many other open job placements. Then, if S' dominates S for all these corner cases, we can decide that S is probably not leading to an optimum solution.

In particular, we pick the following two corner cases of placing the open jobs J_O:

(a) set $\pi_j = F$ for all $j \in J_\mathrm{O}$,
(b) set $\pi_j = \Pi - w_j$ for all $j \in J_\mathrm{O}$.

Both corner case placements are in fact infeasible. Still, we can calculate each resulting objective C_max, and compare S to S' for each corner case. The rationale for picking these two corner cases is as follows. The jobs can be looked at in segments, namely X and Y: the jobs that start before F, and those that start at or after F. Let us consider segment X. Here, corner case (a) starts each job in X as early as possible, while (b) starts them as late as possible. Hence, the corner cases cover both extremes in the X segment. Secondly, we look at segment Y. In Y, all fixed jobs are necessarily late. Any further delay has a proportional effect on each of the fixed jobs, only caused by

the open job placement. Therefore, as long as each open job's placement is the same in both corner cases, the exact place is unimportant. Note that the division into the X and Y segments may differ between each corner case and between schedules. Still, the comparison is fair as the boxes are equally placed in both schedules, respectively for each corner case.

Thus, the heuristic compares each partial schedule only by each corner case's objective value. Moreover, it suffices to conduct only a single comparison, which ideally is to the best known partial schedule with the same set of fixed jobs. Therefore, each time we visit a non-dominated partial schedule, we store each of the two corner case's C_n value in memory. The set of fixed jobs can be encoded by a binary vector of length n that determines membership of each job in set J_F. Hence, we store at most 2^n values for each corner case in a table. If available memory is limited, one may reduce the number of stored values to a constant size with a memory cache. Concluding, calculating the corner case objectives takes $O(n)$ and comparing S to all previously checked partial schedules with the same set of fixed jobs takes $O(1)$ time.

7. Solution Algorithms

Approaches for computationally solving instances of $\mathcal{P}$ are manifold. In this chapter, we first introduce a mixed integer program (MIP) and heuristic methods. Then, the described lower bound (Section 5) and dominance rules (Section 6) are utilized in a branch and bound algorithm. Finally, we derive a heuristic version of the branch and bound algorithm.

7.1. Mixed Integer Program

The box placement problem is described as a sequencing problem, which lends for an application of established modeling strategies for single machine scheduling (Błażewicz *et al.*, 1991; Keha *et al.*, 2009; Pinedo, 2016). In a preliminary test between MIP approaches, we compared time indexing, linear ordering/sequencing variables, and disjunctive box overlapping constraints. The latter turned out as the quickest variant by far. Therefore, we describe it in the following.

For each job $j = 1, \ldots, n$, there is a completion time variable C_j and a box placement variable π_j. To determine a box sequence, we introduce binary variables x_{jk}, $1 \leq j < k \leq n$, each of which is zero if box k is placed before box j and one otherwise.

$$\text{minimize } C_n \tag{5a}$$

$$\text{subject to } C_0 = 0, \tag{5b}$$

$$C_j \geq C_{j-1} + l_j - a\,(C_{j-1} - \pi_j), \qquad 1 \leq j \leq n, \tag{5c}$$

$$C_j \geq C_{j-1} + l_j + b\,(C_{j-1} - \pi_j), \qquad 1 \leq j \leq n, \tag{5d}$$

$$\pi_j + w_j \leq \pi_k + \Pi \cdot (1 - x_{jk}), \qquad 1 \leq j < k \leq n, \tag{5e}$$
$$\pi_k + w_k \leq \pi_j + \Pi \cdot x_{jk}, \qquad 1 \leq j < k \leq n, \tag{5f}$$
$$0 \leq \pi_j \leq \Pi - w_j, \qquad 1 \leq j \leq n, \tag{5g}$$
$$x_{jk} \in \{0, 1\}, \qquad 1 \leq j < k \leq n. \tag{5h}$$

Constraint (5b) sets start time for the first job to zero. Constraints (5c) and (5d) calculate the completion time iteratively from the completion time of the preceding job, which is either larger or smaller than the box placement variable, thus requiring two inequations. Depending on x_{jk}, $1 \leq j < k \leq n$, either (5e) or (5f) ensure as disjunctive constraints that boxes j, k are not overlapping while each is placed in its interval (5g). The objective is to choose feasible box placements $\pi_1, \ldots, \pi_n$ that minimize C_n, the completion time of the last job.

7.2. Heuristics

An intuitive placement for the boxes is to order them identical to the job sequence. This strategy is often used as a standard guideline by production planners in practice. Let us call this the identity sequence (ID) heuristic.

However, the identity sequence heuristic is mostly far from optimum, as we observe from test results in Section 8.4. Therefore, it is sensible to improve this solution. We apply a steepest-descent hill-climbing search to improve this initial box sequence. Repeatedly, the best neighbor of a sequence is chosen as the next sequence, until arriving at a local optimum. The neighborhood consists of swaps between all pairs of two boxes. We call this the identity sequence with a hill climbing search (HC) heuristic.

The resulting solution can be improved even more with a second metaheuristic that tries to evade local minima. An example is the simulated annealing (SA) method (Kirkpatrick *et al.*, 1983). For changing the solution, one can use the same neighborhood as before: swapping arbitrary box pairs. In contrast to a descending hill-climbing search, this metaheuristic allows ascending to worse solutions to a certain degree. This enables leaving local minima for finding the global minimum.

7.3. Branch and Bound Algorithm

To solve the given problem exactly, we introduce a branch and bound (B&B) algorithm. We begin by saving the best sequence of the HC heuristic presented in Section 7.2 and set the upper bound value to its objective value. Then, we start a depth-first search. We begin at the root node with the empty partial sequence where all jobs J are open jobs, $J_\mathrm{O} = J$. For each job in J_O, in order of the job index, a children node is created by removing the job from J_O and appending the job to the current partial sequence. Then, we check if bounding and the dominance rule allows pruning the new partial sequence. Pruning is possible

(a) if the lower bound of Algorithm 1 is larger or equal to the best known upper bound, or
(b) if swapping the last placed box with the box right before, or with any other fixed box of the same width yields a partial schedule for which the dominance rule of Section 6.1 applies.

Else, this branch node is further explored recursively. When reaching a branch node with $J_O = \{\}$, all boxes are placed and the objective ϕ can be calculated. If ϕ is smaller than the current upper bound, we save the sequence and set the upper bound to ϕ. After exploring of all non-pruned branches, the last saved sequence is returned, it is an optimal solution.

7.4. Truncated Branch and Bound Heuristic

In a truncated branch and bound heuristic (TrB&B), we limit the size of the B&B tree. For this, we constrain the branching factor, which is the number of nodes that emerge from each node, by maximum branching factor $\mathrm{BF}_{\max} = \min\{|J_O|, \max\{\lceil\psi\rceil, \lfloor|J_O|/\sigma\rfloor\}\}$, for positive constants ψ, σ. Parameter ψ controls the maximum number of branches at the end of the search tree, and σ at the start and in the middle of the search tree, depending on the number of free boxes $|J_O|$. To rank and select the most promising branches, we evaluate all $|J_O|$ emerging nodes. For this, we use the identity sequence as in the ID heuristic in Section 7.2. Hence, we rank by job sequence and, therefore, select the $\mathrm{BF}_{\max}$ smallest job indices from set J_O. A further reduction of the tree size is achieved by use of the heuristic dominance criterion from Section 6.2: First, we test the heuristic dominance on swapping the last placed box with any other fixed box. Secondly, we test against the best-known solution with the same J_F set.

8. Numerical Results

In a numerical experiment, we assess optimizing the box placement quantitatively. Moreover, we like to analyze the performance of the algorithms from Section 7. Hence, we test them on a variety of generated instances and statistically compare their performance. As a primary criterion, we utilize median runtime and quartile deviation on instance groups of similar parameters, to find which of the algorithms quickly and robustly yield exact solutions. For the heuristics, we additionally compare solution quality by counting optimally solved instances and calculating mean error. We use these criteria to evaluate which heuristic provides the best tradeoff between runtime and solution quality.

8.1. Instance Generation

We generate test instances in several variants to evaluate performance in different settings. A problem instance is characterized by its size, the factors a and b, and each assembly time and box width.

The main influence on the instance size in practice is the cycle time. In the automotive industry, high-volume cars in the compact segment have rather small cycle times, which leaves room for only few operations; their number can be as low as five to ten. On the other hand, low-volume products like luxury vehicles or trucks have much longer cycle times. They allow for many more operations per cycle, often in the range from twenty up to almost thirty. Naturally, this is a more challenging setting, even more so as $\mathcal{P}$ is an NP-hard problem. Therefore, we focus our tests on these larger instance sizes, with number of jobs $n \in \{16, 20, 24, 28\}$. The worker velocity v is a multiple of the conveyor velocity. In practice, it is commonly in the range 8 to 16. To test the algorithms for extreme velocities, we let $v \in \{2, 4, 8, 16, 32\}$. In Section 2.3, we distinguish between three walking strategies. As strategies (A) and (B) are effectively equivalent, we obtain two variants for setting factors a, b for each v:

(S1) $a = 2/(v+1)$ and $b = 2/(v-1)$, as in (A) and (B),
(S2) $a = (2v+1)/(v+1)^2$ and $b = (2v+1)/v^2$, as in (C).

The resulting factors for both variants are listed in Table 1. The assembly time generation follows an established scheme (Jaehn and Sedding, 2016):

(L1) all equal assembly times $l_1, \ldots, l_n = 1$,
(L2) all distinct assembly times $\{l_1, \ldots, l_n\} = \{1, \ldots, n\}$, which randomly permutes integers $1, \ldots, n$,
(L3) assembly times drawn uniformly from $\{1, \ldots, 10\}$,
(L4) assembly times drawn from a geometric distribution with random variable $X = \lceil -\lambda \ln U \rceil$ for U uniformly distributed in $[0, 1]$ and $\lambda = 2$.

As well, we generate box widths in four variants:

(W1) all equal box widths $w_1, \ldots, w_n = 1$,
(W2) all distinct box widths $\{w_1, \ldots, w_n\} = \{1, \ldots, n\}$ in a random permutation,
(W3) box widths drawn uniformly from $\{2^0, \ldots, 2^3\} \cup \{3 \cdot 2^0, \ldots, 3 \cdot 2^2\}$, reflecting seven divisions of ISO1-pallets,
(W4) box widths drawn from rounded up gamma variates with shape 1.25 and unit scale, representing measured box width distributions at automotive assembly lines.

To fill the station (here, we let its width $s = 10 \cdot n$), all boxes of an instance are normalized for a total width of s by scaling and rounding each.

The processing times and the box widths are initially of different scale. This necessitates a harmonization. Ideally, the station length would equal the last completion time C_n. The identity sequence heuristic in Section 7.2 places the boxes according to the job

Table 2. MIP, B&B runtime and performance.

	MIP			B&B		
n	Md	QD	solved	Md	QD	solved
16	0.28	0.62	100%	0.00	0.01	100%
20	2.16	10.52	98%	0.06	0.13	100%
24	25.37	171.43	85%	0.90	3.68	98%
28	205.08	894.88	66%	11.03	73.08	89%
all	4.45	60.25	87%	0.10	1.57	97%

Md: median runtime in seconds
QD: quartile deviation in seconds
solved: percentage of instances solved in 30 minutes

sequence. For harmonizing assembly times and box widths, we use this placement and scale the assembly times linearly by a positive rational factor. The appropriate factor minimizes the absolute difference $|C_n - \Pi|$ and is determined by the algorithm of Brent (1971) for finding a zero of a univariate function.

In summary, the described scheme considers four numbers of jobs n, five times two variants for a and b, four assembly time variants, and four box width variants. For each parameter combination, it generates 10 instances. This yields a total of $4 \cdot (5 \cdot 2) \cdot 4 \cdot 4 \cdot 10 = 6\,400$ test instances.

8.2. Test Setup

The tested algorithms are implemented in C++. Our data structures use plain STL containers without additional dependencies. The TrB&B heuristic is parameterized with $\psi = 5$, $\sigma = 7$. For the simulated annealing algorithm, we rely on the reference implementation of Press *et al.* (1992, pp. 448–451) with default parameters. All code is compiled with GCC 7.2 on Ubuntu Linux, Kernel 4.4, and each instance is executed separately on an Intel Xeon E5-2680 CPU at 2.80 GHz. The MIP model is solved with Gurobi 7.5. We use its C++ interface to ensure the best performance and disabled multiprocessing capabilities for a fair comparison. Each instance and algorithm is terminated after a time limit of 30 minutes, hence at least delivering a lower bound on the runtime. This allows calculating median and quartiles while taking terminated instances into account.

8.3. Exact Algorithms

In Table 2, measured MIP and B&B runtimes are grouped by instance size n. Apparently, they grow exponentially, as it is expected to happen for a strongly NP-complete problem. However, high quartile deviations for both algorithms show that difficulty between instances varies by a high degree. Nonetheless, both algorithms agree on the difficulty of each instance, as there is a weak positive correlation of value 0.31 between MIP and

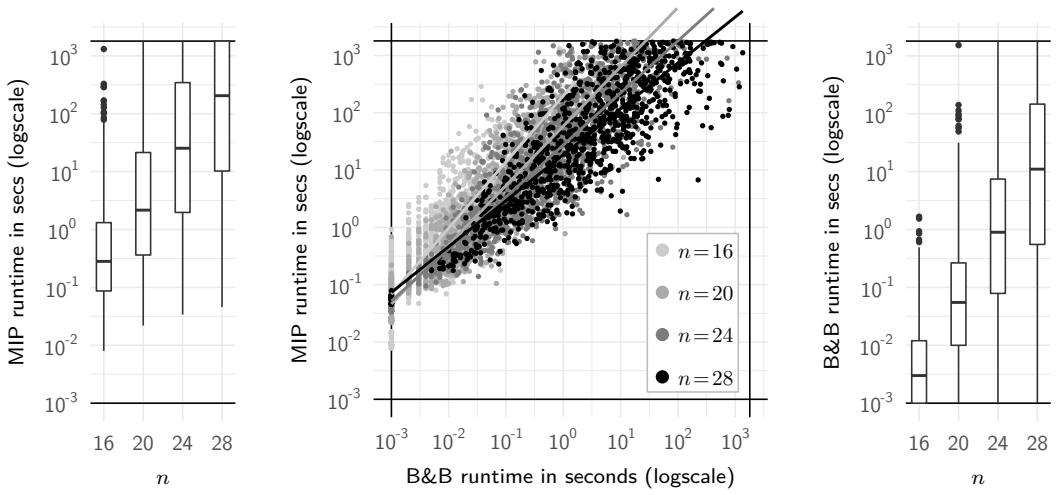

Figure 9. MIP and B&B runtime in seconds is shown on a logarithmic scale ranging from 1 millisecond (10^{-3}) to 30 minutes ($1.8 \cdot 10^{3}$) in box plots and a scatter plot with a linear regression line for each n.

B&B runtimes, see Figure 9. Every instance that MIP solves, is as well solved by B&B, which moreover is 50.0 times faster (median ratio) and solves nearly all instances.

Let us break down the B&B performance in greater detail for one size, $n = 24$. Median runtimes are shown for each (v, S) pair in Table 3, and each (L, W) pair in Table 4. First, the worker velocity apparently affects problem difficulty. It peaks at $v = 16$ and is easiest for the slowest velocity $v = 2$. We assume that this can be explained by the behavior of lower bounds. A slow worker velocity corresponds to high a, b values. Hence, wrongly placed boxes increase processing times by large. In the following, many open jobs become late, the combinatorial lower bound can predict their placement better, and thus, its value is higher. The walking strategy influences problem difficulty as well. Except for $v = 16$, instances with walking strategy (S1) are throughout easier to solve. The assembly time variant influences the difficulty in so far that case (L4) is most difficult and a uniform assembly time in (L1) is easiest. At the same time, the uniform box width case (W1) is easiest while case (W3) is hardest. Trivial is case (L1, W1) as the optimum box sequence is the job sequence here. Most difficult is case (L4, W3), its median runtime is about four times the median for all instances with $n = 24$. Still, its median is within the upper quartile. Hence, the median runtime increase is moderate for the most difficult instance class.

In the literature, material fetching accounts for about 10–15% of total work time (Scholl *et al.*, 2013). Let us look at our most realistic choices for worker velocity, $v = 16$ and $v = 8$ (e.g., the case study in Klampfl *et al.* (2006) similarly assumes $v = 13.6$). Our results conform to these values with 9–16% mean walking time of total work time in optimal solutions, see Table 3.

Table 3. B&B median runtime in seconds for instance size $n = 24$ is displayed for all pairs of walking velocity and walking strategy settings. Below is the mean percentage walking time MPW in optimum solutions for different velocities.

	$v = 2$	$v = 4$	$v = 8$	$v = 16$	$v = 32$
S1	0.02	0.42	1.81	3.88	1.74
S2	0.05	0.91	2.06	3.14	2.26
MPW	45%	29%	16%	9%	5%

Table 4. B&B median runtime in seconds for instance size $n = 24$ is displayed for all pairs of assembly time and box width settings.

	L1	L2	L3	L4
W1	0.00	0.16	0.15	1.54
W2	0.32	1.99	1.88	3.17
W3	1.37	2.17	2.33	3.90
W4	0.28	1.26	1.15	2.90

Table 5. Heuristics' runtime and performance on instances that B&B solved in 10 minutes.

	ID		HC		SA		TrB&B	
n	opt	MPE	opt	MPE	opt	MPE	opt	MPE
16	23%	19%	91%	0.31%	94%	0.18%	98%	0.06%
20	19%	20%	89%	0.29%	94%	0.13%	98%	0.06%
24	16%	22%	84%	0.35%	90%	0.13%	96%	0.11%
28	16%	20%	84%	0.23%	89%	0.11%	96%	0.05%
all	18%	20%	87%	0.29%	92%	0.14%	97%	0.07%

opt: percentage of optimally solved instances
MPE: mean percentage error to minimum walking time

8.4. Heuristics

Results for the heuristics are shown in Table 5. It lists median runtimes, fraction of optimally solved instances, and mean percentage error MPE, which is the mean of the percentage walking time error

$$\mathrm{PE} = \frac{\phi - \phi^*}{\phi^* - \sum_{j \in J} l_j} \cdot 100\%,$$

comparing achieved and minimum walking time, where ϕ^* is the optimum and ϕ the heuristic's objective.

The ID heuristic orders all boxes in job sequence. Surprisingly, this is optimal in 18% of all instances. However, its use is fairly limited: a MPE of 20% is quite high. In practice, this natural order is often used, hence the motivation for improvement is high. Applying a hill climbing search (HC) on this greatly improves the result. Then, the number of optimally solved instances raises to 87% while the MPE drops to 0.29%. Moreover, the median computation time is still not measurable. This result improves further with a

simulated annealing search that temporarily allows worse solutions. Here, the number of solved instances increases to 92% and decreases the MPE to 0.14%. This at least increases median computation time to 0.013 seconds.

A different approach is taken by the TrB&B heuristic. Here, the solution quality further improves by a fairly large amount by optimally solving 97% instances, with a further reduced MPE of 0.07%. Moreover, the median TrB&B runtime of 0.002 seconds is fairly lower than the runtime of the simulated annealing. Concluding, although the TrB&B heuristic is more elaborate to implement, both runtime and quality results suggest that this effort is worthwhile.

9. Assessment of Placements in Job Order

A standard, intuitive strategy for placing the boxes is ordering them identically to the sequence of jobs. In our numerical study, this is represented by the ID heuristic. The results show it is able to deliver a percentage walking time error $\mathrm{PE}^{\mathrm{ID}}$ below 5% for a third (34%) of the solved instances. However, the error is above 20% for another third (35%). This can correspond to a noticeable financial impact. To address this, practitioners should aim to at least recognize instances of particularly high optimization potential. For that to happen, we contribute an easy to handle criterion that allows for an evaluation even without performing a full optimization.

9.1. Financial Impact

In the following, we estimate the process cost savings of rearranging intuitively placed boxes with the model in Gaukler and Hausman (2008) for a sample automotive assembly with 80 stations. Clearly, production processing time directly influences process costs. They define process cost savings per time unit by $\mathrm{PCS}(\bar{l}) = \gamma \cdot \lambda \cdot I \cdot \bar{l}$, with production time cost γ in cost per time unit, number of end products λ per time unit, number of assembly line stations I, and saved time $\bar{l}$ in time units per station. In their numerical study, Gaukler and Hausman (2008) assume $\gamma = \$1/\,\mathrm{min}$, $\lambda = 1/\,\mathrm{min}$, $I = 80$ stations. The cycle time is 1 min. We assume that like in the numerical study, mean total work time percentage error of placement in job order is 3.22% and thus, cycle time decreases by $\bar{l} = 0.0297$ min. Thus, process cost savings amount to $\$2.38/\,\mathrm{min}$. Scaling up to one year, Gaukler and Hausman (2008) assume two shifts per day at 6.5 hours each, and 300 days per year, which gives $\lambda = 234\,000/\mathrm{yr}$. Hence, following this model, estimated process cost savings per year amount on average to $\$556\,044/\mathrm{yr}$ for 80 stations.

9.2. Instance Classification

For a practitioner, it is important to quickly recognize instances where the ID solution is inadequate, bearing noticeable optimization potential. In the following, we introduce

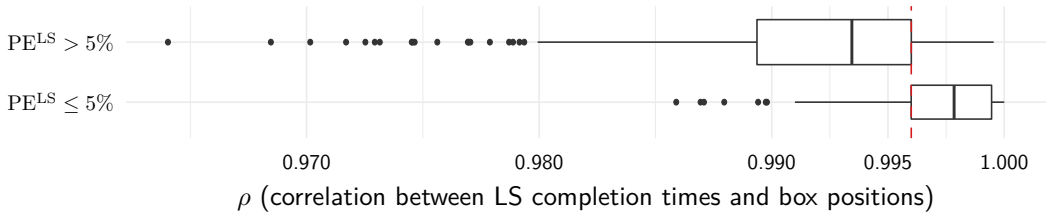

Figure 10. The partition of the observation instances $\mathcal{I}^*_{10\%}$ into $PE^{ID} \leq 5\%$ (an acceptable percentage walking time error of the ID solution) and $PE^{ID} > 5\%$ shows in the box plot that the lower and upper quartiles meet at $\rho = 0.996$.

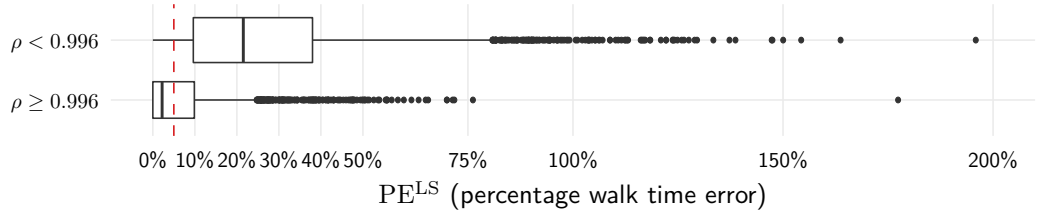

Figure 11. The box plot shows a classification of validation instances $\mathcal{I}^*_{90\%}$ into $\rho \geq 0.996$ (a limit for the correlation coefficient between completion times and box positions in the natural placement) and $\rho < 0.996$ successfully recognizes 77% of the instances with an ID solution's percentage walking time error PE^{ID} above 5%. This allows practitioners to quickly recognize instances where the ID solution of ordering boxes in job order tends to be suboptimal.

a suitable criterion. In particular, we presume the error is small if the points of job completion are close to box positions in the ID solution. This is subsumed by the correlation between completion time and box position. We first analyze only a subset of all instances to avoid overfitting. Let $\mathcal{I}^*_{10\%}$ be a 10% random sample of all instances with known optimum. Spearman's rank correlation coefficient $r_s = -0.652$ shows there is a strong linear relationship with between PE^{ID} and ρ, where

$$\rho = \operatorname{corr}\left(\langle C_j \rangle_{j \in J}, \langle \pi_j \rangle_{j \in J}\right)$$

is the Pearson correlation coefficient between the ID solution's completion times and box positions. We set 5% as an acceptable upper limit for the percentage error of an ID solution. This partitions the instances $\mathcal{I}^*_{10\%}$ into two sets which are shown in a box plot in Figure 10. It shows that in the instance subset with $PE^{ID} \leq 5\%$, the lower (25%) quartile for ρ is 0.996. For the $PE^{ID} > 5\%$ instances, the upper (75%) quartile is 0.996. Thus, the quartiles meet at the same value. Although there is an overlap, this observation hints on using $\rho < 0.996$ as a classifier to decide if an ID solution is suboptimal.

To test this hypothesis, we classify the remaining instances $\mathcal{I}^*_{90\%} = \mathcal{I}^* \setminus \mathcal{I}^*_{10\%}$ by $\rho \geq 0.996$. The resulting partition is depicted in a box plot in Figure 11. Association between two binary variables is commonly measured by the phi coefficient; value $\varphi = 0.48$ indicates a clear relationship between $\rho \geq 0.996$ and $PE^{ID} \leq 5\%$. For $\rho \geq 0.996$, median PE^{ID} is 2%; while for $\rho < 0.996$, median PE^{ID} is 21.5%. Moreover, the upper quantile

$\mathrm{PE}^{\mathrm{ID}}$ is 10% for $\rho < 0.996$ instances, and the lower quantile $\mathrm{PE}^{\mathrm{ID}}$ for $\rho < 0.996$ instances is 9.5%. Hence, the criterion finds a suitable partition, for which the median values are clearly in the right range and moreover, the quartiles meet at a similar values. Of course, there is an error: 15% of the instances are missed by the criterion, wrongly classified as $\mathrm{PE}^{\mathrm{ID}} \leq 5\%$ although they actually have $\mathrm{PE}^{\mathrm{ID}} > 5\%$. This error decreases to 4% unrecognized $\mathrm{PE}^{\mathrm{ID}} < 20\%$ instances. On the other hand, selecting for $\rho < 0.996$ allows to correctly recognize 77% of the instances with $\mathrm{PE}^{\mathrm{ID}} > 5\%$, and 88% of the instances with $\mathrm{PE}^{\mathrm{ID}} > 20\%$.

For an illustration, let us apply it to the example instance from Figure 1 and 2. Its correlation coefficient $\rho = \mathrm{corr}\left(\langle 2, 4, 6, 8\rangle, \langle 4, 8, 9, 10\rangle\right) \approx 0.933$ is well below 0.996. Hence, the criterion indicates optimization potential for the ID solution. Indeed, the actual percentage error $\mathrm{PE}^{\mathrm{ID}} = \frac{8.822-8.584}{8.584-8} \cdot 100\% \approx 41\%$ considerably exceeds 5%.

Concluding, selecting instances with $\rho < 0.996$ successfully filters test instances for which the ID solution of ordering boxes in job sequence is particularly worse. Moreover, this criterion is readily usable by practitioners because all required input is obtained by just measuring completion times and box positions. However, please note that it is required to validate the criterion on present particular conditions, especially if they depart from the range of settings in Section 8.1.

10. Conclusion

We introduce a core box placement problem at a moving assembly line to minimize worker walking times. Although it turns out as NP-hard in the strong sense, we find two polynomial cases which facilitate the construction of a lower bound. A dominance rule allows the direct comparison of partial solutions if they exchange two boxes without affecting others. Moreover, the latter constraint is alleviated by a heuristic dominance rule. By this, we introduce an exact branch and bound search as well as a heuristic version. Numerical results show that both possess superior runtime and quality compared to solving mixed integer programming models and metaheuristic approaches. Moreover, the tests provide convincing evidence for the practical relevance of this approach: compared to the commonly used, intuitive strategy of placing boxes in the order of assembly operations, optimum solutions attain a substantial mean walking time reduction of 20%. Resulting process costs savings alone are estimated at an average of \$556 044/yr for 80 stations. Therefore, we devise a criterion to assess the gap of an instance with just a few measurements, see Section 9.2.

The assessment of model assumptions (see Section 2.2) shows that the constructed model is already close to reality. As it is nonetheless relatively generic, it should also be suited for derivation and extension in several directions. Hence, by shifting the assumptions we suppose it is feasible to expand the model's applicability to further real world scenarios. Such a change may even lead to a shorter walking time: For example, if it is allowed to alter the sequence of assembly operations (removing assumption A4)

in addition to optimizing the box placement in a holistic optimization, the alignment of job start times and box positions can improve further. For this, one can couple the described placement optimization with a job sequence optimization model as in Jaehn and Sedding (2016). Another research direction is extending our model to the widespread *mixed-model* assembly line (removing assumption A3). This enables the assembly of several different products, which alternate between cycles. Such a production system is studied in terms of line balancing and sequencing in Thomopoulos (1967), see Battaïa and Dolgui (2013); Boysen *et al.* (2008, 2009) for recent reviews on both topics. It is in common use in automotive final assembly to produce several car models. Each has specific operations and material requirements. Hence, parts of several products are intermixed along the line. This can impact walking times positively: As the points in time for fetching parts usually differ between products, it is more likely that each product's subset of boxes can be placed close to their ideal positions. However, optimizing this placement is computationally challenging as it impacts walking times for all product sequences at once. To minimize total walking time, it is thus of interest to devise efficient algorithms. In our literature review, we see that already the studied sequencing problem is dissimilar to other scheduling problems. Thus, we find few work to derive from. However, preliminary tests show that a suitable strategy is to extend our approach accordingly.

References

Alidaee, B. and Womer, N.K. (1999) Scheduling with time dependent processing times: Review and extensions. *The Journal of the Operational Research Society*, **50**, 711–720.

Andrés, C., Miralles, C. and Pastor, R. (2008) Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. *European Journal of Operational Research*, **187**(3), 1212–1223.

Baker, K.R. and Scudder, G.D. (1990) Sequencing with earliness and tardiness penalties: A review. *Operations Research*, **38**, 22–36.

Battaïa, O. and Dolgui, A. (2013) A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, **142**, 259–277.

Bautista, J. and Pereira, J. (2007) Ant algorithms for a time and space constrained assembly line balancing problem. *European Journal of Operational Research*, **177**(3), 2016–2032.

Baybars, İ. (1986) A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, **32**(8), 909–932.

Becker, C. and Scholl, A. (2009) Balancing assembly lines with variable parallel workplaces: Problem definition and effective solution procedure. *European Journal of Operational Research*, **199**(2), 359–374.

Błażewicz, J., Dror, M. and Węglarz, J. (1991) Mathematical programming formulations for machine scheduling: A survey. *European Journal of Operational Research*, **51**(3), 283–300.

Boysen, N., Emde, S., Hoeck, M. and Kauderer, M. (2015) Part logistics in the automotive industry: Decision problems, literature review and research agenda. *European Journal of Operational Research*, **242**(1), 107–120.

Boysen, N., Fliedner, M. and Scholl, A. (2008) Assembly line balancing: Which model to use when? *International Journal of Production Economics*, **111**(2).

Boysen, N., Fliedner, M. and Scholl, A. (2009) Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, **192**, 349–373.

Brent, R.P. (1971) An algorithm with guaranteed convergence for finding a zero of a function. *The Computer Journal*, **14**(4), 422–425.

Bukchin, Y. and Meller, R.D. (2005) A space allocation algorithm for assembly line components. *IIE Transactions*, **37**(1), 51–61.

Cheng, T.C.E., Ding, Q. and Lin, B.M.T. (2004a) A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research*, **152**, 1–13.

Cheng, T.C.E. and Gupta, M.C. (1989) Survey of scheduling research involving due date determination decisions. *European Journal of Operational Research*, **38**, 156–166.

Cheng, T.C.E., Kang, L. and Ng, C.T. (2004b) Due-date assignment and single machine scheduling with deteriorating jobs. *Journal of the Operational Research Society*, **55**, 198–203.

Farahani, M.H. and Hosseini, L. (2013) Minimizing cycle time in single machine scheduling with start time-dependent processing times. *The International Journal of Advanced Manufacturing Technology*, **64**(9), 1479–1486.

Finnsgård, C., Wänström, C., Medbo, L. and Neumann, W.P. (2011) Impact of materials exposure on assembly workstation performance. *International Journal of Production Research*, **49**(24), 7253–7274.

Garey, M.R. and Johnson, D.S. (1979) *Computers and intractability*, Series of books in the mathematical sciences, W.H. Freeman.

Gaukler, G.M. and Hausman, W.H. (2008) RFID in mixed-model automotive assembly operations: Process and quality cost savings. *IIE Transactions*, **40**(11), 1083–1096.

Gawiejnowicz, S. (2008) *Time-dependent Scheduling*, Monographs in Theoretical Computer Science, Springer.

Gordon, V.S., Proth, J.M. and Chu, C. (2002a) Due date assignment and scheduling: SLK, TWK and other due date assignment models. *Production Planning & Control*, **13**(2), 117–132.

Gordon, V.S., Proth, J.M. and Chu, C. (2002b) A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research*, **139**, 1–25.

Gordon, V.S., Proth, J.M. and Strusevich, V.A. (2004) Scheduling with due date assignment, in *Handbook of Scheduling*, J.Y.T. Leung, (ed.), Chapman&Hall/CRC.

Gordon, V.S., Strusevich, V.A. and Dolgui, A. (2012) Scheduling with due date assignment under special conditions on job processing. *Journal of Scheduling*, **15**, 447–456.

Gusikhin, O., Klampfl, E., Rossi, G., Aguwa, C., Coffman, G. and Martinak, T. (2003) E-workcell: A virtual reality web-based decision support system for assembly line planning, in *Enterprise Information Systems IV*, M. Piattini, J. Filipe and J. Braz, (eds), Kluwer Academic Publishers, 4–10.

Hall, N.G., Kubiak, W. and Sethi, S.P. (1991) Earliness–tardiness scheduling problems, II: Deviation of completion times about a restrictive common due date. *Operations Research*, **39**, 847–856.

Jaehn, F. and Sedding, H.A. (2016) Scheduling with time-dependent discrepancy times. *Journal of Scheduling*, **19**(6), 737–757.

Józefowska, J. (2007) *Just-in-time scheduling*, vol. 106 of *International series in operations research & management science*, Springer.

Kaminsky, P. and Hochbaum, D.S. (2004) Due date quotation models and algorithms, in *Handbook of Scheduling*, J.Y.T. Leung, (ed.), Chapman&Hall/CRC.

Keha, A.B., Khowala, K. and Fowler, J.W. (2009) Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering*, **56**(1), 357–367.

Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. (1983) Optimization by simulated annealing. *Science*, **220**(4598), 671–680.

Klampfl, E., Gusikhin, O. and Rossi, G. (2006) Optimization of workcell layouts in a mixed-model assembly line environment. *International Journal of Flexible Manufacturing Systems*, **17**(4), 277–299.

Pinedo, M.L. (2016) *Scheduling: Theory, Algorithms, and Systems*, Springer.

Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P. (1992) *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, NY.

Salveson, M.E. (1955) The assembly line balancing problem. *Journal of Industrial Engineering*, **6**(3), 18–25.

Scholl, A. and Becker, C. (2006) State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, **168**(3), 666–693.

Scholl, A., Boysen, N. and Fliedner, M. (2013) The assembly line balancing and scheduling problem with sequence-dependent setup times: problem extension, model formulation and efficient heuristics. *OR Spectrum*, **35**(1), 291–320.

Shabtay, D. (2016) Optimal restricted due date assignment in scheduling. *European Journal of Operational Research*, **252**(1), 79–89.

Thomopoulos, N.T. (1967) Line balancing-sequencing for mixed-model assembly. *Management Science*, **14**(2), B59–B75.

Wan, L. and Yuan, J. (2013) Single-machine scheduling to minimize the total earliness and tardiness is strongly NP-hard. *Operations Research Letters*, **41**, 363–365.

Wee, T. and Magazine, M.J. (1982) Assembly line balancing as generalized bin packing. *Operations Research Letters*, **1**(2), 56–58.

Yin, Y., Liu, M., Cheng, T.C.E., Wu, C.C. and Cheng, S.R. (2013) Four single-machine scheduling problems involving due date determination decisions. *Information Sciences*, **251**, 164–181.