OR Spectrum (2019) 41:727–756
https://doi.org/10.1007/s00291-019-00553-0

REGULAR ARTICLE

The cafeteria problem: order sequencing and picker routing in on-the-line picking systems

David Füßler[1] **· Stefan Fedtke**[1] **· Nils Boysen**[1]

Received: 14 August 2018 / Accepted: 15 March 2019 / Published online: 22 March 2019

Abstract
This paper is dedicated to the cafeteria problem: given a single waiter operating multiple counters for different dishes arranged along a line and a set of customers with given subsets of dishes they desire, find a sequence of customers, which may not overtake each other, and a service schedule for the waiter, such that the makespan is minimized. This generic problem is shown to have different real-world applications in order picking with blocking restrictions. We present different heuristic and exact solution procedures for both problem parts, i.e., customer sequencing and waiter scheduling, and systematically compare these approaches. Our computational results reveal that the largest performance gains are enabled by not strictly processing order after order. Instead, the waiter should be allowed to flexibly swap between customers waiting along the line. Such a flexible service policy considerably reduces the makespan and the total walking distance of the waiter.

Keywords Facility logistics · Blocking · Order sequencing · Scheduling

1 Introduction

Non-crossing constraints, e.g., among quay cranes processing container vessels in harbors, have gained plenty attention in recent years. The survey paper of Boysen et al. (2017), for instance, documents the great research effort in this field. The focus of this research area is on obstructions among the processors of a service process, e.g., among

✉ Nils Boysen
nils.boysen@uni-jena.de

David Füßler
david.fuessler@uni-jena.de

Stefan Fedtke
stefan.fedtke@uni-jena.de

[1] Friedrich-Schiller-Universität Jena Lehrstuhl für Operations Management, Carl-Zeiss-Str. 3, 07743 Jena, Germany

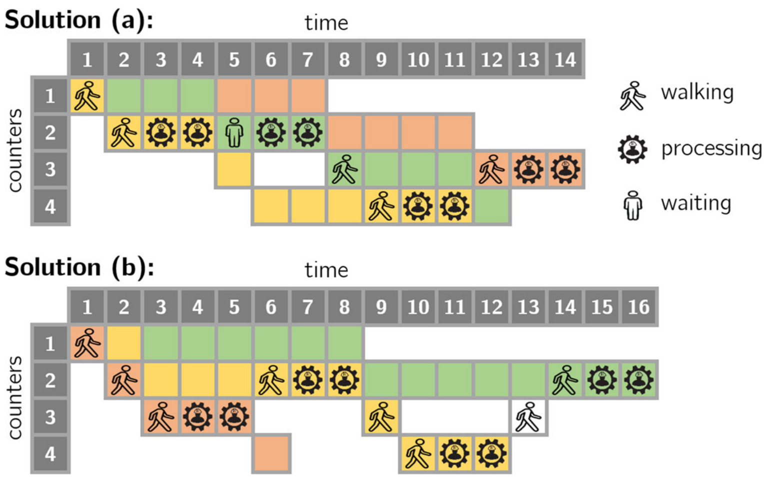
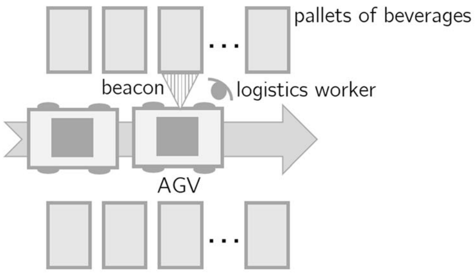


Fig. 1 Example for the cafeteria problem

cranes interfering with each other when operating along the quay wall. However, there exist other processes where rather the recipients of a service face obstructions among each other. In this context, the paper on hand investigates a general problem setting, which we call the *cafeteria problem*:

Consider a cafeteria consisting of multiple counters arranged along a straight line each providing a unique dish (or beverage). These counters are served by a single waiter, who walks along the line and requires a deterministic service time that varies from counter to counter. Furthermore, we have a set of customers each demanding a given individual meal defined by the subset of counters to be visited. Initially, all customers queue at the start of the line and, then, enter the service area one after another. They are only allowed to either wait behind another customer or move in forward direction. To ensure a clearly arranged process (or due to limited space), customers cannot overtake each other, if their way toward their next counter is blocked by a preceding customer waiting for service in front of an earlier counter. They have to wait for the processing of their predecessor and are, thus, blocked. We aim at a sequence of customers entering the service area and at a detailed service schedule in which the customers' counter visits are processed by the waiter, such that the makespan is minimized. The makespan is reached once the waiter, who is typically the bottleneck resource in a cafeteria process, has readily serviced the final dish.

Example Consider a service area consisting of four successive counters operated by a single waiter (represented by the icons in Fig. 1), where three customers have to be served. The red, yellow, and green customer demand the dishes of counters three, two and four, and two, respectively. Waiter and customers walk one counter per time unit, and processing at each counter, where both waiter and customer have to be present, takes two time units. Figure 1 depicts two alternative solutions for this example instance of the cafeteria problem. Solution (a) with customer sequence ⟨yellow, green, red] results in a makespan of 14 time units, whereas in solution (b) customer sequence ⟨red, yellow, green] is not completed before time unit 16 has elapsed.



(a) AGV-assisted order picking of beverages

(b) CaddyPick™ by Swisslog

Fig. 2 Applications for the cafeteria problem

1.1 Applications

Our cafeteria is just a placeholder for quite a few real-world applications. Related problems may also occur in the context of electronic circuit board assembly along a line of successive automatic insertion machines [see Weber and Weiss (1994)], or order picking in warehouses with narrow aisles [see Gue et al. (2006); Hong et al. (2012)]. However, our work was inspired by the following two examples we recently encountered in distribution centers.

- In a German distribution center that supplies liquor stores, supermarkets, and large restaurants with beverages, an *AGV-assisted order picking process* is organized as follows (see Fig. 2a). The inventory of palletized beverages is stored in successive slots arranged along a picking aisle. Automated-guided vehicles (AGVs) travel along the one-way path, and each vehicle conveys one or multiple trolleys dedicated to a specific customer. Whenever an AGV reaches a beverage requested by its customer, the vehicle stops, indicates the beverage with a beacon, and displays the number of requested crates on a display. Then, the logistics worker (waiter) operating the line segment approaches the AGV, loads the requested number of crates on the vehicle, accredits the processed order line, and the AGV continues its travel along the path. During the loading process, successive AGVs are blocked since overtaking in the narrow picking aisle is not possible. Due to tight delivery schedules the managers of the distribution center aim at an efficient picking process. This general aim can be operationalized by minimizing the makespan, i.e., the point in time when the last customer order is processed. However, the worker serving the AGVs suffers from considerable walking distances during a work shift. Thus, reducing the physical burden of their workers is another practical aim.
- Almost the same picking process can also be realized by *carriers hanging from a monorail*. The CarryPick^{TM} system of Swisslog (see Fig. 2b) is one example for such a system. With reference applications at a German drugstore chain and a Swiss supermarket chain (Swisslog 2017), each carrier carries a pallet dedicated to a specific store (customer) and automatically stops in front of a requested stock keeping unit (SKU). In addition to a screen and an LED dot indicating the current picking position to the picker (waiter), carriers can also be equipped with an

automatic weighing mechanism to further reduce picking errors (Swisslog 2017). The carriers utilize the same monorail for their movement along the picking path, so that they cannot overtake each other. Thus, we have to solve our cafeteria problem, i.e., order sequencing plus picker scheduling, to ensure an efficient picking process.

It can be concluded that our generic cafeteria problem occurs as a building block in quite a few relevant applications.

1.2 Literature review

The name *cafeteria process* has previously been applied by Weber and Weiss (1994) to denominate a related stochastic process along successive servers. In their process, customers also block each other, but demand service at only a single station, this being station i with probability p_i. In our distribution center context, however, customers announce their demands ahead by submitting their orders, so that we have a deterministic problem. Furthermore, we also include the movement of the waiter servicing our counters. Thus, to the best of the authors' knowledge our cafeteria problem has not been treated in the scientific literature before. We, therefore, only review related fields that share some similarities with our problem. First, we review two fields of application with similar scheduling problems.

Blocking among order pickers when routing them through a warehouse has already been considered by Gue et al. (2006), Hong et al. (2012), and Chen et al. (2013). The pickers blocking each other while moving in a narrow aisle equal the customers blocking each other along the counters of the cafeteria. However, while pickers can instantaneously access a shelf (once they are not blocked) without any further resource, there is no self-service in our cafeteria and customers have to be served by the waiter. Integrating the waiter considerably complicates the problem setting, so that previous warehousing research is not transferable.

Another related field of application is the scheduling of quay cranes in container ports, which was first described by Daganzo (1989). Here, quay cranes (un-)load a deterministic or stochastic set of ships with some objective, e.g., minimizing the sum of the ships' waiting costs. The manifold exact and heuristic solution methods developed in this area are, for instance, summarized by the recent surveys of Bierwirth and Meisel (2010, 2015). Again, there is a major difference of this field of research to our cafeteria problem. In quay crane scheduling, the processors, i.e., the cranes, are mobile whereas the recipients, i.e., the ships, remain immobile during the service process. In the cafeteria, both the waiter and the customers are mobile. Consequently, the non-crossing constraints of both problems are different [for a survey on scheduling under non-crossing constraints, see Boysen et al. (2017)]. In ports the cranes, i.e., the processors, interfere, whereas in our problem it is the recipients of the service process blocking each other. Again, the research of this area is not directly transferable to our problem.

From a structural point of view, our cafeteria problem is closely related to flow-shop scheduling [for a first survey and elementary complexity results see, for instance, Graham et al. (1979) and Garey et al. (1976)]. In one problem variant, i.e., the permutation flow-shop problem, jobs cannot change their position in a (global) production sequence, so that all production stages process jobs in exactly the same sequence (Ruiz and Maroto 2005). In the cafeteria problem, customers (corresponding to the jobs of the flow-shop) can also not overtake each other and thus, have to enter all counters (corresponding to the production stages) in the same sequence. Note that if a customer does not require a specific dish at a counter, the processing time at the corresponding production stage can simply be set to zero [missing operation, see Hefetz and Adiri (1982)] in flow-shop scheduling. Therefore, the sequencing part of the cafeteria problem resembles the permutation flow-shop problem.

However, customer sequencing is just one part of our cafeteria problem, so that we have to check whether existing extensions of basic flow-shop scheduling are flexible enough to model the remaining part of our problem too. Transportation times of the jobs when moving between stages are considered by Sawik (1995), Tang et al. (2002), Xuan and Tang (2007). They can be applied to model the movement of customers between counters. Also blocking between jobs has been considered in the flow-shop context, e.g., in Liu et al. (2008), Ribas et al. (2011). Here, limited buffers between production stages lead to obstructions, because a job finished at its current production stage may block this machine until the successive machine is free. Note, however, that the blockings occurring in the cafeteria are even more complicated. The counters each have a length and each customer occupies space along the line of counters, so that multiple customers waiting behind a blocked counter build a queue that may block multiple preceding counters too. Even with transportation times and blockings, there is still no counterpart to our waiter in the basic flow-shop scheduling problem. Additional human resources operating the machines, however, have also been considered by machine scheduling research. An additional human operator required to load each job onto its machine during a setup time has been considered by Bruckner et al. (2002), Brucker et al. (2005), Cheng et al. (1999), Glass et al. (2000), Hall et al. (2000), Wang and Cheng (2001). In our setting, however, the waiter has to be present during the whole processing time. For job-shop scheduling, i.e., jobs may vary in the succession of machines they have to visit, and multiple human operators with infinite velocity when moving between machines such a setting has been considered by Agnetis et al. (2011, 2014) and Mencia et al. (2013). A similar setting in an open-shop environment has been considered by Ciro et al. (2016). We, however, have only a single human operator, i.e., our waiter, moving along the line of counters with a given finite velocity. Furthermore, we have a permutation flow-shop environment and machine transfers subject to limited transportation times and blocking. To the best of the authors' knowledge, such a problem has not yet been treated in the machine scheduling literature.

It can, thus, be concluded that our cafeteria problem constitutes a novel problem that requires dedicated solution procedures.

1.3 Contribution and paper structure

This paper is dedicated to solving the cafeteria problem with optimization procedures. In the distribution center for beverages we encountered (see Sect. 1.1), the process was not optimized, but the picker in each aisle had to operate according to a very simple policy. Customer orders are sequenced according to the first-come-first-served (FCFS) policy. Given this sequence, the picker just processes order after order. Starting with the first order of the sequence, the picker accompanies the AGV through the picking line and loads the demanded crates until the current order is completed. Then, the picker returns to the next AGV with the subsequent order, which is again accompanied until completion, and so on until all orders are processed. We call this order sequencing approach the *random approach* and the waiter scheduling policy the *order-by-order* policy. Note that solution (b) of Fig. 1 results from the order-by-order policy for the given customer sequence. Naturally, this leaves two levers for an optimization procedure to improve this process:

- One problem of the order-by-order policy is that the picker always traverses (almost) the complete picking aisle with each order. This leads to long unproductive walking times of the bottleneck resource ‘picker’. Instead, the picker could flexibly swap between orders, which we dub the *order-swapping policy*. Instead of completing order after order, the picker can process neighboring stops of different orders before moving to another section of the line where other orders are waiting for processing. The potential improvement is exemplified by solution (a) of Fig. 1, where the picker starts with the yellow customer at counter two, then proceeds with the green customer at the same counter, before moving onwards to counter 4 where the yellow customer is finished. We present solution procedures for the order-swapping policy, if customer sequences are already given in Sect. 2. Note that the order-by-order policy completely specifies the picker movement once the order sequence is given, so that no optimization procedure for scheduling the waiter is required under this policy. We benchmark both approaches in Sect. 4.
- The second lever for a more efficient service process is the customer sequence. Instead of a random processing of customers according to FCFS, finding the right customer sequence can be part of the optimization problem, which we treat in Sect. 3. Once a suited optimization approach is available, we can apply this procedure to quantify its benefit compared to the random approach.

The remainder of the paper is structured as follows: Section 2, first, addresses the waiter scheduling once the customer sequence is given. We formulate the resulting problem, prove computational complexity, and present suited exact and heuristic solution procedures. Then, Sect. 3 extends our view on the cafeteria problem and also integrates the sequencing of customer orders. In a comprehensive computational study, we benchmark our algorithms (see Sect. 4) and evaluate the impact of our two levers for an efficient service process (see Sect. 5). Finally, Sect. 6 concludes the paper.

2 Scheduling the waiter for given customer sequences

For a given sequence of customers our cafeteria problem is left with the decision on the waiter’s processing sequence of the customers’ dish requests, i.e., routing the waiter to the respective counters. We dub this subproblem the *cafeteria waiter scheduling problem* (CWSP). This section tackles the CWSP by describing possible routing approaches and introducing suitable solution procedures.

Recall that a common method in warehousing practice is to schedule the waiter via the *order-by-order* policy. Under this policy, the waiter serves each customer separately in the order they arrive. Starting with the first dish request, the waiter accompanies each customer along the cafeteria line until the last dish is served and the customer's meal is complete. Afterward, she returns to the next customer in line. This process is repeated until all customers are served. On the positive side, this straightforward policy offers a very simple scheduling approach that does not require any form of optimization and can easily be communicated to the picker. On the other hand, escorting each customer from counter to counter results in excessive (unproductive) walking time for the waiter, which may increase the makespan.

Another approach for scheduling the waiter is the *order-swapping* policy. Instead of serving each customer separately, the waiter can swap between customers and serve dish requests in an arbitrary order (as long as they do not block each other). This policy promises better solutions, due to the additional flexibility given to the waiter. However, in order to determine a good or even an optimal waiter schedule suited solution procedures are required. The following sections investigate the CWSP under the order-swapping policy in detail. After a problem description in Sects. 2.1, Sects. 2.2–2.4 are devoted to different solution methods.

2.1 Problem definition

Our cafeteria waiter scheduling problem (CWSP) under the order-swapping policy is defined as follows: Let $D = \{1, \dots, m\}$ be a set of m dishes each served at a separate counter. Without loss of generality, we assume that the dishes are numbered according to their position along the line, i.e., dish d is served at the d'th counter. In line with our warehousing example, we assume that all counters have an identical size (e.g., the width of a standardized euro-pallet) and are successively arranged without gap along the cafeteria line. Furthermore, let $C = \{1, \dots, n\}$ be the given set of n customers, numbered according to the given customer sequence in which they are processed. Thus, customer 1 is the first to enter the line and so on until, finally, customer n moves into the line. Each customer i orders a meal defined by a set $O_i = \{1, \dots, m_i\}$ of m_i dishes, again, numbered in the order of their line position. The set $\Phi = \{(i,k) | i \in C, k \in O_i\}$ includes the dish requests of all customers, and $d_{i,k} \in D$ specifies the k-th dish requested by customer i. Note that $d_{i,k}$ also corresponds to the counter index and the counters position in the line. Due to the numbering of orders and the premise that customers cannot overtake each other, precedence constraints restricting the waiter's processing sequence of dish requests can be preprocessed. Parameter $\lambda_{(i,k),(j,l)}$ defines these relations, i.e., obtains value 1, if (i,k) has to be a predecessor of (j,l) and value 0, otherwise:

$$\lambda_{(i,k),(j,l)} = \begin{cases} 1, & \text{if } i = j \wedge d_{i,k} < d_{i,l} \quad (1) \\ & \text{or } i < j \wedge d_{i,k} \leq d_{j,l} \quad (2) \\ & \text{or } i < j \wedge d_{j,l} < d_{i,k} \leq d_{j,l} + (j - i - 1) \quad (3) \\ 0, & \text{else} \quad (4) \end{cases}$$

These precedence constraints define that dish $d_{i,k}$ has to be serviced by the waiter before dish $d_{j,l}$, if i and j refer to the same customer and k-th dish is available at an earlier counter [see (1)]. If i refers to an earlier customer than j (according to the given customer sequence), then any dish for earlier customer i served at an earlier counter blocks dishes demanded from later counters (or the same counter) of later customer j [see (2)]. If later customer j demands a dish served at an earlier counter compared to earlier customer i's current dish, then the tailback of customers queueing behind i and blocking access to the counter has to be considered [see (3)]. Again, in line with our warehousing example, we assume that each customer requires identical space which equals the length of a single counter (i.e., the width of a euro-pallet). Thus, if two customers queue behind another customer waiting at counter 5, then counters 3–5 are blocked for another customer arriving at the queue. Finally, no precedence constraints exist, if neither of these previous conditions holds [see (4)]. The set $\Lambda_{(j,l)} = \{(i,k) | \lambda_{(i,k),(j,l)} = 1\}$ contains all predecessors of (j,l).

Given these precedence constraints, we seek a waiter schedule defined by a processing sequence $\pi = (\pi_1, \ldots, \pi_T)$, with $T = |\Phi|$, in which all dish requests are serviced by the waiter. Thus, $\pi_t \in \Phi$, defines the dish served by the waiter at *service period* (or sequence position) t within π. Such a solution is called feasible, if

- for each $(i,k) \in \Phi$ there is exactly one service period $t \in \{1, \ldots, T\}$ with $\pi_t = (i,k)$ in π, that is each dish request of customers has to be served exactly once by the waiter,
- for each service period $t \in \{1, \ldots, T\}$ there is exactly one $(i,k) \in \Phi$ with $\pi_t = (i,k)$ in π, that is the waiter has to serve a dish request in each service period, and
- if $\lambda_{(i,k),(j,l)} = 1$, dish request (i,k) has to be served before dish request (j,l), that is all precedence relations are satisfied.

For a given waiter schedule π, we are able to determine the position $p_{i,t} \in \mathbb{R}$ of each customer i during each service period t of the service process as well as the amount of (actual) time $\Delta_t \in \mathbb{R}_{\geq 0}$ required by each service period.
Customer positions At the beginning of our planning horizon, i.e., in $t = 0$, all customers line up in front of the cafeteria. That is $p_{i,0} = 1 - i$ for each $i \in C$. The positions of each customer during the service process solely depend on the waiter schedule π. Customers move along the line as far as possible and stop at the next counter that serves a desired dish. If a dish is to be served in service period t, the customer has to be positioned at the respective counter:

$$\pi_t = (i,k) \rightarrow p_{i,t} = d_{i,k}. \tag{5}$$

Furthermore, customers can neither move faster than their given speed v^{cust} allows, nor overtake other customers in front of them, nor pass the next relevant counter:

$$\pi_t \neq (i,k) \rightarrow p_{i,t} = \begin{cases} \min\left\{p_{i,t-1} + \Delta_t \cdot v^{\text{cust}}; p_{i-1,t} - 1; d_{i,k}\right\}, & \text{if } (i,k) \text{ has not yet been served} \\ \min\left\{p_{i,t-1} + \Delta_t \cdot v^{\text{cust}}; p_{i-1,t} - 1\right\}, & \text{else.} \end{cases} \tag{6}$$

Service time The actual time required for processing a dish in service period t depends on the waiter walking speed v^{wait}, the customer moving speed v^{cust}, and the serving time $\tau_{i,k}$ for the respective dish (i, k). If either the waiter or the customer reaches the next relevant counter first, she has to wait for the other one in order to serve the respective dish. Therefore, the maximum of waiter walking time and customer moving time has to be determined and added to the serving time:

$$\Delta_t = \max\left\{\frac{d_{i,k} - p_{i,t-1}}{v^{\text{cust}}}; \frac{|d_{i,k} - d_{i',k'}|}{v^{\text{wait}}}\right\} + \tau_{i,k}, \tag{7}$$

where dish (i', k') is the dish served in service period $t-1$. Among all feasible solutions CWSP seeks one waiter schedule π that minimizes the total processing time of all customer orders

$$F(\pi) = \sum_{t=1}^{T} \Delta_t. \tag{8}$$

Due to the structure of our problem, objective value $F(\pi)$ corresponds to the makespan, i.e., the point in time when all orders of the given order sequence are readily serviced.

Theorem 1 *CWSP is strongly NP-hard.*

Proof See "Appendix A". □

Note that presupposing varying walking speeds of waiter and customers may seem superfluous, because in a real-world cafeteria we only have human actors with similar speed. However, in our warehouse application human waiters and mechanical AGVs may have different speeds, so that in this application differentiating speeds is essential.

2.2 Mixed-integer programming model

To solve our CWSP to optimality with a standard MIP solver, we introduce a mixed-integer program first.

Given the notation summarized in Table 1, a mixed-integer program for our CWSP is defined by objective function (9) and constraints (10)–(22):
CWSP-MIP

$$\text{Minimize } F(Z, P, \Delta) = \sum_{t=1}^{T} \Delta_t \tag{9}$$

subject to

$$\sum_{t=1}^{T} z_{i,k,t} = 1 \quad \forall (i, k) \in \Phi \tag{10}$$

Table 1 Notation for CWSP

D	Set of dishes with $D = \{1, \ldots, m\}$
C	Set of customers with $C = \{1, \ldots, n\}$
O_i	Set of dish requests of customer i with $O_i = \{1, \ldots, m_i\}$ and $d_{i,k} < d_{i,k+1} \; \forall k = 1, \ldots, m_i - 1$
Φ	Set of all dish requests with $\Phi = \{(i,k) \mid i \in C, k \in O_i\}$
T	Number of service periods with $t = 1, \ldots, T = \lvert\Phi\rvert$
$d_{i,k}$	Dish required by dish request (i,k) with $d_{i,k} \in D$
$\tau_{i,k}$	Processing time of dish request (i,k)
v^{wait}	Walking speed of the waiter
v^{cust}	Moving speed of customer
M	Big integer, e.g., $M = m + n + 1$
$z_{i,k,t}$	Binary variable: 1, if dish request (i,k) is executed in service period t; 0, else
$p_{i,t}$	Continuous variable: position of customer i at the end of service period t
Δ_t	Continuous variable: actual duration of service period t

$$\sum_{(i,k)\in\Phi} z_{i,k,t} = 1 \quad \forall t = 1, \ldots, T \tag{11}$$

$$\Delta_t \geq \left(\frac{d_{i,k} - p_{i,t-1}}{v^{\text{cust}}} + \tau_{i,k} \right) - M \cdot (1 - z_{i,k,t}) \quad \forall t = 1, \ldots, T;\ (i,k) \in \Phi \tag{12}$$

$$\Delta_1 \geq \sum_{i \in O} \left(\frac{d_{i,1}}{v^{\text{wait}}} + \tau_{i,1} \right) \cdot z_{i,1,1} \tag{13}$$

$$\Delta_t \geq \frac{\sum_{(i,k)\in\Phi} d_{i,k} \cdot z_{i,k,t} - \sum_{(i,k)\in\Phi} d_{i,k} \cdot z_{i,k,t-1}}{v^{\text{wait}}} + \sum_{(i,k)\in\Phi} \tau_{i,k} \cdot z_{i,k,t} \quad \forall t = 2, \ldots, T \tag{14}$$

$$\Delta_t \geq \frac{\sum_{(i,k)\in\Phi} d_{i,k} \cdot z_{i,k,t-1} - \sum_{(i,k)\in\Phi} d_{i,k} \cdot z_{i,k,t}}{v^{\text{wait}}} + \sum_{(i,k)\in\Phi} \tau_{i,k} \cdot z_{i,k,t} \quad \forall t = 2, \ldots, T \tag{15}$$

$$p_{i,t} \leq d_{i,1} + M \cdot \left(\sum_{t'=1}^{t-1} z_{i,1,t'} \right) \quad \forall i = 1, \ldots, n; \quad t = 1, \ldots, T \tag{16}$$

$$p_{i,t} \leq d_{i,k} + M \cdot \left(1 - \sum_{t'=1}^{t-1} z_{i,k-1,t'} \right) + M \cdot \left(\sum_{t'=1}^{t-1} z_{i,k,t'} \right) \quad \forall i = 1, \ldots, n; \quad k = 2, \ldots, m_i; \quad t = 1, \ldots, T \tag{17}$$

$$p_{i,t} \leq p_{i,t-1} + \Delta_t \cdot v^{\text{cust}} \quad \forall i = 1, \ldots, n; t = 1, \ldots, T \tag{18}$$

$$p_{i,t} \leq p_{i-1,t} - 1 \quad \forall i = 2, \ldots, n; t = 1, \ldots, T \tag{19}$$

$$p_{i,t} \geq p_{i,t-1} \quad \forall i = 2, \ldots, n; t = 1, \ldots, T \tag{20}$$

$$p_{i,t} \geq d_{i,k} - M \cdot (1 - z_{i,k,t}) \quad \forall (i,k) \in \Phi;\ t = 1, \ldots, T \tag{21}$$

$$z_{i,k,t} \in \{0, 1\} \quad \forall (i,k) \in \Phi;\ t = 1, \ldots, T \tag{22}$$

Our objective function (9) minimizes the makespan, i.e., the point in time when all customers are served. Constraints (10) and (11) ensure that each dish request is executed exactly once and that a dish is served within each service period. The actual processing time required by service period t, Δ_t, is defined by (12)–(15). First, Δ_t depends on the time required by the customer to move from its initial position to the relevant counter, stated in (12). Moreover, the waiter has to move from the counter, where she served the last dish, to the next relevant counter, stated in (13)–(15). In both cases, the processing time $\tau_{i,k}$ at the respective counter is added too. The exact positions of customers throughout the process are modeled via constraints (16)–(21). Hereby, (16) and (17) ensure that the customer served in period t does not pass the relevant counter. The moving speed of customers is considered in inequalities (18). Due to (19), customers are not able to overtake each other, and due to (20), they can only move in forward direction. Furthermore, constraints (21) ensure that customers have reached their targeted counter whenever the waiter serves them a demanded dish. Finally, (22) define the domain of the variables. Note that additional constraints to ensure the precedence relations $\lambda_{(i,k),(j,l)}$ among dish requests are redundant due to the exact modeling of customer positions.

2.3 A dynamic programming procedure

This section presents an alternative to solve our CWSP to optimality with the help of a dynamic programming (DP) procedure.

Our DP is composed of $|T| + 1$ stages, each representing a service period, i.e., processing a dish request of a customer by the waiter (plus a virtual initial stage). Each stage contains states $Z = (J, w, p)$ with $J \subseteq \Phi^0 = \Phi \cup \{(0, 0)\}$, $w \in \{0, \ldots, m\}$, and $p \in ([1 - n, m] \cup \{\infty\})^n$ defining the set of already served dish requests, the current waiter position along the line, and the array of current customer positions, i.e., $p = (p_1, p_2, \ldots, p_n)$, respectively. Note that $p_i < 0$ means that a customer is still queuing in front of the cafeteria and $p_i = \infty$ that she has left the cafeteria. The partial objective value for each state is denoted by $f(Z)$ or $f(J, w, p)$ and represents the completion time of all served dish requests so far. There is a transition from state $Z = (J, w, p)$ to state $Z' = (J', w', p')$, if $\exists\, (j, l) \in \Phi$ with $\Lambda_{j,l} \setminus J = \emptyset$ and

- $J' \setminus J = \{(j, l)\}$, that is an additional dish request is served during the service period,
- $w' = d_{j,l}$, that is the waiter is currently at the correct counter, and
- the customer movement is well defined, i.e.,

$$p' = (p'_i)_{i \in C} \text{ with } p'_j = d_{j,l} \text{ and } p'_i = \min\{\alpha, \beta, \gamma\}\ \forall i \in C, i \neq j \text{ with} \tag{23}$$

$$\alpha = \begin{cases} d_{i,k^*} \text{ with } k^* = \min\{k|(i,k) \in \Phi \setminus J\}, & \text{if } |\{(i,k)|(i,k) \in \Phi \setminus J\}| \geq 1 \\ \infty, & \text{else} \end{cases} \tag{24}$$

$$\beta = \begin{cases} p_i + \Delta(Z, Z') \cdot v^{\text{cust}}, & \text{if } p_i + \Delta(Z, Z') \cdot v^{\text{cust}} \leq m \\ \infty, & \text{else} \end{cases} \tag{25}$$

$$\gamma = p'_{i-1} - 1 \text{ with } p'_0 = \infty, \tag{26}$$

that is the served customer is at the correct counter (23), customers do not pass their next relevant counter (24), customers cannot move faster than their speed allows (25), and customers cannot pass each other (26).

The additional processing time $\Delta(Z, Z')$ associated with such a transition amounts to

$$\Delta(Z, Z') = \max\left\{\frac{|w' - w|}{v^{\text{wait}}}, \frac{d_{j,l} - p_j}{v^{\text{cust}}}\right\} + \tau_{j,l}. \tag{27}$$

Given the initial state $Z_0 = (\emptyset, 0, (0, -1, \ldots, -n))$ with $f(Z_0) = 0$ and the transitions' contribution to the objective value, we have the basic Bellman recursion

$$f(Z') = \min_{\substack{Z=(J,w,p): \\ |J' \setminus J|=1}} \{f(Z) + \Delta(Z, Z')\}. \tag{28}$$

After a stage-wise forward recursion, we can determine the final objective value by comparing all states of the final stage:

$$F = \min_{\substack{Z=(J,w,p): \\ J=\Phi^0}} \{f(Z)\}. \tag{29}$$

Via a simple backward recursion, an optimal order sequence can be extracted.

Regarding the computational effort of our DP, we have a maximum of $O(2^{|\Phi|} \cdot m \cdot m^n)$ states, at most $O(2^{|\Phi|} \cdot m \cdot m^n \cdot |\Phi|)$ transitions, and each transition requires an iteration through all customers n. Thus, we have an exponential worst-case runtime, which is in line with our complexity result.

Example (cont.) Consider the example presented in Fig. 1b. We extend the example by a fourth customer which enters the list first. She demands dishes at counters one and four, followed by customers red (dish three), yellow (dishes two and four), and green (dish two). A brief summary of relevant instance parameters and the resulting DP graph are depicted in Fig. 3. One of two optimal solutions is given by dish request sequence $(1, 1) \rightarrow (2, 1) \rightarrow (1, 2) \rightarrow (3, 1) \rightarrow (4, 1) \rightarrow (3, 2)$ with an optimum makespan of 21.

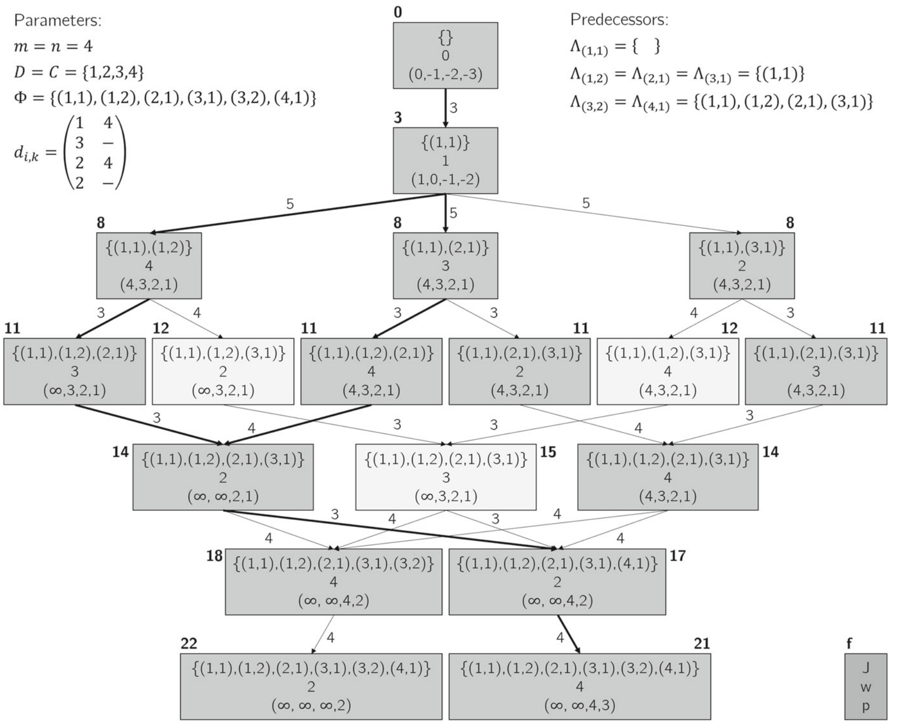


Fig. 3 Dynamic programming/beam search graph for CWSP

2.4 Beam search approach

Because the number of states grows exponentially with the number of dish requests, we modify our DP approach to obtain a heuristic beam search procedure (BS). Beam search requires less computational time and memory as it only branches the ζ (beam width) most promising states, according to their partial objective value, of each stage in the DP tree. Hence, if ζ is sufficiently small, space and CPU time required for solving our CWSP are polynomially bounded. However, it is not guaranteed that the approach finds an optimal solution.
Example (cont.) Consider the graph given in Fig. 3. When applying the beam search procedure with a beam width of $\zeta = 4$ to the same example, we derive a smaller graph without the highlighted states (lighter gray color). In this example, we still find an optimal solution, which is not generally the case.

3 Sequencing the customers

In the distribution center for beverages we visited (see Sect. 1), the customer sequence is not optimized and orders are just processed according to a FCFS policy. Since FCFS does not consider any order characteristics, but simply sequences incoming orders according to their arrival times, we emulate this approach by a random order sequence (dubbed the *random* approach). Beyond this status quo, this section suggests different straightforward optimization procedures for the sequencing part of our cafeteria problem. We seek a suited sequence of customers in which they enter the line of counters. Recall that this sequence remains unaltered during the whole cafeteria process, because customers cannot overtake each other. If the waiter processes customers according to the *order-by-order policy* and strictly serves customer after customer according to their sequence, the whole cafeteria problem reduces to the sequencing of customers. In this case, the whole cafeteria problem can be solved by a modification of the famous algorithm of Gilmore and Gomory (1964) for sequencing transport requests on the line. This is elaborated in Sect. 3.1. Naturally, this order sequence can also be applied under the *order-swapping* policy, where the waiter may flexibly swap between orders. However, we also suggest a simple priority rule-based approach (see Sect. 3.2). Once these solutions procedures are available, we can evaluate whether order sequences are a suited lever for a more efficient cafeteria process (see Sect. 5).

3.1 Iterative Gilmore and Gomory approach

When applying the order-by-order policy, the waiter iteratively serves customers in the same order as they enter the cafeteria. Therefore, each customer i defines a job $\text{job}_i = (d_{i,1}, d_{i,m_i})$ which starts at the first counter $d_{i,1}$ and ends at the last counter d_{i,m_i} she demands a dish from. The goal of our customer sequencing problem (CSP) under the order-by-order policy is to schedule these jobs such that the makespan of the serving process is minimized.

At first glance, this problem resembles a popular special case of the traveling salesman problem (TSP), the TSP on the line. The famous solution procedure of Gilmore and Gomory (1964) solves this special case to optimality in polynomial time [see also Burkard et al. (1998)]. Specifically, it can be applied to sequence transport requests along a line, such that the total travel distance (of the waiter, in our case) is minimized in $O(n^2)$. However, the TSP on the line and our CSP differ in two points:

- The TSP on the line looks for a round trip, i.e., a tour through all jobs along the line with a final return to the start position. Within a solution of our CSP, however, the waiter starts at the beginning of the cafeteria and ends at the last counter she serves a dish at. Thus, we have to solve the path version of the problem, which can be obtained by embedding the approach of Gilmore and Gomory (1964) in an iterative solution procedure with a runtime in $O(n^3)$, dubbed "IGG", given by Algorithm 1.
  Example (cont.) Consider our example of Fig. 1. In solution (b), the waiter follows the order-by-order policy for customer sequence ⟨red, yellow, green], which results in a total walking distance of 8 and a makespan of 16 time units including the 8 time units for serving the four dish requests for two time units each. When applying IGG, we derive solution (c) and customer sequence ⟨green, yellow, red] depicted in Fig. 4 with a total walking distance of 5 and a makespan of 14 time units.
- IGG optimizes the customer sequence according to the walking distance of the waiter. In general, this solution is not optimal according to our makespan objective,

Algorithm 1: Iterative Gilmore-Gomory approach (IGG)

```
input: set of $n$ customer jobs (on the line)
1  add virtual starting job $job_0 = (0, 0)$
2  initialize best customer sequence $\pi^*$
3  foreach customer job $job_i$ do
       // construct optimal sequence, which ends at $d_{i,m_i}$
4      add virtual customer job $job^i_{n+1} = (d_{i,m_i}, 0)$ from the last dish of customer $i$ to the beginning of the cafeteria
5      apply the approach of Gilmore and Gomory (1964) to determine an optimal tour
6      delete $job^i_{n+1}$ from the tour
7      divide the tour at $job_0$ and receive sequence $\pi$
       // check for new best solution
8      if $F(\pi) < F(\pi^*)$ then
9          $\pi^* = \pi$
10     end
11 end
return: best customer sequence $\pi^*$
```

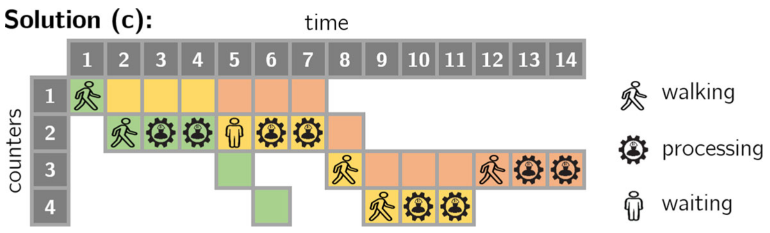


Fig. 4 Gilmore–Gomory solution with minimal waiter walking distance

because the additional travel of the customers is not considered. The optimal IGG solution of Fig. 4, for instance, results in an optimal walking distance of 5 plus the 8 time units for serving the four dishes; this suggests a makespan of 13 time units. The actual makespan, however, is 14 time units, because the single time unit the waiter remains idle until the arrival of the yellow customer at counter two is not included. However, if we presuppose that the customers are fast enough that they always wait at their first counter once the waiter returns to the next customer, then IGG can directly be applied to solve our customer sequencing problem for the order-by-order policy. If this is not the case, IGG serves as a heuristic approach for CSP.

Naturally, IGG can also be applied to determine a heuristic solution for CSP, if the waiter follows the order-swapping policy. The next section, however, introduces an alternative approach.

3.2 A priority rule-based approach considering customer blockings

We tried out quite a few simple priority rule-based approaches, but only report the one working best. This approach is based on the following two simple observations. First, with much more customers than counters, i.e., $n > m$, there tend to be clusters of up to m customers potentially blocking each other. Furthermore, to fairly distribute the blocking over these clusters, each cluster should contain a mix of customers causing many, medium and few potential blockings. To realize these basic characteristics in a customer sequence, we proceed as follows:

First, we quantify the number of potential blockings Ψ_i for each customer $i \in C$. This is done by determining the total number of all other customers $j \in C$, with $j \neq i$, which are potentially blocked. Specifically, we determine $\Psi_i = \sum_{j \in C} \psi_{ij}$, where

$$\psi_{ij} = \begin{cases} 1, & \text{if } d_{i,1} \leq d_{j,1} \\ 0, & \text{else} \end{cases} \quad \forall\, i, j \in C. \tag{30}$$

Then, we sort all customers $i \in C$ according to non-increasing Ψ_i values. Finally, we successively consider this intermediate sequence, assign the j-th customer of this intermediate sequence to cluster $j \mod m$, and bring the resulting clusters into a random sequence, which constitutes our final customer sequence.

Note that our computational study presented in the following section shows that this straightforward approach for customer sequencing is only slightly outperformed by a metaheuristic, i.e., a multi-start simulated annealing procedure, with a runtime of several hours. This metaheuristic is described in "Appendix B." Due to this result, we abstain from presenting even more elaborate sequencing approaches.

4 Computational performance

In this section, we elaborate on the computational performance of our solution procedures. Specifically, we determine the runtimes and optimality gaps for our exact and heuristic solution procedures for differently sized test instances. Before we describe the results of these tests in Sects. 4.2 and 4.3, we, first, elaborate on the generation scheme for deriving our test instances in Sect. 4.1.

4.1 Instance generation

Unfortunately, there exists no established testbed for our cafeteria problem. Therefore, we had to generate our own instances. Specifically, we proceeded as follows: The values listed in Table 2 are combined in a full factorial manner, which leads to 27 unique parameter settings. For each setting, instance generation is repeated 10 times, so that in total 270 instances have been obtained.

Each instance is generated as follows: First, the amount of requested dishes is drawn from interval $[\underline{\gamma}; \overline{\gamma}]$ for each single customer. Then, according to this result, the specific dishes per customer are randomly drawn from the m dishes available. All random numbers follow a uniform distribution. The size of each counter and each customer are normalized to one distance unit, and both waiter and customers move one distance unit per time unit. Note that, later when investigating managerial aspects in Sect. 5, we also address the impact of varying speeds.

Table 2 Parameter values for instance generation

Parameter	Description	Values
n	Number of customers	10, 30, 50
m	Number of counters	10, 30, 50
$[\underline{\gamma}; \overline{\gamma}]$	Interval of requested dishes per customer	[1; 3], [7; 10], [1; 10]

Table 3 Performance criteria

Criterion	Description
#best	Number of best solutions among all procedures
#feas	Number of instances where a feasible solution is found
#opt	Number of instances where a proven optimal solution is found
∅gap	Average gap to best solution in %
∅sec	Average computational time in CPU seconds

All procedures have been coded in Visual Basic (Visual Studio 2012 Ultimate) based on Microsoft's .NET Framework 4.6, and all computations have been performed on a personal computer with Intel Core i7-3770 processor with 4×3.4 GHz clock speed and 8 GB DDR-3 RAM. As a standard solver, we apply Gurobi Optimizer 7.5.

4.2 Performance tests for the CWSP

First, we address the solution approaches dedicated to the waiter scheduling problem CWSP for a given customer sequence (see Sect. 2). Specifically, we benchmark standard solver Gurobi solving CWSP-MIP (see Sect. 2.2), our exact dynamic programming (DP) procedure (see Sect. 2.3), and our beam search (BS) heuristic (see Sect. 2.4) executed with a beam width $\zeta = 300$. Since CWSP is an operational problem with a rather short-term planning horizon, each solution approach receives a timeout of 300 s. Note that for Gurobi we only restrict the (pure) solution time, so that by adding the time required by the standard solver to prepare the model, a total time requirement larger than the time out may result. Also for our DP runtimes larger than the timeout may occur, because we measure time after completing each stage only. For these three competitors, we report the performance criteria listed in Table 3. The performance results summarized in Table 4 suggest the following findings:

- Among the exact solution procedures, Gurobi is clearly outperformed by DP. In total, the latter solves 98 instances to optimality (i.e., 36.3%) within the given runtime and even finds 15 optimal solutions for the largest instances with $n = 50$ customers. Gurobi only finds 28 proven optimal solutions for the smallest instances with $n = 10$ customers, i.e., 10.4%. On the negative side, DP either finds an optimal solution within the given runtime or returns unsuccessfully without a feasible solution, i.e., in 63.7% of all instances. Therefore, we only report #opt

Table 4 Computational performance for CWSP of Gurobi solving CWSP-MIP, dynamic programming (DP), and beam search (BS)

n	m	Gurobi solving CWSP-MIP					DP		BS		
		#best	#feas	#opt	∅gap	∅sec	#opt	∅sec	#best	∅gap	∅sec
10	10	12	30	10	7.55	202.31	30	30.66	29	0.03	1.51
10	30	10	30	9	11.63	220.62	16	198.35	30	0	4.68
10	50	9	30	9	12.27	218.32	11	254.54	30	0	6.33
Total		31	90	28	10.48	213.75	57	161.18	89	0.01	4.17
30	10	0	30	0	35.79	304.29	18	162.56	30	0	23.84
30	30	0	30	0	77.11	308.49	7	300.5	30	0	63.99
30	50	0	30	0	76.28	309.96	1	365.76	30	0	83.29
Total		0	90	0	63.06	307.58	26	276.27	90	0	57.04
50	10	0	30	0	39.43	319.02	15	203.57	30	0	87.61
50	30	0	20	0	90.56	340.64	0	376.46	30	0	172.79
50	50	0	20	0	112.39	345.8	0	396.04	30	0	201.05
Total		0	70	0	80.79	335.16	15	325.36	90	0	153.82

for DP; in all other cases DP fails. Gurobi finds feasible solutions in 92.6% of all instances and only struggles with the very largest instances.

- With regard to the heuristic solution performance, it can be concluded that BS delivers convincing results even for the largest instances of real-world size. It misses just a single optimal solution among those instances where the optimal objective value is known and clearly outperforms the heuristic values obtained by Gurobi for all larger instances. However, BS too requires a considerable solution time for the largest instances, i.e., an average of 154 s over all instances with $n = 50$ customers.

To further benchmark the quality of the solution procedures introduced for waiter scheduling, we extend the computational results and compare lower bounds. Table 5 shows the gap of each solution approach compared to the LP-relaxation, which proved best among the lower bounds we tested. Other lower bounds, e.g., based on the minimum spanning tree-relaxation of the TSP [see Held and Karp (1970)], performed even worse. Unfortunately, already the best lower bound is not tight and for all instances solved to optimality (see Table 4) the gap is about 55%. This gap increases among all instances, see Table 5. Whether more sophisticated lower bounds for the sequential ordering problem, e.g., described in Ascheuer et al. (1993), lead to considerably better results for waiter scheduling remains questionable. It can be concluded that finding tight lower bounds for waiter scheduling is a challenging task and should receive more attention in future research. Preliminary computational tests confirm this finding for the overall cafeteria problem. This result is not astounding, since the cafeteria problem extends waiter scheduling and also includes customer sequencing.

Table 5 Gap to the LP-relaxation of CWSP-MIP for Gurobi solving CWSP-MIP, dynamic programming (DP), and beam search (BS)

n	m	Gurobi solving CWSP-MIP	DP	BS
		∅gap	∅gap	∅gap
10	10	56.44	53.37	53.39
10	30	63.52	59.45	59.71
10	50	64.98	59.89	61
Total		61.65	57.57	58.03
30	10	69.48	61.03	58.67
30	30	81.76	71.48	67.79
30	50	83.83	73.37	71.84
Total		78.36	68.63	66.1
50	10	70.82	62.3	59.32
50	30	84.94	No solution found	69.66
50	50	88.26	No solution found	71.88
Total		81.34	62.3	66.96

4.3 Performance tests for the holistic problem

Next, we skip to the overall cafeteria problem where also determining the customer sequence is part of the problem. Note that we formulated a MIP model for the holistic problem including customer sequencing and waiter scheduling under the order-swapping policy, but, unfortunately, not even for tiny instances with a handful of customers standard solver Gurobi could obtain reasonable results, even with a much larger runtime of several hours. Gurobi even struggled with finding lower bounds, which is in line with our investigation of bound quality for waiter scheduling presented in the previous section. Therefore, we decided to benchmark our three competitors [i.e., a RANDOM sequence, the customer sequence obtained by our iterative Gilmore–Gomory (dubbed IGG, see Sect. 3.1), and our priority rule-based approach (dubbed PRIO, see Sect. 3.2)] against the results of the multi-start simulated annealing metaheuristic (see "Appendix B") executed with a runtime of 2 h. In relation to this benchmark, we report the average gap (∅gap) and the number of best solutions obtained by the respective procedure (#best) for all three competitors in Table 6. Note that all three procedures only evaluate a single customer sequence and solve the remaining waiter scheduling problem (CWSP) under the order-swapping policy with BS and beam width $\zeta = 300$. Thus, there is no (significant) difference in the solution time, and in all three cases, the share of the sequencing parts is negligible compared to the execution time of BS.

The results of Table 6 lead us to the following conclusions. Our priority rule-based approach for determining the customer sequence clearly outperforms both competitors. The worst results are obtained by the random customer sequences, which resemble the status quo in the distribution center we visited. IGG leads to more best solutions and a much smaller average gap, but for the largest instances with $n = 50$ customers and

Table 6 Solution performance for the holistic cafeteria problem (i.e., customer sequencing and waiter scheduling under the order-swapping policy) for competitors random solution (RANDOM), iterative Gilmore–Gomory (IGG), and the priority rule-based approach (PRIO) in relation to the multi-start simulated annealing metaheuristic

n	m	RANDOM		IGG		PRIO	
		#best	∅gap	#best	∅gap	#best	∅gap
10	10	0	13.97	2	5.77	3	4.55
10	30	0	19.3	1	7.1	7	3.69
10	50	0	17.55	0	8.4	12	3.06
Total		0	16.94	3	7.09	22	3.76
30	10	0	10.64	0	2.74	16	0.43
30	30	0	17.69	0	8.02	29	0.04
30	50	0	22.87	0	10.71	30	0
Total		0	17.07	0	7.16	75	0.16
50	10	0	9.59	0	2.54	28	0.09
50	30	0	18.44	0	8.58	30	0
50	50	0	23.44	0	14.06	30	0
Total		0	17.16	0	8.4	88	0.03

$m = 50$ counters the average gap is substantial (i.e., 14.06%). PRIO leads to the best results. In fact, due to the large solution space and the long runtime of BS for solving the waiter scheduling problem, not even metaheuristic mSA with a runtime of 2 h is able to considerably improve these results.

It can be concluded that our BS heuristic seems well suited to solve even large-sized instances of real-world size of the waiter scheduling problem. Moreover, our simple rule-based approach delivers acceptable objective values for the sequencing part of the cafeteria problem. Therefore, we apply these two approaches throughout the further tests elaborated in the following section, if not explicitly stated otherwise.

5 Managerial aspects

This section addresses important managerial aspects and supports managers having to setup and operate a cafeteria system. First, Sect. 5.1 explores the impact of our two levers (i.e., optimized customer sequences instead of random sequences and flexible order swapping of the waiter instead of the order-by-order policy, see Sect. 1.3) on the system performance. Then, we take the waiter's perspective and address the question whether optimized schedules also reduce the total walking distance (see Sect. 5.2). Finally, we explore whether faster AGVs can considerably improve the system performance (see Sect. 5.3).

5.1 Impact of levers

The status quo in the distribution center we consider is to process random customer sequences according to the order-by-order policy. This section explores the performance gains if optimized order sequences are applied and the waiter follows the more flexible order-swapping policy instead. Table 7 summarizes the impact of these two

Table 7 Performance gains for $n = 10/30/50$ customers for both sequencing and both waiter scheduling approaches in relation to the status quo in percent

		Customer sequencing	
		Random	Optimized
Waiter	Order-by-order	Status quo	IGG
scheduling		0/0/0	$-0.48/0.08/-0.07$
	Order-swapping	RANDOM + BS	PRIO + BS
		45.18/50.5/51.73	51.18/56.99/57.98

levers on the system performance. Specifically, we report the relative decrease of makespan compared to the makespan obtained by the status quo solution in percent for $n = 10$, $n = 30$, and $n = 50$ customers. The following three approaches are benchmarked with the status quo:

- Our iterative Gilmore–Gomory procedure (IGG, see Sect. 3.1) applied to a random customer sequence represents the case where only the customer sequence is optimized, but the waiter still processes the customers order by order.
- The case where customer sequences are not optimized (e.g., they are processed according to FCFS), but the waiter follows an optimized order-swapping schedule is represented by approach 'RANDOM + BS'.
- Finally, both customer sequences and the waiter schedule under the order-swapping policy can be optimized. This approach is denoted 'PRIO + BS'.

The results of these four solution approaches averaged over all our 270 data instances elaborated in Sect. 4.1 are summarized in Table 7. The following conclusion can be drawn from these results:

- Only optimizing the customer sequence, if the waiter still follows the order-by-order policy cannot improve system performance. Recall that the IGG presented in Sect. 3.1 minimizes the walking distance of the waiter if customers are strictly serviced one after another. However, this approach neglects additional waiting time of the waiter until the customers have arrived at the respective counters. Obviously, this simplification deteriorates the objective values to such an extent, that IGG cannot even outperform random customer sequences. In fact, for $n = 10$ and $n = 50$ customers the average makespan delivered by IGG is even larger than that of a random sequence.
- A similar result with regard to the impact of customer sequences can be concluded if the waiter operates under the order-swapping policy. In this case, too, the performance gains of optimized customer sequences are not overly large. However, at least about 6% additional performance can be gained by customer sequences optimized with our priority rule-based approach.
- The largest performance gains, however, are promised by allowing the waiter to flexibly swap among orders. An optimized waiter scheduling under the order-swapping policy almost halves the makespan compared to the status quo where the waiter processes customer after customer according to the order-by-order policy.

It can be concluded that among our two levers, especially, the order-swapping policy promises a large improvement. It about halves the makespan compared to the order-by-order policy, whereas optimizing the customer sequences only promises just 6% additional reduction.

5.2 Impact on picker walking

In the distribution center we visited, another concern of the managers was the excessive walking of the pickers during their shifts. Their calculations have shown that regularly a dozen kilometers per shift were exceeded. Consequently, this section is dedicated to the question whether optimizing the waiter schedule under the order-swapping policy not only boosts the picking performance, but also considerably reduces the picker's total walking distances. To explore this question, we setup the following experiment.

To emulate a realistic on-the-line picking environment, we apply the following data. The picker is assumed to walk $v^{\text{wait}} = 1.36\,\text{m/s}$ (about 5 km/h). This is a typical moderate walking speed that is often agreed with the trade unions as an average target speed (Füßler and Boysen 2017). The AGVs are assumed to have the same speed $v^{\text{cust}} = 1.36$ m/s. Due to safety reasons, many AGVs are restricted to walking speed if they interfere with human pickers. The counters (storage positions for crates of beverages) are assumed to be 1 m wide, which is enough space to store a standardized euro-pallet (i.e., 80 cm) with maneuvering space to the left and right. We presuppose $m = 50$ counters (storage positions), which is about the size we observed in our distribution center. Assuming an average processing time (picking duration) of 15 s/dish (order line), preliminary tests have shown that about 100 (27) customers can be served within an hour, if the number of order lines per order is drawn from interval $[\underline{\gamma}; \overline{\gamma}] =$ [1;3] ($[\underline{\gamma}; \overline{\gamma}] =$ [7;10]). To emulate different shift lengths, we therefore apply $n = 100$ ($n = 27$), $n = 200$ ($n = 54$), and $n = 800$ ($n = 216$) customers for shifts of lengths of 1, 2, and 8 h, respectively, if we have just a few (many) order lines per customer with $[\underline{\gamma}; \overline{\gamma}] = [1; 3]$ ($[\underline{\gamma}; \overline{\gamma}] = [7; 10]$).

In this environment, we benchmark the following competitors:

- IGG: We apply the iterative Gilmore–Gomory approach (IGG, see Sect. 3.1) for optimizing the customer sequence, while the waiter follows the order-by-order policy.
- PRIO + BS − M: This approach optimizes the customer sequence according to our priority rule-based approach (see Sect. 3.2) and optimizes the waiter schedule under the order-swapping policy with the help of our beam search procedure (BS, see Sect. 2.4) applied with beam width $\zeta = 300$. The aim of this approach is to minimize the makespan.
- PRIO + BS − W: To explore to what extent the waiter's total walking distance can be reduced, we adapt our dynamic programming scheme for optimizing the waiter schedule under the order-swapping policy to the walking distance objective (see" Appendix C"). The resulting beam search approach minimizes the total walking distance of the waiter for a given customer sequence, which is again derived by applying our priority rule-based approach of Sect. 3.2.

Table 8 Reduction of the waiter's total walking distance (W) and makespan (M) of our three optimization approaches IGG, PRIO + BS − M, and PRIO + BS − W in relation to the status quo approach in % for different shift lengths and varying numbers of order lines

$[\underline{\gamma}; \overline{\gamma}]$	Shift (h)	IGG		PRIO + BS − M		PRIO + BS − W	
		W	M	W	M	W	M
[1; 3]	1	15.01	7.41	84.12	46.16	84.48	45.99
[1; 3]	2	15.05	7.61	84.24	46.02	84.36	45.83
[1; 3]	8	14.72	7.71	84.03	45.95	84.39	45.8
[7; 10]	1	0.48	0.16	76.98	31.81	77.93	31.7
[7; 10]	2	0.34	0.11	77.11	32.27	77.87	32.11
[7; 10]	8	0.09	0.03	76.6	32.24	77.12	32.04

- Finally, we have the status quo where random order sequences are processed under the order-by-order policy. We report the improvement over this policy for the previous three approaches (i.e., the reduction of the respective approach in relation to either the makespan or the total walking distance of the status quo policy in %) in Table 8.

The results of this test summarized in Table 8 suggest the following findings:

- Only optimizing the customer sequence by IGG while still applying the order-by-order policy improves over the status quo only if each customer demands just a few order lines. In this case, the total walking distance is reduced by 15% and also a better makespan can be achieved. If, however, each customer demands plenty order lines, then there is a high probability that the picker has to pass the whole line anyway when processing each order according to the order-by-order policy. Optimizing the customer sequence has not much flexibility in this case, and only negligible improvements over the status quo can be realized.
- In line with our previous results, optimizing customer sequences and waiter schedules under the order-swapping policy with our PRIO + BS − M leads to a considerable reduction of the makespan compared to the status quo. Note that the relative reductions here are slightly smaller than our previous 50%, because in this test a processing time of 15 s/dish is included. The reductions of the waiter's total walking distance is even more impressive. Even for plenty order lines the reduction is by more than 75%.
- If we directly minimize the total walking distance of the waiter by applying PRIO + BS − W, we reach almost the same results as for PRIO + BS − M. As expected, the total walking distance is even smaller, but just to a tiny extent well below 1%. With regard to the makespan, PRIO + BS − W is slightly outperformed by PRIO + BS − M, but, again, just to a very small extent. Since computational tests (we do not report here) have shown that the solution performance of both approaches is almost identical, we can conclude that there is a strong correlation among both objectives. Minimizing the makespan also tends to minimize the waiter's walking distance and vice versa.

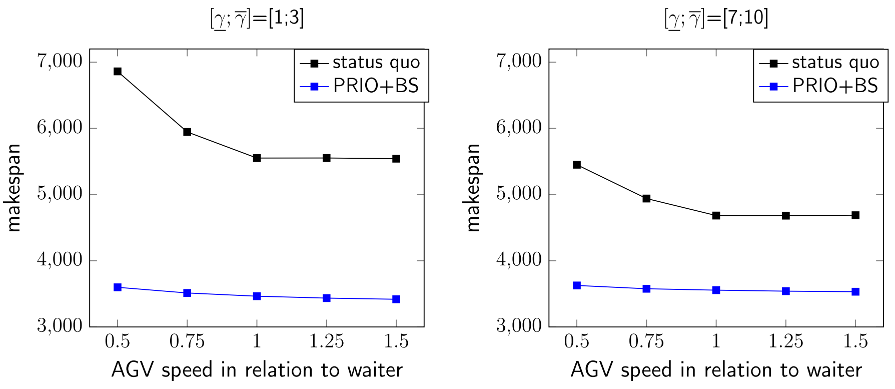


Fig. 5 Impact of AGV speed on the makespan for the status quo approach and optimization procedure PRIO + BS

It can be concluded that optimization also promises a great relief for the waiter. Our average result for an 8-h shift suggests a total walking distance of about 17 km for the current process in our example distribution center. By optimization, this can be reduced to an average walking distance of just about 4 km/shift.

5.3 Impact of faster AGVs

Instead of implementing the order-swapping policy steered by an optimization procedure, a distribution center seeking better system performance could also consider investing into a technical solution. Faster AGVs, for instance, also promise a reduction of the makespan. This section explores whether the gains of faster AGVs can compete with the huge improvement promised by optimization. We apply the same setting as is elaborated in the previous section, and only vary the velocity of the AGVs. Specifically, we vary v^{cust} by applying values $(0.5; 0.75; 1; 1.25; 1.5) \cdot v^{\text{wait}}$, with $v^{\text{wait}} = 1.36$ m/s. Furthermore, we distinguish between a large and a small number of dishes per customer $[\underline{\gamma}; \overline{\gamma}]$, assume $m = 50$ counters and $n = 27$, or $n = 100$ customers for 1-h shifts. Figure 5 shows the results for the status quo (i.e., random customer sequences processed under the order-by-order policy) and our beam search approach PRIO + BS, which optimizes customer sequences and waiter schedules according to the order-swapping policy.

It can be concluded that investing additional budget into faster AGVs is only a worthwhile idea if operating under the order-by-order policy and only until the velocity equals the speed the waiter. However, the money seems much better spent into an optimization procedure enabling the order-swapping policy. This approach promises a much lower makespan, which cannot be considerably improved by faster AGVs.

6 Conclusion

This paper explores the cafeteria process: A given set of customers demanding deterministic subsets of dishes served at the counters of a cafeteria are processed by a single waiter. We show that optimizing the customer sequence, in which the customers enter the cafeteria, and the waiter schedule, in which the waiter serves the dish requests of queuing customers, considerably improves the performance compared to a non-optimized system. We introduce suited solution procedures for both problem parts, i.e., customer sequencing and waiter scheduling, and test them in a comprehensive computational study. The main findings of these tests are the following:

- Large performance gains are obtainable by optimizing customer sequences and waiter schedules compared to non-optimized solutions. The makespan is almost halved. Especially, allowing the waiter a flexible swapping among customers greatly improves the results. This is good news for managers of a cafeteria, but also for the waiter. Our results show that the makespan is closely related to the waiter's walking distance. Thus, by minimizing the makespan also the waiter's ergonomic strain while walking between the counters can be considerably reduced.
- When having to improve the performance of a cafeteria process, money is much better spent for optimization procedures determining suited customer sequences and waiter schedules compared to investing into faster AGVs. Their impact is shown to quickly diminish.

These findings may help to improve order picking in the real-world distribution center supplying liquor stores, supermarkets, and large restaurants with beverages where we saw the cafeteria process in action.

Future research could challenge our solution procedures. For instances of real-world size determining good waiter schedules takes considerable time, so that the evaluation of many different customer sequences is even more time-consuming. Thus, solving very large instance of the cafeteria problem remains a challenging task. Furthermore, future research could also support the layout design phase. By altering the assignment of dishes to counters even larger performance gains may be obtainable.

Appendix A: Complexity of CWSP

This appendix shows that our CWSP, i.e., the scheduling of the waiter once the customer sequence is given, is strongly NP-hard. The reduction is from the 3-Partition problem, which is well known to be NP-complete in the strong sense (Garey and Johnson 1979).

3-Partition Given an integer $B \in \mathbb{Z}^+$ and a set A (with $|A| = 3q$) containing integers $B/4 < a_j < B/2, \forall j = 1, \ldots, 3q$. The problem is to find a partition of set A into q disjoint subsets $\{A_1, A_2, \ldots, A_q\}$, such that $\sum_{j \in A_i} a_j = B$ for each $i = 1, \ldots, q$ or to prove that no such partition exists.

Proof Our pseudo-polynomial transformation scheme for deriving an instance I' of CWSP from an instance I of 3-Partition is as follows: We introduce $n = 3q + 1$ customers. For each integer value of I, we introduce one customer demanding just a single dish. These customers build the counterparts to the integer values and are, thus, called the *counterpart customers*. The single additional customer interrupts the waiter again and again when serving the counterpart customers, so that we call her the *stop customer*. All customers move with a velocity of $v^{\text{cust}} = \frac{12q}{B}$ along the

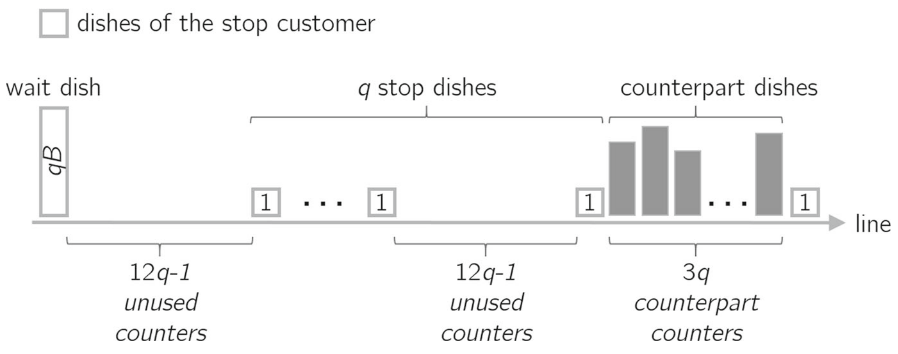


Fig. 6 Schematic cafeteria layout of transformation

counters, which all have a normalized length of one distance unit. The line consists of $m = 12q^2 + 3q + 2$ counters, and our single waiter moves with infinite velocity. Note that it is easily possible to give the waiter a finite velocity in this proof, which, however, makes the calculations a bit more complicated. The first $3q$ customers, which enter the line according to the given customer sequence, are the counterpart customers. They each demand a single dish served at the end of the line. i.e., at the last $3q$ counters before the final one. The first counterpart customer, thus, demands the dish served at the second to last counter, the second customer the third seen from the end of the line and so on. Servicing each dish demanded by a counterpart customer requires the waiter a processing time that equals the corresponding integer value, so that we have $\tau_{j,1} = a_j$ for all $j = 1, \ldots, 3q$. The stop customer is the last to enter the line. She demands $q+2$ dishes. Her first dish is served at the very first counter and lasts until all counterpart customers have passed the line and reached their successive counters at the end of the line. We call this dish request the *wait dish*, which takes the waiter a processing time of $\tau_{3q+1,1} = qB$. Her next q dishes are the so-called *stop dishes*, which, starting at counter $12q+1$, are served at counters having $12q-1$ unused counters between them. The processing times at these *stop counters* are $\tau_{3q+1,i} = 1$ for all $i = 2, \ldots, q+1$, and it takes the stop customer exactly B time units to move from stop counter to stop counter. Finally, the stop customer is serviced at the very last counter with what we call the *final dish*. Servicing the final dish takes the waiter a processing time of one time unit. Figure 6 schematically depicts the setup of the resulting cafeteria. The question we ask is whether there is a solution to I' with makespan $Z \leq \frac{B}{2} + \frac{B}{6q} + 2qB + q + 1$.

A feasible solution for an instance I of 3-Partition can be transformed to a feasible solution of the corresponding instance I' of CWSP by pursuing the following waiter schedule. At first, the waiter remains idle for $\frac{B}{4} + \frac{B}{12q}$ time units until all counterpart customers have passed the first counter and the wait dish of the stop customer can be serviced. The latter requires qB time units. Once processing the wait dish ends, all counterpart customers have exactly reached their $3q$ counters of the end of the line where they aim to receive their respective dish. Afterward, the stop customer moves onwards and passes $12q-1$ counters toward her first stop dish. At a velocity of $v^{\text{cust}} = \frac{12q}{B}$, the movement from wait dish to the first stop counter (and later on between two successive stop counters) takes her exactly B time units. During these B

time units, the waiter is not occupied by the stop customer, but can rush (with infinite velocity) to the counterpart customers, where she can serve exactly the three dishes of the counterpart customers corresponding to an integer subset of 3-Partition. In this way, the waiter successively services all subsets of dishes corresponding to subsets of 3-Partition each time interrupted by one time unit servicing the stop customer once having reached the next stop dish after exactly B time units. Once the stop customer has received her last stop dish, it takes her $\frac{B}{4} + \frac{B}{12q}$ time units to pass the last $3q$ counters and another single time unit to be served the final dish at the last counter. Thus, we have $\frac{B}{4} + \frac{B}{12q}$ time units at the start until the wait dish can be serviced. This takes qB time units to be processed, and then, another $qB + q$ time units are required to processes all counterpart customers (i.e., serviced in q subsets of duration B) interrupted by q stop dishes of the stop customer. Finally, it takes the stop customer another $\frac{B}{4} + \frac{B}{12q}$ time units to pass the last $3q$ counters and an additional time unit at the last counter. Altogether, this leads to a makespan of $Z = \frac{B}{2} + \frac{B}{6q} + 2qB + q + 1$ and a feasible schedule for I'.

On the other hand, a feasible solution for an instance I' of CWSP is also a feasible solution for the corresponding 3-Partition instance I. A makespan of $Z \le \frac{B}{2} + \frac{B}{6q} + 2qB + q + 1$ for I' can only be reached if the stop customer reaches the end of the line without any waiting at a counter. Thus, once the wait dish at the start is serviced, this leaves exactly B time units for servicing counterpart customers, before the stop customer reaches another stop counter. If these gaps of B time units are not exactly filled with three counterpart customers whose processing time of their respective dishes adds up to exactly B time units, then there remains at least one dish for a counterpart customer to be serviced after the stop customer has left her last stop counter toward the end of the line. Even if there is just one remaining dish and this final dish is the one with the shortest processing time among all counterpart customers and serviced at the second to last counter, then due to the restrictions of the integer values of 3-Partition, i.e., $B/4 < a_j < B/2$, this dish cannot be completed before the stop customer reaches this job (because the way from the last stop counter to the second last counterpart counter takes her just $\frac{B}{4} - \frac{B}{12q}$ time units). She is blocked, and a makespan $Z \le \frac{B}{2} + \frac{B}{6q} + 2qB + q + 1$ cannot be realized. Only if all counterpart customers are already serviced, once the stop customer leaves the final stop counter, no blocking occurs. This, however, is only possible if sets of three dishes dedicated to three counterpart customers are serviced in each of the q gaps of duration B. These three dishes directly correspond to a feasible subset of 3-Partition, which completes the proof. □

Note that the same proof also holds for the case where the sequencing of customers is part of the problem. No other sequence than the one applied in the previous proof can reach the applied bound on the makespan. All other sequences lead to cases where customers block each other and the makespan bound is exceeded.

Appendix B: A multi-start simulated annealing algorithm

In this appendix, we introduce a straightforward multi-start simulated annealing approach for benchmarking our other (even more) basic approaches to determine customer sequences. We follow the basic simulated annealing scheme introduced by Kirkpatrick et al. (1983). First, we apply the priority rule-based approach (see Sect. 3.2) to find an initial customer sequence, which is evaluated under the order-swapping policy with beam search (see Sect. 2.4). Neighborhood solutions are determined by swapping the positions of two randomly selected customers within current customer sequence ϵ. This leads to a new solution ϵ_{new} whose objective value $F(\epsilon_{\text{new}})$ is determined by applying the beam search procedure for waiter scheduling. This new solution ϵ_{new} is accepted if a random number of interval [0,1] with uniform distribution is smaller than $\exp\frac{F(\epsilon_{\text{old}})-F(\epsilon_{\text{new}})}{\text{TP}}$ and refused otherwise. Temperature TP is controlled via the traditional cooling scheme of Kirkpatrick et al. (1983) with an initial temperature of 50, a stop temperature of 1, and a cooling factor of 0.99, which is multiplied with the current temperature in each iteration. When reaching the stop temperature, we restart the procedure with the best customer sequence found so far. If no improved solution is found in the next iteration, we draw a new customer sequence randomly and restart the procedure. The procedure ends after a total runtime of 2 h (7200 CPU seconds).

Appendix C: A dynamic program for CWSP minimizing the waiter's total walking distance

In this appendix, we derive a DP scheme that can be applied to our CWSP when minimizing the waiter's walking distance. This approach is a simple modification of the DP approach described in Sect. 2.3 which minimizes the makespan for a given customer sequence.

The DP consists of $|T| + 1$ stages, each representing a service period. As we minimize the walking distance of the waiter, we can neglect the exact positions of the customers during the process. Note that customers are assumed to always be at the respective counter currently demanded, because potential waiting times of the waiter are not relevant. This leaves us with states $Z = (J, w)$ defined by set J of finished dish requests and current waiter location w. The partial objective value of a state $f(Z)$ or $f(J, w)$ represents the total walking distance of the waiter after completing all dish requests in J. We have a transition from state $Z = (J, w)$ to another state $Z' = (J', w')$, if $\exists\,(j,l) \in \Phi$ with $\Lambda_{j,l} \setminus J = \emptyset$, $J' \setminus J = \{(j,l)\}$, and $w' = d_{j,l}$. The additional walking distance of such a transition amounts to

$$\Delta(Z, Z') = |w' - w|. \tag{31}$$

Starting from initial state $Z_0 = (\emptyset, 0)$ with $f(Z_0) = 0$, we can determine the partial objective value for a new state with the basic Bellman recursion

$$f(Z') = f(J', w') = \min_{\substack{Z=(J,w):\\ |J'\setminus J|=1}} \{f(Z) + \Delta(Z, Z')\}. \tag{32}$$

After a forward recursion through all stages, we compare all final states in order to determine the optimal solution value

$$F = \min_{\substack{Z=(J,w):\\ J=\Phi^0}} \{f(Z)\}. \tag{33}$$

Finally, an optimal order sequence can be extracted by a backward recursion. Analogously to the previous dynamic program for our CWSP when minimizing the makespan (see Sect. 2.3), we derive a heuristic beam search procedure from the DP scheme by applying the alterations defined in Sect. 2.4.

References

Agnetis A, Flamini M, Nicosia G, Pacifici A (2011) A job-shop problem with one additional resource type. J Sched 14:225–237

Agnetis A, Murgia G, Sbrilli S (2014) A job shop scheduling problem with human operators in handicraft production. Int J Prod Res 52:3820–3831

Ascheuer N, Escudero LF, Grötschel M, Stoer M (1993) A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing). SIAM J Optim 3:25–42

Boysen N, Briskorn D, Meisel F (2017) A generalized classification scheme for crane scheduling with interference. Eur J Oper Res 258:343–357

Bierwirth C, Meisel F (2010) A survey of berth allocation and quay crane scheduling problems in container terminals. Eur J Oper Res 202:615–627

Bierwirth C, Meisel F (2015) A follow-up survey of berth allocation and quay crane scheduling problems in container terminals. Eur J Oper Res 244:675–689

Brucker P, Dhaenens-Flipo C, Knust S, Kravchenko SA, Werner F (2002) Complexity results for parallel machine problems with a single server. J Sched 5:429–457

Brucker P, Knust S, Wang G (2005) Complexity results for flow-shop problems with a single server. Eur J Oper Res 165:398–407

Burkard RE, Deineko VG, van Dal R, van der Veen JA, Woeginger GJ (1998) Well-solvable special cases of the traveling salesman problem: a survey. SIAM Rev 40:496–546

Ciro CG, Dugardin F, Yalaoui F, Kelly R (2016) Open shop scheduling problem with a multi-skills resource constraint: a genetic algorithm and an ant colony optimisation approach. Int J Prod Res 54:4854–4881

Chen F, Wang H, Qi C, Xie Y (2013) An ant colony optimization routing algorithm for two order pickers with congestion consideration. Comput Ind Eng 66:77–85

Cheng TCE, Wang G, Sriskandarajah C (1999) One-operator-two-machine flowshop scheduling with setup and dismounting times. Comput Oper Res 26:715–730

Daganzo CF (1989) The crane scheduling problem. Transp Res B 23:159–175

Füßler D, Boysen N (2017) Efficient order processing in an inverse order picking system. Comput Oper Res 88:150–160

Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, New York

Garey MR, Johnson DS, Sethi R (1976) The complexity of flowshop and jobshop scheduling. Math Oper Res 1:117–129

Gilmore PC, Gomory RE (1964) Sequencing a one state-variable machine: a solvable case of the traveling salesman problem. Oper Res 12:655–679

Glass CA, Shafransky YM, Strusevich VA (2000) Scheduling for parallel dedicated machines with a single server. Naval Res Logist 47:304–328

Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discrete Math 5:287–326

Gue KR, Meller RD, Skufca JD (2006) The effects of pick density on order picking areas with narrow aisles. IIE Trans 38:859–868

Hall N, Potts C, Sriskandarajah C (2000) Parallel machine scheduling with a common server. Discrete Appl Math 102:223–243

Hefetz N, Adiri I (1982) A note on the influence of missing operations on scheduling problems. Naval Res Logist 29:535–539

Held M, Karp RM (1970) The traveling-salesman problem and minimum spanning trees. Oper Res 18:1138–1162

Hong S, Johnson AL, Peters BA (2012) Batch picking in narrow-aisle order picking systems with consideration for picker blocking. Eur J Oper Res 221:557–570

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680

Liu B, Wang L, Jin YH (2008) An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. Comput Oper Res 35:2791–2806

Mencia C, Sierra MR, Varela R (2013) An efficient hybrid search algorithm for job shop scheduling with operators. Int J Prod Res 51:5221–5237

Ribas I, Companys R, Tort-Martorell X (2011) An iterated greedy algorithm for the flowshop scheduling problem with blocking. Omega 39:293–301

Ruiz RM, Maroto C (2005) A comprehensive review and evaluation of permutation flowshop heuristics. Eur J Oper Res 165:479–494

Sawik TJ (1995) Scheduling flexible flow lines with no in-process buffers. Int J Prod Res 33:1357–1367

Swisslog (2017) CaddyPick: picking made easy. https://www.swisslog.com/de-de/kontakt/downloads?mediaItem=5551B202AE7541FA9662BDC8A95B7041. Accessed Dec 2018

Tang L, Luh PB, Liu J, Fang L (2002) Steel-making process scheduling using Lagrangian relaxation. Int J Prod Res 40:55–70

Wang G, Cheng TCE (2001) An approximation algorithm for parallel machine scheduling with a common server. J Oper Res Soc 52:234–237

Weber RR, Weiss G (1994) The cafeteria process—Tandem queues with 0–1 dependent service times and the bowl shape phenomenon. Oper Res 42:895–912

Xuan H, Tang L (2007) Scheduling a hybrid flowshop with batch production at the last stage. Comput Oper Res 34:2718–2733