Proceedings of the 2023 Winter Simulation Conference
C. G. Corlu, S. R. Hunter, H. Lam, B. S. Onggo, J. Shortle, and B. Biller, eds.

CROSSSTACKS: A DATASET AND A SIMULATIVE STUDY OF STORAGE ALLOCATION STRATEGIES FOR CROSS-DOCKING BLOCK-STACKING WAREHOUSES

Alexandru Rinciog
Jakob Pfrommer
Hardik Rathod
Anne Meyer

Department of Mechanical Engineering
TU Dortmund University
Leonhard-Euler-Straße 5
Dortmund, 44227, GERMANY

Natalia Ogorelysheva
Anna Vasileva

Fraunhofer-Institut for
Material Flow and Logistics
Joseph-von-Fraunhofer-Straße 2-4
Dortmund, 44227, GERMANY

ABSTRACT

Cross-docking is a warehousing strategy that (ideally) moves goods from inbound docks directly to outbound docks. In reality, goods often need to be temporarily stored. Cross-docking is typically set up as a block-stacking warehouse (BSW), where goods are stored directly on the ground. Autonomous mobile robots (AMRs) could significantly reduce BSW costs. To deploy AMR systems to BSWs, five interlaced decision problems, including the storage location assignment problem (SLAP), need to be solved. Because of the combinatorial complexity of BSWs, and the absence of pertinent use case data and fitting simulation software, this is a challenging task. This work seeks to alleviate these gaps by (1) extending SLAPStack, a fine-grained open-source BSW simulation framework to accommodate cross-docking, (2) providing CROSSStacks, a real-world cross-docking dataset, and (3) evaluating two dual command cycle SLAP strategies as of yet untested for BSWs. One of the approaches outperforms a naive cross-docking SLAP strategy.

1 INTRODUCTION

Cross-docking terminals are an effective solution for reducing warehouse handling costs and improving supply chain efficiency. This warehousing solution ideally moves goods from trucks at inbound docks directly to trucks waiting at outbound docks. While this is not always possible (Van Belle et al. 2012), the goods' duration of stay in the warehouse is short-lived.

Cross-docking terminals that handle pallets are usually set up as a block stacking warehouse (BSW). Figure 1 shows a BSW, where pallets are stored on the floor and stacked on top of each other. Pallets are typically associated with additional information, such as a stock-keeping unit (SKU). The layout of a BSW is typically grid-based, divided into storage bays and pathways (aisles), with each storage bay consisting of several lanes made up of a sequence of stacks (Pfrommer et al. 2022). As interim nodes in the supply chain, cross-docking terminals are subject to relocation as dictated by changes in the network. Being infrastructure-free solutions, BSWs are fast and comparatively easy to set up and operate.

However, realizing autonomous BSWs is a challenging task. BSWs require solving five interdependent decision problems serving a common optimization goal, e.g., service time or total travel distance (Pfrommer and Meyer 2020). The solution to the Layout Design Problem (LDP) yields the warehouse layout, including the dock positions (1). The Vehicle Dispatching Problem (VDP) defines the order in which vehicles perform transport tasks and the associated routes (2). The Storage Location Assignment Problem (SLAP) maps inbound items to storage locations (3). The Unit-Load Selection Problem (ULSP) defines which items are selected from storage given outbound orders (4). Last but not least, the Unit Load Relocation Problem (ULRP) concerns itself with rearranging pallets within the warehouse for faster access (5). Together with

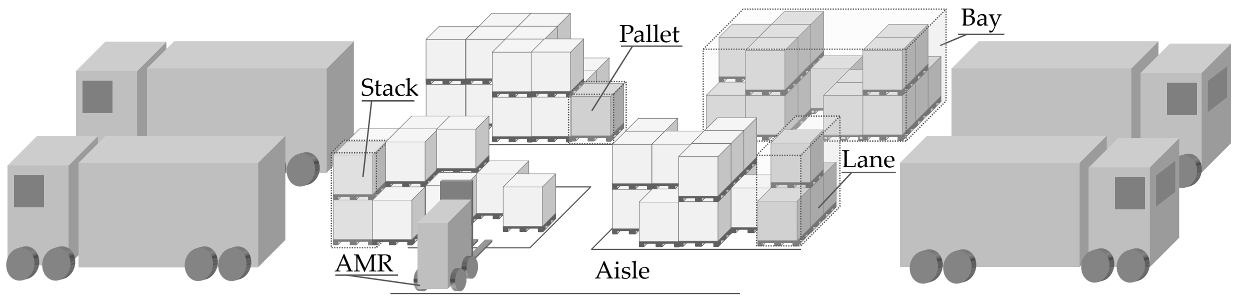


Figure 1: Example of a cross-docking terminal based on a block stacking warehouse.

autonomous vehicles, these problems define the Autonomous BSW Problem (Pfrommer and Meyer 2020). Because of the large number of decision variables coupled with a need for quick resolution within an environment prone to uncertainty, warehouse operators generally resort to heuristics for solving constituent problems, e.g., class-based SLAP strategies (Le-Duc and De Koster 2005).

In cross-docking terminals, the outbound docks of SKUs are known on arrival. Additionally, in some use cases, SKUs are unique (e.g., parcel logistics), leading to the five BSW sub-problems being typically tackled as follows. In terms of LDP, the storage bay lanes are designed to be shallow for quick access, and input docks are placed on the opposite side of the warehouse relative to outbound docks. Outbound docks are assigned to specific destinations on a daily basis. Forklift drivers make VDP decisions. The ULSP solution is trivial since pallets have unique SKUs. If tackled at all, the ULRP is usually coordinated by the shift supervisor or handled locally by forklift drivers. Regarding SLAP, if corresponding outbound trucks have not yet arrived, the typical SLAP strategy is to place pallets into zones near their destination.

As noted by Pfrommer et al. (2022), BSWs in general reveal a gap in research, particularly in the matter of SLAP, simulation frameworks, and evaluation data, with cross-docking being no exception. SLAP is mostly investigated for warehousing solutions other than BSWs in general and cross-docking in particular. Given that ULSP is trivial for the current use case, and accepting the aforementioned authors' argument that a holistic BSW solution needs to at least tackle SLAP, VRP, and ULSP, along with the conjecture that the greedy VRP strategy of picking the closest available vehicle to execute a transport task is acceptable, leaves us with SLAP to focus on. Furthermore, neither a simulation framework nor appropriate use-case data needed to evaluate algorithms are available for the realm of cross-docking BSWs.

This works endeavors to alleviate these gaps by making the following three contributions to the domain of intralogistics optimization while focusing on Autonomous Moving Robot (AMR) systems: (I) *the fine-grained BSW Simulation framework dubbed SLAPStack is extended* (Pfrommer et al. 2022) (Section 3) to account for cross-docking particularities, optimizing the contained routing scheme, and implementing dock-to-dock transit. Additionally, functionality enabling queueing multiple orders to the same AMR is implemented. (II) *An anonymized real-world benchmarking dataset is provided*. The dataset, which consists of a warehouse layout and an order stream, was created in collaboration with a freight company and an indoor localization company (Section 4). (III) Shortest Leg (SL) and Closest to the Next Delivery (CTNR), which are *two SLAP strategies typical for high-bay and picking warehouses are transferred to and evaluated on BSWs* using SLAPStack. SL and CTNR are baselined against the typical cross-docking strategy of picking the location Closest to the pallet's Destination (CTD), random location picking (RND), and the greedy strategy of picking the Closest Open Location (COL). The experiments show SL to be the best strategy for the use case at hand.

2 RELATED WORK

Closely related literature is sparse. The cross-docking literature mainly focuses on problems other than SLAP. Conversely, investigations of SLAP strategies are applied to setups other than cross-docking. Given the particularities of the investigated setup, the bulk of the schemes present in the literature are not applicable. Dual command cycle heuristics are present in a lot of publications (e.g., Pan and Wang 1996; Bortolini et al.

2020), but their application domain is only distantly related with the present one. Warehouse simulations are often either too task-specific or, again, too far removed from the current application.

Cross-Docking. Within the realm of cross-docking, SLAP is frequently neglected seen as, the storage of items is explicitly avoided. Vis and Roodbergen (2008) use a network flow model to minimize travel distances. Similarly, Werners and Wülfing (2010) use a linear programming model to optimize the traveled distance at a parcel sorting center, while simultaneously assigning docks to incoming trucks. A shortcoming of such exact approaches is the assumption of truck arrival times and loads being known in advance. In CROSSStacks this is not the case. Given the time required for exact approaches to find a solution, solving SLAP on demand using such methods is impractical.

Storage Location Assignment Problem. A review of SLAP policies by Gu et al. (2007) classifies policies using the availability of product information as a distinguishing factor, yielding three categories: either detailed item information (II) is available, coarser product information (PI) can be used to prioritize incoming orders or no information (NI) pertaining to items or products is available. The literature mostly focuses on the second category. The current work presents strategies in the third category (SL, CTNR) and baselines them against two standard NI-strategies (COL, RND) and one II-strategy (CTD). Other NI strategies are Furthest Open Location and Longest Open Location (Gu et al. 2007). Another publication falling in the II category is authored by Goetschalckx and Ratliff (1990), who use item due dates for their location prioritization scheme. In this cross-docking case, reliable information on due dates is not available. Most of the SLAP literature falls into the PI class. Since product information is unavailable for the use case at hand, PI strategies are only briefly mentioned here. Here, either several products are grouped into classes that are assigned fixed zones in the warehouse, or the zone assignment occurs on a product level rather than a product group level (dedicated storage). Schwartz et al. deliver the oldest account of class-based storage policies, positioning high-turnover products closer to docks (favorable locations) and vice-versa (Schwarz et al. 1978). Frazelle correlates favorable locations with high popularity, low stock, or a low cube-per-order index (ratio of an item's size to its popularity) and compares the results (Frazelle 2016). Heskett also uses the cube-per-order index but for dedicated location assignment, rather than defining classes (Heskett 1964). Hausman et al. compare a turnover-based class assignment scheme with a turnover-based dedicated storage assignment strategy. The latter outperforms the former (Hausman et al. 1976).

Simulation Frameworks. In terms of simulations, most openly available works are too task-specific or too far from the domain of BSWs to warrant their use for a holistic BSW decision investigation. The simulation used to demonstrate forklift motion planning (Mahajan et al. 2005), for instance, only focuses on simple trajectory planning. Another example is a simulation used to analyze the lane layout and depth of bulk storage warehouses (Clements et al. 2016). An example of a simulation that is broad enough to study decisions in a warehouse environment holistically, is RAWSim-O (Merschformann et al. 2017). However, RAWSim-O is tailored to a very different warehousing solution, namely a robotic mobile fulfillment system. Herein, items are stored on movable shelves and carried to pick stations by robots.

The most closely related work was put forward by Pfrommer et al. (2022). The authors use a fine-grained open-source simulation framework named SLAPStack to investigate a minimally viable autonomously operated BSW combining multiple decision problems (SLAP, ULSP, and VDP) in a real-world large-scale finished goods use case of a hygienic paper producer which they make public. The authors demonstrate that a variant of COL, namely closest open pure lane, outperforms 15 further class-based heuristics constructed using different popularity metrics and number of classes. Their experiments additionally reveal the optimization potential inherent to sorting the warehouse.

3 SLAPSTACK FRAMEWORK AND EXTENSIONS

SLAPStack follows the OpenAI Gym API and has a modular design. Therefore, it can easily be extended to accommodate the use case at hand. After a brief framework introduction in Section 3.1, the modifications made to SLAPStack are described in Section 3.2.

3.1 SLAPStack at a Glance

To make it possible to discern the implemented extensions and performed experiments, the simulation framework's architecture, its core logic as reflected by SLAPStack's event sequence, and its routing functionality are first summarily described. For a more in-depth account, see Pfrommer et. al (2022).

Framework Components. SLAPStack is constructed following the architecture proposed by Rinciog et al. (2021). The entry point is located in the SlapEnv class which implements the OpenAI Gym interface (i.e., the step, reset methods along with the state_space and action_space properties). SlapEnv is a wrapper around SlapCore which implements the simulation logic by means of the step function. The core uses an EventManager object to maintain a heap of time sorted future_events, along with two structures tracking currently pending orders, namely queued_delivery_orders and queued_retrieval_orders. The central SLAPStack information structure is the State property of SlapCore. The State contains several manager objects dedicated to distinct tasks. For instance, the RouteManager is used to compute routes on the storage matrix S grid during Transport event initialization. The AmrManager monitors and handles changes in AMR location and state. The LocationManager keeps track of open and occupied positions in the warehouse. SLAPStack implements several key performance indicators (KPI)s which are maintained by the Trckers property of the state. Additionally, SLAPStack offers an interface defining and documenting the simulation inputs along with the SlapLogger among other objects.

Event Chains. The future_events heap is at the heart of the simulation. Contained events are self-handling, meaning that the state update logic is implemented within their handle function. Currently, Events are either of type Orders or of type Transport. Orders – Delivery/Retrieval – are constructed on initialization from the simulation input during initialization. Transport events – DeliveryFirst/SecondLeg, RetrievalFirst/SecondLeg – are instantiated during the simulation execution as orders are serviced. Retrieval and DeliverySecondLeg, are "blocking" and require external decisions. Specifically, the Retrieval event requires a ULSP decision, while the DeliverySecondLeg event requires a SLAP decision.

During simulation steps, events are popped from the heap, and their handle function is called. Since order events can only be handled if an AMR is free, the pending order structures are used to buffer them until the next AMR release. Buffered orders are handled in a FIFO fashion. When encountering a blocking event, the execution halts, and the state is returned. The execution resumes as soon as SLAPStack is provided with an action by an external control algorithm. In the case of SLAP, this action corresponds to a free storage position. In the ULSP case, it corresponds to the position of a pallet with an SKU corresponding to the order SKU. This loop continues until all environment orders are executed.

Travel events are generated on handling of Order events as shown in 2. The delivery order event chain starts the occurrence of a Delivery event ❶. At this point, the simulation retrieves the closest AMR and assigns it to the delivery task by adding a DeliveryFirstLeg event, which links AMR and Order, to the heap ❷. When this event triggers, the simulation requests a SLAP decision from a control algorithm ❸, leading to the creation of a DeliverySecondLeg event. Finally, when this second travel event triggers ❹, the AMR is released ❺. The retrieval order event chain represents the complementary process: The chain starts with the arrival of a Retrieval order ①, whereupon the control algorithm takes a ULSP decision. The AMR closest to the chosen pallet position is marked as busy and a RetrievalFirstLeg event is added to the event heap ②. When the associated travel concludes ③, a RetrievalSecondLeg is automatically created and added to the heap. When the RetrievalSecondLeg occurs ④, the AMR is once more marked as free ⑤. All event triggers lead to intuitive constant time state updates, e.g., AMR and pallet positions.

Routing and Runtime. SLAPStack is efficient in terms of runtime. All state updates run in constant or amortized constant time. Since, future_events is a binary heap, pushing Order events to it during initialization and Transport events to it during step takes logarithmic time with respect to the total number of orders n. In total, the heap manipulation operations have an asymptotic runtime of $O(n\log(n))$. However, a particularity of SLAPStack is the fine-grained routing mechanism implemented. Whenever a Transport event is created, the exact closest grid cell route between the AMRs current position and its destination is computed. A naive implementation lead to an additional overhead of $O(m\log(m))$, where m is the

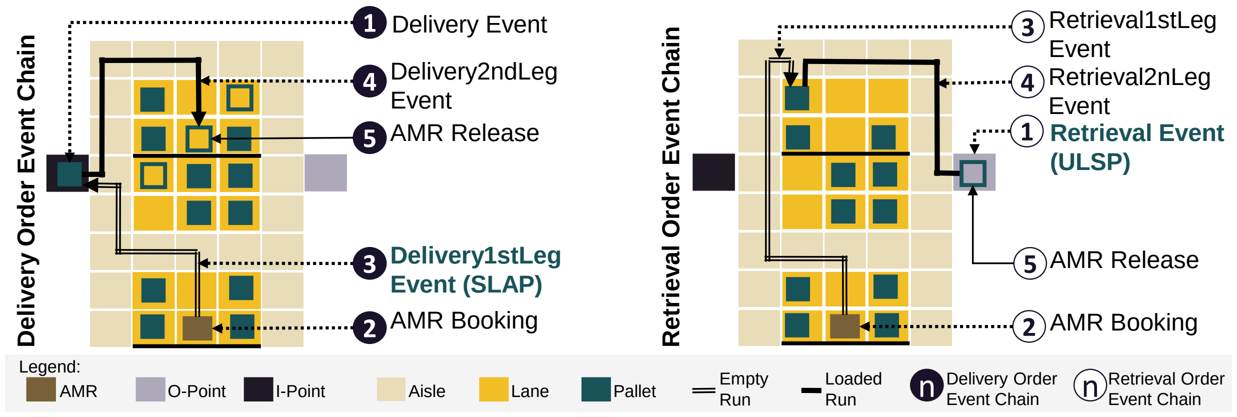


Figure 2: The main process flow in SLAPStack (see Rinciog et al. 2022)

number of grid cells, on route computation. To deal with this bottleneck, SLAPStack's RouteManager pre-computes all shortest paths between aisle tiles and all inbound/outbound docks.

3.2 Cross-Docking-Induced Modifications

To enable the fast experiment execution on the use case at hand, three modifications were required. Firstly, SLAPStack was modified to allow for dock-to-dock transits. Secondly, to enable the dual command cycle-inspired heuristics, the AMR implementation was modified to accept order queues rather than single orders. Thirdly, to ensure fast runtime, the route pre-computation scheme was extended and streamlined.

Dock-to-dock Transports. To allow for dock-to-dock transports, the queued_retrieval_orders property of the EventManager is used in conjunction with the open_storage_locations property maintained by LocationManger. In the previous SLPAStack version, to ensure the fast retrieval of correct actions, the LocationManager which is accessible through the state, exposed two methods to be used by control algorithms. For SLAP actions, controls could query the state for valid open storage locations by means of the get_open_locations method. Similarly, whenever a ULSP action was required, controls could make use of the get_sku_locations method. The two methods retrieve data in constant time ($O(1)$) from an open_locations set and a sku_locations mapping of SKU to sets, respectively. These properties are updated between simulation steps in $O(1)$ as well. The present version adds the get_direkt_sink_action method to uncover possible dock-to-dock transits. This method takes an SKU as a parameter and uses it to check the queued_retrieval_orders for a matching SKU. If one is found, the dock associated with the order is returned. To maintain an $O(1)$ runtime during possible action queries, the retrieval order data structure was changed from a list to a dictionary indexed by SKUs.

Order Queueing on AMRs. Having an AMR execute a dual command cycle is equivalent to concatenating the event chains in Figure 2 and requires minimal modifications to the AMR class and the handle function of DeliverySecondLeg events. The AMRManager maintains an index of the AMR objects currently present in the simulation. On AMR booking ②, the AMR picked for order execution is added to the associated Travel event. Whenever events get popped from the future_events heap, they are temporarily stored in the previous_event property of the core object. Since SLAP decisions occur after a DeliveryFirstLeg event ③, the AMR handling the transportation task can be retrieved from the previous_event property. The amr_index property of the AMRManager can then be used to retrieve the exact AMR object handling the DeliverySecondLeg event soon to be created ④. From this object, the current AMR position can be read and used for calculating the shortest distance to the older retrieval order or the shortest leg. Different from the previous SLAPStack version, AMR objects now have a property called dcc_retrieval_order, wherein further orders can be pushed. When handling DeliverySecondLeg events, this property is checked. If a Retrieval order is present, instead of releasing the AMR ⑤, the position of the closest item with the SKU defined by the order is retrieved, and a RetrievalFirstLeg ❸ is created and added to future events without outside control intervention. Thereby the ULSP decision ❶ and AMR booking ❷ steps of the retrieval event chain are skipped. The retrieval event chain then continues until the AMR release ❺. Note that using further modifications very similar to the ones lain down, any number of orders can be booked to an AMR, meaning that externally compiled order servicing schedules can be evaluated within SLAPStack.

Routing Extension. Different from the traditional SLAP heuristics, which result in paths between input/output docks and storage tiles only, dual command cycle strategies additionally require computing routed between storage tiles. Storage tile to storage tile routes are not pre-computed, constructing such paths, requires running Dijkstra anew, thereby breaking SLAPStack's log-linear total runtime. To eliminate this bottleneck, the path pre-computation was widened to include all routes between aisle tiles not just between inbound/outbound docks and aisle tiles. To this end, the Floyd-Warshall (Mehlhorn and Sanders 2008) all-pair shortest path algorithm was used. Additionally, the Cython implementation of the pathfinding algorithm was replaced with a scipy implementation, eliminating the need for compiling the project, which in turn streamlines the simulation framework's installation process. Dual command cycle heuristics can retrieve the exact distance between storage tiles by calling the get_distance function of the RouteManager.

4 CROSSSTACKS

Along with the open-source simulation, a novel benchmark dataset, CROSSStacks, is offered. Its construction is based on two weeks of orders of a medium-sized, manually operated cross-docking terminal for food industry goods run by a large freight company. The use case setup is detailed in Section 4.1. The data provided was gathered with the help of a process mining company focusing on increasing the transparency of (manual) industrial processes without revealing any sensitive customer information. The Dataset construction process is described in Section 4.2.

4.1 Warehouse Setup

The freight company's warehouse spans a length of 82m and a width of 52m. The storage area is accessible via 21 output docks arranged equidistantly along the length of the warehouse. The 19 input docks are positioned to directly face the first 19 output docks on the opposite side. This layout contains a total of 2150 storage locations on the ground. The storage locations are grouped into two rows of 14 storage bays each, with 25 lanes per bay that can be accessed from two opposite directions. Each lane is designed to accommodate three storage locations. This warehouse does not allow stacking.

During a 15-day period, a total of 16,800 orders are received at the warehouse, with an equal number of delivery and retrieval orders (8,400 each). Orders are not received during the night, and there are no orders on weekends. At the beginning of each day, the warehouse is mostly empty, ready to receive and fulfill orders. Small deviations from this rule occur because of warehouse external factors, e.g., truck delays. However, these deviations are infrequent and do not significantly impact warehouse operations.

4.2 Dataset and Assumptions

The data obtained from the freight and process analysis companies consists of two components, namely the layout specification, and a table tracking the movement of localization sensors attached to human forklift operators between different stations on the warehouse premises. Aside from source, destination, and distance information, the dataset contains a process breakdown by time (e.g., loading/unloading time, time spent securing the load, and time driving the forklift). Travel sources and destinations are vaguely defined by means of warehouse regions, as depicted by Figure 3. Herein the storage bays (black rectangles with a transparent gray fill) extracted from the exact layout provided by the freight company, are overlayed with the regions defined by the process mining company. Green regions mark different segments of the storage area, yellow boxes mark inbound docks, brown boxes mark outbound docks and white-filled black rectangles mark regions outside the storage space. Note the lack of correspondence between storage bays and storage regions. The exact positions of sensors within particular regions are not tracked. Additionally, no product or item information is provided. Furthermore, no direct mapping between operators and sensors is possible, thereby guaranteeing data privacy. Within this setup, the warehouse processes are transparent enough for analysis while obstructing any potentially sensitive information related to goods, exact process execution, or personal data.

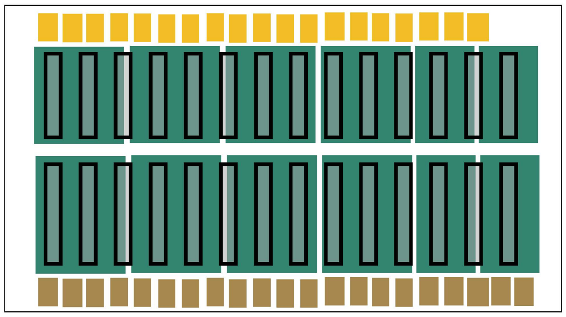


Figure 3: Inbound dock (yellow), outbound dock (brown) and storage (green) movement regions overlayed with storage by positions. Exact movement inside the regions is not tracked for privacy.

SLAPStack requires two key inputs to run a use case, namely a layout and an order list (see Pfrommer et al. (2022)). SLAPStack is currently limited to defining warehouse layouts over square unit tiles which need to be sufficiently large to fit an item. Given this limitation, setting the unit distance slightly smaller than the pallet, i.e., 1.1m instead of 1.2, and arranging 5518 square tiles in 89 columns and 62 rows yields the layout closest to the specification in the previous subsection. With a total length of 97.9m and a width of 68.2m, the simulation layout is somewhat larger than the real-world warehouse. The storage bays and input/output docks were positioned to approximate the exact layout specification of the freight company.

Aside from the complex order generation process, an extension of the SLAPStack order structure was required. Orders originally consisted of six fields, namely the order type (delivery or retrieval), an SKU number, the arrival time in seconds, the dock the order arrives at, a batch number, and the week number. In the use-case at hand batch numbers represent distinct truck loads. The last field is not relevant to the present scenario and was initialized to zero for all orders. Because in cross-docking the destination-dock is known in advance, a seventh field representing it was added to the order structure. The required information from the process mining company is generated in five Steps. First, records were filtered out if they (1) did not contain an input or an output dock as either source or destination region, (2) contained process times clearly outside order servicing (e.g., times spent in the office) or (3) consisted of empty runs. Secondly, the order type was assigned based on the transit source and target respectively: transits from input docks were marked as delivery orders while transits ending in an outbound dock were marked as retrieval orders. Thirdly, the delivery order transits were sorted by time and assigned sequential integers as IDs. Fourthly, an approximate mapping of delivery to retrieval orders was created by matching the sequence of orders flowing into a storage zone to the sequence of retrieval orders exiting that zone assuming a first-in-first-out material handling scheme. Fifthly, this mapping was used to assign both the retrieval order SKUs and the delivery order destination dock.

CROSSStacks imposes constraints on the maximal size of the (virtual) delivery and retrieval queues, as well as the maximum truck loading times. To ensure trucks do not queue up and assuming each truck has a capacity of 33 pallets, the number of queued delivery orders can not exceed 627 (33×19). Similarly, the number of queued retrieval orders can not exceed 693 (33×21). To additionally force fast truck loading times, which stands to reason if one considers the cross-docking scenario, service times no longer than 1800s, i.e., 30 minutes, are required.

5 EXPERIMENTS

The present experiments reveal the optimization potential of AMR systems when combined with an adequate SLAP strategy. In Section 5.1, the SLAPStack-compatible implementation of the storage strategies is detailed and the simulation parametrization is lain down. Since AMRs tend to be slow, SLAPStack was parameterized with significant material handling penalties. To determine the required number of AMRs, a simulative bottleneck analysis is performed in Section 5.2. Evidence is provided, that SLAP heuristics which are impractical or infeasible in manually operated warehouses, specifically COL and SL, outperform a common strategy in cross-docking, i.e., CTD. Note that, save for RND, the ran experiments are deterministic. For experiment reproducibility, RND makes use of random number generator seeding.

5.1 SLAP Strategy Implementation and SLAPStack Parametrization

Save for the dual command cycle heuristics, the SLAP strategies tested are fairly straightforward in terms of their integration with SLAPStack and need not be discussed at length. COL loops over the open locations provided by SLAPStack and picks the one closest to the dock associated with the delivery order that triggered the need for a SLAP decision. Similarly, CTD retrieves the feasible storage locations from the simulation framework and compares their distance relative to the known destination dock of the delivery order to be handled. RND picks a random open location from the set provided by the current simulation environment. Note that any and all storage allocation heuristics should make use of the RoutingManager's get_distance function to leverage the simulation's path pre-computation.

SL and CTNR seek to minimize travel distance by combining delivery and retrieval orders. An AMR servicing a delivery order will proceed to execute a retrieval order close by without being released. Their implementation uses the order queueing on AMRs mechanism and operates on the queued_retrieval_orders structure (see Section 3.1). First, the orders are retrieved from this queue. In the case of CTNR, the oldest order needs to be popped from the retrieval queue. To avoid looping over the queue, the EventManager's pop_queued_retrieval_order method should be used. Leveraging dictionaries and ordered lists, this operation takes constant time. Having found the oldest retrieval order, all viable SKU positions are retrieved using the LocationManager's get_sku_locations method. Because in the present use case, SKUs are unique, exactly one location will be returned. At this point CTNR loops through the retrieved open locations retrieved and selects the one closest to the desired retrieval order's SKU position.

SL works analogously, with the exception that it loops over SKU locations associated with *any* delivery order and not just the oldest. Additionally, SL considers not only the distance between the candidate positions for storage and retrieval but also the distance from the inbound dock to the storage position and the distance from the retrieval position to the outbound dock. Having found the SKU location-open location combination with the shortest total distance (dock to open location to retrieval position to dock), the final two steps are: (1) pop the order from the retrieval queue and (2) append it to the order queue of the AMR object booked for executing the storage operation. Said object is retrievable using the core's previous event property in conjunction with the amr_manager property of the state: state.amr_manager.amr_index[core.previous_event.amr_id].

Aside from the layout, order stream, number of stacks and unit distance discussed in the previous section, three essential additional parameters need to be set, that is the material handling time, the related pallet shift penalty, and the AMR speed. To reflect the incipient stage of AMR technology, the material handling parameter is set to a relatively high 45 seconds. In contrast, skilled forklift drivers only need several seconds for the corresponding operation. Since AMRs either pick up or store a pallet once per Travel event leg, this penalty is simply added to the event occurrence time on Travel event instantiation prior to pushing it to the future_events_heap. The pallet shift penalty, which is incurred whenever a pallet blocked by others needs to be retrieved from the back of the lane, is the double of the material handling penalty (90 seconds). The pallet shift penalty accumulates for each pallet that needs to be removed from a lane to access the desired location (Pfrommer et al. 2022). The AMR speed is set to more or less match the human operator speed on loaded runs inferred from the data. To this end, all the available traces containing a nonzero value loaded run value are isolated from the available data. The distance between the midpoint regions associated with the trace is calculated and divided by the process time to obtain an estimate of the travel speed. Averaging all such speed values yields 1.2 m/s. Note that this still places the AMR at a slight disadvantage compared to human operators, since the simulation uses this travel speed for both loaded and empty runs.

5.2 Results

The experiment is ran in two phases. First, SLAPStack is used in conjunction with CTD to establish the number of AMRs needed to tackle the feasibility challenge described in Section 4.2. Thereafter the implemented storage strategies are compared with respect to five (KPIs).

Bottleneck Analysis. Different from the fixed parameters described above, the number of AMRs has to be determined experimentally, by running the simulation and monitoring the average service time and buffer size bottleneck KPIs. To that end, the simulation is run using CTD as a SLAP strategy while gradually increasing the number of AMRs. The indicators exposed by SLAPStack's state by means of the Trackers property were used for monitoring the bottleneck KPIs at each decision step. The service time indicator is defined as the time between order arrival and order completion and includes the time an order spends waiting to be assigned an AMR, the AMRs travel time, the material handling time, and any pallet shift penalty that may occur. Individual service times are averaged over all orders. As soon as one of the constraints imposed by the use case, i.e., the average service time is higher than 1,800 seconds or the number of delivery or retrieval orders is higher than 627 and 693 respectively, SLAPStack's execution is stopped and started anew with a higher number of AMRs. This loop continues until the simulation terminates without breaching the imposed constraints.

Figure 4, which displays the behavior of the average service time for different numbers of AMRs, shows that eight AMRs are needed for warehouse operations. For concision, only the last four experiments were plotted. The constraint value is represented by the dotted red line. For the present use case, the tighter constraint is induced by the average service time. When the simulation runs halted because of the service time constraint infringement, the recorded delivery and retrieval buffer sizes were both under 200.

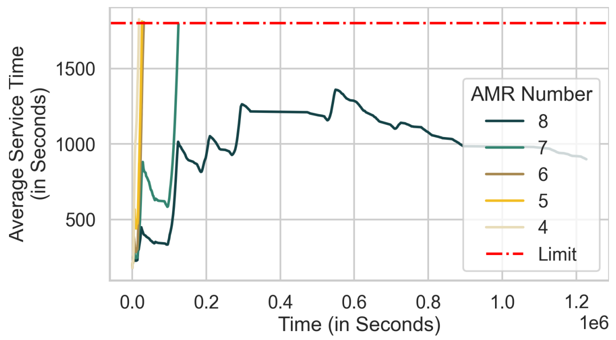
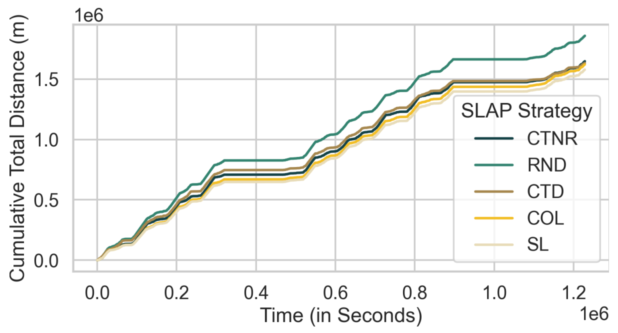
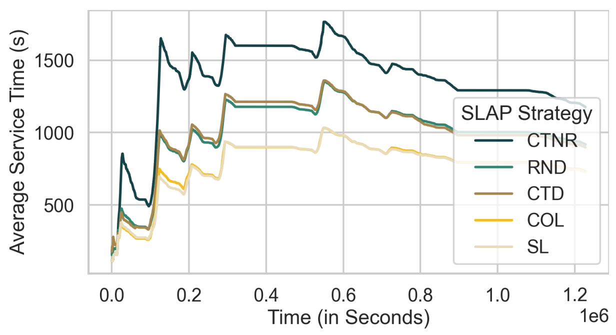


Figure 4: Number of AMRs required for average service time constraint satisfaction.

Storage Strategy Comparison. Figure 5 shows the evolution of the total distance traveled by all AMRs (Figure 5a) and average service time (Figure 5b). The data was generated by using a logger implementing the SlapLogger interface to accumulate selected KPI values on each simulation event occurrence, including non-blocking events. The figure reveals the following aspects. First, SL is the best storage strategy both in terms of travel distance and service time with COL as a close second. However, the distinction between the two heuristics is only marginal with respect to both indicators. Secondly, CTNR is by far the worst heuristic. The main intuition behind CTNR was to target retrieval orders for service time reduction by ensuring a FIFO processing thereof, while simultaneously reducing the travel distance. The poor performance of the heuristic suggests that focusing too much on retrieval orders is a bad strategy. Thirdly, the baseline heuristic, CTD, is only marginally better than random in terms of average service time. On the one hand, this reinforces the relevance of this study, since CTD is a good approximation of the strategy employed by humans in cross-docking. On the other hand, this result may suggest that the spatial distribution of AMRs in the warehouse is a relevant driver of performance. Within SLAPStack when AMRs are freed, they remain in place at the position associated with the last order they completed, e.g., the sink in the case of a retrieval order. As such, CTD, while minimizing retrieval event distance, leads to AMR idle positions being closer to outbound docks. Conversely, RND most likely distributes the AMRs within the warehouse more evenly. Since the two heuristics are so close in terms of performance, the impact of minimizing retrieval distances seems to be on par with more uniformly distributing AMRs in the warehouse.



(a) Total Distance Traveled



(b) Average Service Time

Figure 5: Behavioral comparison of the four best storage strategies.

The exact numeric values associated with the experiment (Table 1) shed additional light on the investigated SLAP heuristics' behavior. The results achieved by the heuristics in terms of the three metrics already discussed (travel time, service time, and utilization) along with the delivery and retrieval queue sizes, the average turnover, and the average hourly throughput are indexed in Table 1. Aside from the absolute values achieved by the controls, the percentual difference between these and the corresponding baseline values (CTD) are noted in parentheses. The baseline is separated from the other heuristics by means of a dashed line. The results of the best heuristic for the particular column are highlighted in bold.

Table 1: Metric values for the storage strategies compared. The best results achieved are set in bold. The baseline strategy is marked by the dotted line.

	Metric				
Storage Strategy	**Average Service Time (s)**	**Total Distance (km/AMR)**	**Max. Delivery Queue**	**Max. Retrieval Queue**	**Utilization Rate**
SL	**729.21 (-18.74%)**	**198.19 (-3.31%)**	141.00 (15.57%)	**89.00 (-25.83%)**	**0.32 (-3.03%)**
COL	732.27 (-18.40%)	203.50 (-0.72%)	**119.00 (-2.46%)**	100.00 (-16.67%)	**0.32 (-3.03%)**
CTD	897.35	204.98	122.00	120.00	0.33
RND	917.98 (2.30%)	232.85 (13.60%)	125.00 (2.46%)	120.00 (0.00%)	0.34 (3.03%)
CTNR	1176.44 (31.10%)	206.65 (0.81%)	188.00 (54.10%)	119.00 (-0.83%)	**0.32 (-3.03%)**

Three aspects can be observed. First, the exact results further reinforce that SL is the best storage strategy for the scenario at hand. SL offers a 18.74 % decrease in service time and a 3.31 % decrease in service time relative to CTD. While COL and SL are almost equivalent in terms of service times, COL barely offers an improvement in terms of traveled distance (less than 1 % relative to CTD). Secondly, the order queue columns reinforce the skewed spatial distribution of the AMRs hypothesis. In the SL case, the delivery queues are 26 % longer than in the CTD case. This is likely because when using SL, AMRs must often travel the entire width of the warehouse to service delivery orders. Thirdly, similarly to the use case described by Pfrommer et al. (2022), the average AMR utilization is around 33 %. This is due to inactivity at night and on weekends. Since AMR systems can work 24/7 and the destination docks of products are known in advance, off times could be used to sort the warehouse by destination, e.g., as described by Pfrommer et al. (2022). This could lead to faster service times.

6 CONCLUSION

This work first extended the SLAPStack simulation framework introduced by Pfrommer et al. (2022) to cover cross-docking setups and allow for the fast execution of dual command cycle-inspired heuristics. Specifically, a mechanism enabling simulation controls to queue multiple orders to the same AMR was added and the framework's fast routing functionality was extended. Due to the SLAPStacks' modular design, these extensions were implemented with relative ease. Secondly, a realistic simulation input set, CROSSStacks, was created. CROSSStacks consists of a layout and an order stream based on 15 days of data provided by a freight company in conjunction with a process mining company. The provided data was sufficiently transparent to generate the simulation inputs while not revealing any sensitive information. Thirdly, two dual command cycle-inspired SLAP heuristics were implemented, namely Closest to the Next Retrieval Order (CTNR) and Shortest Leg (SL), and baselined against a common industry practice in cross-docking, i.e., placing pallets as close as possible to their intended outbound dock (Closest to Destination – CTD), and the greedy Closest Open Location strategy (COL). For CROSSStacks, SL is the best strategy for optimizing both travel distance and service time, outperforming the baseline 18.74 % and 3.21 % respectively. The experiments carried out additionally revealed that further optimization potential could lie with the distribution of AMRs in the warehouse and the sorting of pallets based on the distance to their target dock.

The present elaboration reveals the following streams of future work. First, the dual command cycle heuristics could be further evaluated by implementing a spatial redistribution scheme AMRs can follow when idle. Additionally, simulations on layouts with inbound and outbound docks on the same side of the warehouse should be run. Secondly, to monitor the effects of the AMR spatial distribution, an indicator measuring the uniformity of the distribution in a 2D space must be developed and integrated with SLAPStack. Thirdly, algorithms for sorting items by destination should be implemented and evaluated. Fourthly, researchers could follow up on the order queuing on the AMR mechanism by extending it. Order execution schedules could then be generated, assigned to AMRs, and evaluated using SLAPStack. Finally, given that SLAPStack was explicitly designed for reinforcement learning (RL), such algorithms could be used for solving SLAP.

REFERENCES

Bortolini, M., F. G. Galizia, M. Gamberi, and F. Gualano. 2020. "Integration of Single and Dual Command Operations in Non-Traditional Warehouse Design". *The International Journal of Advanced Manufacturing Technology* 111(9–10):2461–2473.

Clements, K., K. Sweeney, A. Tremont, V. Muralidhara, and M. E. Kuhl. 2016. "Evaluation of Warehouse Bulk Storage Lane Depth and ABC Space Allocation Using Simulation". In *Proceedings of the 2016 Winter Simulation Conference*, edited by T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, 2239–2249. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Frazelle, E. H. 2016. *World-Class Warehousing and Material Handling*. New York: McGraw-Hill Education.

Goetschalckx, M., and H. D. Ratliff. 1990. "Shared Storage Policies Based on the Duration Stay of Unit Loads". *Management Science* 36(9):1120–1132.

Gu, J., M. Goetschalckx, and L. F. McGinnis. 2007. "Research on Warehouse Operation: A Comprehensive Review". *European Journal of Operational Research* 177(1):1–21.

Hausman, W. H., L. B. Schwarz, and S. C. Graves. 1976. "Optimal Storage Assignment in Automatic Warehousing Systems". *Management Science* 22(6):629–638.

Heskett, J. L. 1964. "Putting the Cube-Per-Order Index to Work in Warehouse Layout". *Transportation and Distribution Management* 4(8):23–30.

Le-Duc, T., and R. B. De Koster. 2005. "Travel Distance Estimation and Storage Zone Optimization in a 2-Block Class-Based Storage Strategy Warehouse". *International Journal of Production Research* 43(17):3561–3581.

Mahajan, K. R., C. Laroque, W. Dangelmaier, M. Kortenjan, C. Soltenborn, and D. Kuntze. 2005. “D3 Fact Insight: A Motion Planning Algorithm for Material Flow Simulations in Virtual Environments”. In *Proceedings of the 2005 Simulation and Visualization Conference*, edited by T. Schulze, G. Horton, B. Preim, and S. Schlechtweg, 115–126. Erlangen: SCS Publishing House e.V.

Mehlhorn, K., and P. Sanders. 2008. *Algorithms and Data Structures: The Basic Toolbox*. Berlin: Springer.

Merschformann, M., L. Xie, and H. Li. 2017. “Rawsim-O: A Simulation Framework for Robotic Mobile Fulfillment Systems”. *Arxiv Preprint ARXIV:1710.04726*.

Pan, C.-H., and C.-H. Wang. 1996. “A Framework for the Dual Command Cycle Travel Time Model in Automated Warehousing Systems”. *International Journal of Production Research* 34(8):2099–2117.

Pfrommer, J., and A. Meyer. 2020. “Autonomously Organized Block Stacking Warehouses: A Review of Decision Problems and Major Challenges”. *Logistics Journal: Proceedings. doi:*10.2195/$lj\_Proc\_pfrommer\_en$_202012_01.

Pfrommer, J., A. Meyer, and K. Tierney. 2022. “Solving the Unit-Load Pre-Marshalling Problem in Block Stacking Storage Systems With Multiple Access Directions”. *Arxiv Preprint ARXIV:2207.09118*.

Pfrommer, J., A. Rinciog, S. Zahid, M. Morrissey, and A. Meyer. 2022. “Slapstack: A Simulation Framework and a Large-Scale Benchmark Use Case for Autonomous Block Stacking Warehouses”. In *Proceedings of the 2022 International Conference on Computational Logistics*, edited by J. de Armas, H. Ramalhinho, and S. Voß. Cham, Switzerland: Springer International.

Rinciog, A., and A. Meyer. 2021. “Fabricatio-Rl: A Reinforcement Learning Simulation Framework for Production Scheduling”. In *Proceedings of the 2021 Winter Simulation Conference*, edited by S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper, 1–12. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Rinciog, A., J. Pfrommer, M. Morrissey, S. Zahid, and A. Meyer. 2022. “Slapstack”. *Github Repository, https://github.com/malerinc/slapstack.git, accessed 31st August 2023*.

Schwarz, L. B., S. C. Graves, and W. H. Hausman. 1978. “Scheduling Policies for Automatic Warehousing Systems: Simulation Results”. *Aiie Transactions* 10(3):260–270.

Van Belle, J., P. Valckenaers, and D. Cattrysse. 2012. “Cross-Docking: State of the Art”. *Omega* 40(6):827–846.

Vis, I. F., and K. J. Roodbergen. 2008. “Positioning of Goods in a Cross-Docking Environment”. *Computers & Industrial Engineering* 54(3):677–689.

Werners, B., and T. Wülfing. 2010. “Robust Optimization of Internal Transports at a Parcel Sorting Center Operated by Deutsche Post World Net”. *European Journal of Operational Research* 201(2):419–426.

AUTHOR BIOGRAPHIES

ALEXANDRU RINCIOG is a PhD Candidate at the Mechanical Engineering Department of the TU Dortmund University. He earned his M.Sc. in Computer Science from Karlsruhe Institute of Technology in 2019. He works on combinatorial optimization problems using simulation and machine learning. Email: alexandru.rinciog@tu-dortmund.de.

NATALIA OGORELYSHEVA is a researcher at the Fraunhofer Institute for Material Flow focusing on embeded systems and automation in logistics since 2018. In 2022 she obtained her Masters Degree in Computer Science from the TU Dortmund University. Email: natalia.ogorelysheva@iml.fraunhofer.de.

JAKOB PFROMMER is a PhD Candidate at the Mechanical Engineering Department of the TU Dortmund University since 2019 and the coordinator of the DFG research training group GRK2193. His research focuses on decision problems in block-stacking warehouses. He obtained his M.Sc. in Industrial Engineering from the TU Darmstadt and KTH Stockholm. Email: jakob.pfrommer@tu-dortmund.de.

ANNA VASILEVA is a researcher at the Fraunhofer Institute for Material Flow since 2022. Her main research focus is virtualization and simulation. She obtained her PhD from the TU Dortmund University in 2023 and has six years of research experience at Computer Science and Mechanical Engineering departments. Email: anna.vasileva@iml.fraunhofer.de.

HARDIK RATHOD obtained his M.Sc. in Automation and Robotics from the TU Dortmund University in 2023. He is currently working on automation systems as an employee of fruitcore robotics GmbH. Email: hardik.rathod@tu-dortmund.de.

ANNE MEYER is professor for Digitalization in Logistics and Supply-Chain management at the TU Dortmund University and co-director of the FZI Research Center for Information Technology in Karlsruhe. She obtained her Ph.D. from Karlsruhe Institute of Technology in 2015 and accumulated more than 10-years of research experience at FZI in the fields of logistics, optimization, and data analytics. anne2.meyer@tu-dortmund.de. Site: https://dls.mb.tu-dortmund.de.