Sequencing With Decentralized Control Using Modular Highest-Density Conveyor Systems

Julia Fleischmann and Kai Furmans, *Member, IEEE*

***Abstract*— We present a decentralized sequencing algorithm applicable to modular high(est)-density conveyor systems that enables items to enter the system randomly from multiple input points. They are rearranged within the system to be retrieved at an assigned output point observing a predefined sequence. A variety of applications benefits from this, such as storing and retrieving, order picking, packing and shipping operations as well as Just-in-Sequence (JiS) applications of production systems. We prove that our decentralized algorithm – based on the concept of logical time – is able to prevent deadlocks, livelocks, and starvation. It guarantees a complexity of order $\mathcal{O}(n^2)$ at module level depending on system size n. Using throughput evaluations, we show how to parameterize the decentralized sequencing algorithm to optimize its performance. Furthermore, we deduce that small batch sizes, high conveying speeds, a high number of output points, or non-perforated network structures increase system throughput. A simulative demonstration of the developed algorithm can be found at our institute channel.**

***Note to Practitioners*—The need for flexibility, scalability, and robustness has led to decentralized control concepts becoming more and more popular in practice. In the field of material handling systems, transportation, storage, sortation, and picking functionality is already being realized based on decentralized control. In this paper, we present a decentralized control algorithm which dispatches randomly introduced items to match a specified sequence at their assigned destination. It applies to systems including multiple input and output points (($m : n$) setting) and thus allows for parallel order processing and customizable system configurations of various application scenarios. As sequencing combines routing, buffering and relocation functionality, additionally, an overall logic for implementing transport, storage and retrieval, sortation and picking systems is provided as well.**

***Index Terms*— Material handling, Plug-&Play, logical time, deadlock-free, decentralization, sequence, complexity analysis.**

I. Introduction

TODAY'S highly complex, dynamic, and uncertain industrial environments require sophisticated automated systems to operate efficiently. Firms are looking for solutions that can be deployed quickly, reconfigured without long downtimes, and are robust to local failures. Furthermore, automation is also being challenged to operate in smaller spaces to reduce costs or to operate closer to the customer. Decentralized systems offer such flexibility, robustness, scalability, and fault-tolerance [1] because of their modular design that only requires localized information and decision-making. This allows, for example, a company to flexibly extend or modify, maintain and move their material handling equipment without external expertise. While decentralized system design provides this number of benefits, a major challenge is their high risk for creating deadlocks, livelocks, and starvation [2], as the overall system state can never be analyzed in its entirety. Additionally, the necessary decentralized communication effort may affect system performance.

Production and logistics environments often require products or parts to arrive in a pre-defined sequence for being handled or processed. In storing and retrieving, order picking, packing and shipping operations as well as Just-in-Sequence (JiS) applications, manual work is reduced and overall process efficiency increases if items arrive at their point of use in the correct order of consumption. This work thus presents a decentralized sequencing algorithm, where items – referred to as transport units (TUs) – randomly arrive at multiple input points, are rearranged within the system, such that they exit the system at an assigned output point observing the predefined sequence required at their point of use. TUs represent physical unit loads, such as parcels, boxes, small load carriers etc. or even bulk materials, liquids or gases when combined with suitable packaging and/or load carriers. These are organized in *batches* of a certain *size k* corresponding to the number of TUs included in the batch. Each TU of a batch is assigned a rank to specify the predefined unloading sequence at its output points. We consistently characterize batches using colors, while the ranks of TUs within a batch are denoted with numbers.

The system is made up of identical right-angle-transfer conveyor modules without any supervising element. These act autonomously based on local data bases and control units. Connecting several adjacent modules creates the overall sequencing system. The presented algorithm is capable of operating in highest-density[1] conveyor systems, i.e., it guarantees maximum achievable space efficiency. In Figure 1 we show an example use case, where three input conveyors provide items from an upstream supplying station for sequenced delivery to four processing stations of a downstream process.

The paper is structured as follows: Section II reviews the current state of the art regarding the investigated sequencing

Manuscript received 31 October 2022; revised 8 March 2023; accepted 26 April 2023. This article was recommended for publication by Associate Editor Y. Tong and Editor C. Seatzu upon evaluation of the reviewers' comments. This work was supported by the Bundesministerium für Wirtschaft und Klimaschutz (BMWK) through by the Research Project "MultiSequencer-Development of a Compact, Flexible m:n Sequencer" under Grant ZF4251413DB8. *(Corresponding author: Julia Fleischmann.)*

The authors are with the Department of Mechanical Engineering, Institute for Material Handling and Logistics, Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany (e-mail: julia.fleischmann@kit.edu; kai.furmans@kit.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TASE.2023.3272971.

Digital Object Identifier 10.1109/TASE.2023.3272971

[1]*Highest-density* is an extreme case of *high-density* as defined in [3]. Details are given in Chapter II.

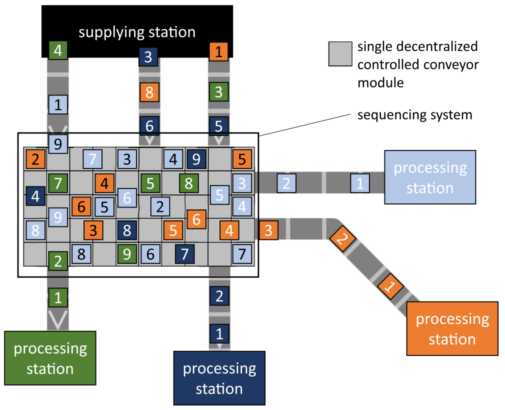

Fig. 1. Sequencing use case, where items are sequenced according to their number.

problem. In Section III, we present the decentralized sequencing algorithm by specifying the interactions at module level as well as the decentralized system design. The absence of deadlocks, livelocks, and starvation is proven in Section IV. We investigate the algorithmic complexity in Section V. Section VI analyzes the system performance using simulation studies. We summarize the central conclusions of our work in Section VII.

II. State of the Art

Decentralized algorithms are already applied to realize specific material handling operations using modular conveyor systems (cf. Section II-A). Restrictive approaches for sequencing systems have been developed as well (cf. Section II-B).

A. Conveyor Systems With Decentralized Control

Existing approaches in the field of decentralized controlled conveyor systems enable transporting, storing, picking, and sorting functionalities. These are summarized in Table I. The system density follows from [3]. In *high-density* systems, interfering items prevent accessing desired items. We further define *highest-density* systems as those which can operate with just one empty module. The implemented routing strategy is classified into *offline* and *online* approaches. With the former, item routes are planned entirely from start to destination before initiating transportation, whereas with the latter, they are planned incrementally with stepwise item movements [4]. Systems with online routing are synchronized involving a higher level coordination element to ensure that all modules execute the algorithmic phases consistently. Therefore, buffer times are incorporated within the negotiation cycles, which can reduce system performance [5]. Furthermore, they necessarily require rectangular grid layouts [6]. Based on the deadlock handling strategies according to [7], we investigate whether and how the presented algorithms ensure system liveliness.

Low-density systems are studied most commonly incorporating an offline route planning approach. Route planning in high(est)-density systems requires relocating interfering items. So far, this is only possible using online approaches, which suffer from the drawbacks stated above. None of the existing approaches provides a decentralized algorithm based on offline route planning applying to high(est)-density conveyor systems. Thus, preventing deadlocks or ensuring system liveliness in general under these conditions remains unresolved as well. Apart from [11], complexity aspects of the presented decentralized algorithms are not studied.

TABLE I
Conveyor Systems With Decentralized Control

Publication	Functionality	Density	Routing	Deadlock handling
[8]	transport	low	offline	avoidance
[9]	transport	low	offline	avoidance
[10]	transport	low	offline	-/-
[11]	transport	low	offline	avoidance
[12]	store	high	online	prevention
[5]	pick	high	online	avoidance[2]
[13]	pick	highest	online	avoidance
[14]	sort	low	offline	prevention
This paper	sequence	highest	offline	prevention

TABLE II
Approaches for Sequencing Systems

Publication	Decentralization	I/O points	Density	Routing	Deadlock handling
[15]	none	1:1	highest	offline	none
[16]	complete	1:1	high	online	none
[17]	partial	1:1	high	online	prevention
[18]	partial	m:n	highest	online	centralized detection
[19]	partial	m:n	medium	offline	detection
[6]	complete	m:n	highest	online	prevention[3]
This paper	complete	m:n	highest	offline	prevention

B. Approaches for Conveyor-Based Sequencing Systems

Several approaches for conveyor-based sequencing systems have been proposed in scientific research (cf. Table II). We classify them according to their level of decentralization. Processing several batches simultaneously relies on an $(m : n)$ setting with $m > 1$ input and $n > 1$ output points. We investigate density, routing, and integrated deadlock handling strategies by analogy with Section II-A.

All of the presented approaches of Table II offer a solution for sequencing in modular conveyor systems. The majority require centralized system functionality where essential algorithmic parts, such as path finding, data holding, or deadlock handling, are delegated to a centralized element. Often, single batch problems are considered. Several of the proposed systems reserve unoccupied parts for relocations, which reduces the achievable density. Online routing algorithms are used frequently, such that only synchronized systems with rectangular network structures can be realized [6]. None of the presented approaches provides decentralized sequencing from multiple input to multiple output points in general high(est)-density conveyor systems while ensuring system liveliness in terms of deadlock, livelock, and starvation prevention. This paper aims to comprehensively close this research gap.

[2]Not generally proven and only under certain conditions.

[3]Livelock risk is present.

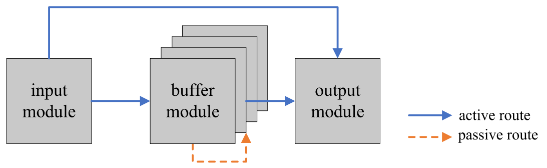

Fig. 2. Splitting the overall route of a TU into active and passive routes.

III. Decentralized Sequencing

We present a decentralized sequencing algorithm starting with the overall design approach to outline its main ideas at system level (cf. Section III-A). Based on that, we detail the concept of decentralized interactions at module level (cf. Section III-B). For demonstration, we use a showcase system based on square right-angle-transfer conveyor modules (cf. Section III-C).

A. Design Approach

The overall route of each TU initially starts at its assigned input module where it is introduced and ends at its assigned output module where it is unloaded. To observe the predefined unloading sequences at the output modules, buffering is necessary. We split the overall routes of buffered TUs into sub-routes, where buffer modules are used as intermediate destinations (cf. Section III-A1). These need to be allocated to support efficient sequencing (cf. Section III-A2). Path finding for each (sub-)route considers the resulting system occupation (cf. Section III-A3).

1) Transport Unit Routing: Processing a TU means specifying its route through the system. We refer to a TU as *requested* if its corresponding output module is able to claim this TU, as all necessary preceding TUs have already been scheduled there. Thus, only requested TUs may be routed to their output module.

Within the decentralized sequencing algorithm, we define two types of routes:

- *active routes* starting at an input module and/or ending at an output module and
- *passive routes* starting and ending at a sequencing module for buffering.

Active routes are necessary to introduce arriving TUs into the system and to unload requested TUs at their output modules. Thus, active routes generally guide the process of sequencing. Passive routes, instead, depend on active routes. If a buffered TU interferes with an active route, a passive route is initiated for relocation. This allows us to represent the overall route of each buffered TU from the input to the output module as a combination of one active route at the beginning and end and an arbitrary number of passive routes in between (cf. Figure 2). In case a TU is requested when arriving at its input module, the two active routes are consolidated such that it is routed directly from the input to the output module.

From the perspective of a TU, sequencing follows the flowchart given in Figure 3. This is achieved by decentralized interaction of the autonomous conveyor modules (cf. Section III-B).

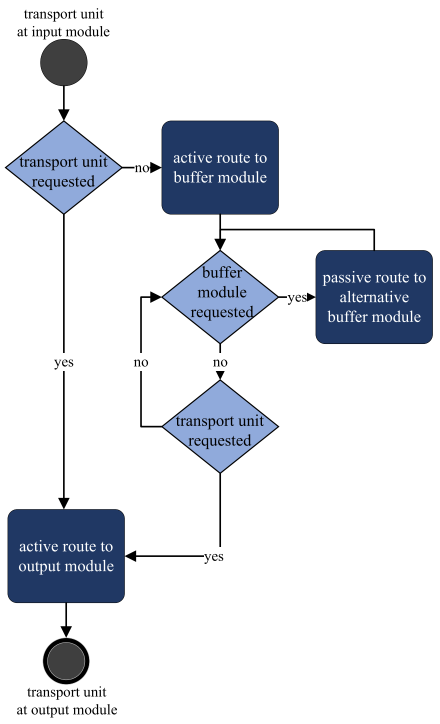
transport unit at input module
transport unit requested
no
active route to buffer module
buffer module requested
yes
passive route to alternative buffer module
yes
no
no
transport unit requested
active route to output module
yes
transport unit at output module

Fig. 3. Flow chart for sequencing from the perspective of a TU.

TABLE III
Buffer Module Allocation

	Active route	Passive route
Requirement	Pre-sequenced buffering to enable fast unloading	Preserve pre-sequenced arrangement
Objective	Buffer predecessors closer to the output module than their successors	Move TU to an adjacent module
Selection rule	Unloading sequence-based buffer selection rule	Distance-based buffer selection rule
Selecting module	Output module	Buffer module of interfering TU

2) Buffer Module Allocation: To enable efficient sequencing, we develop the *unloading sequence-based buffer selection rule* for active routes, while for passive routes, the *distance-based buffer selection rule* applies (cf. Table III). Both are designed to meet different requirements of the specific route type when identifying buffer modules.

3) Path Characteristics: While the path of an active route generally includes multiple modules, each passive route represents exactly one relocation step of a TU to an adjacent module (cf. Table III). Therefore, clearing the path of an active route due to an interfering buffered TU may require moving several TUs in a chain – especially when system density is high. We refer to this chain of relocation steps as a *relocation route*, i.e. a relocation route may combine multiple passive routes in direction to a non-buffering module (cf. Figure 4).

For efficiently processing TUs, paths of active as well as relocation routes aim to consider:

- the length of the route, as this defines the time for transferring the TU,
- the number of buffered TUs on the route, as these need to be relocated to be able to access there, and
- the number of directional changes on a route, as this implies a certain delay due to acceleration and

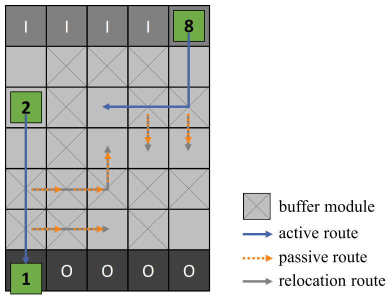

Fig. 4. Use case of two active routes from buffer to output and from input to buffer module, respectively, including their induced passive routes.

deceleration as well as switching the conveyor modules between their rectangular transport directions.

B. Algorithm Concept

The decentralized interactions of the autonomous modules are designed to process TUs within the system as illustrated in Figure 3. Active routes are initiated observing the necessary predecessor-successor relations (cf. Section III-B1). Each active route is planned from start to destination (cf. Section III-B2) before being executed (cf. Section III-B3).

1) Route Initiation: Route initiation notifies the module currently assigned to a TU that planning an active route for this TU can start. Input modules identify arriving TUs and register them at their output module. Based on the current state of its unloading sequence, the latter is able to determine whether the active route for the corresponding TU requires a buffer module or goes directly to the output module. In case of buffering, the output module requests potential buffer modules based on the unloading sequence-based buffer selection rule for active routes (cf. Section III-A2). The output module then returns the destination of the active route to the requesting input module. When scheduling a TU at its output module, previously buffered ones might become requested. Their assigned output module requests such buffered TUs directly at their allocated buffer module to initiate an active route for unloading. Due to relocations of interfering buffered TUs, the local information at the output module about allocated buffer modules may be outdated up to this request. Buffer modules discard outdated requests if the allocation no longer applies. Additionally, each reallocated buffer module updates the corresponding output module, which we will discuss in more detail when presenting passive route planning in Section III-B2.c. For each active route, the message received from the output module triggers route planning at the input and buffer module, respectively.

2) Route Planning: Planning an active route follows the series of acquiring the planning authorization (cf. Section III-B2.a), selecting the path of modules forming the active route (cf. Section III-B2.b), and negotiating reservations for the active and all of its induced passive routes for relocation (cf. Section III-B2.c).

a) Authorization procedure: The decentralized authorization procedure (cf. Figure 5) guarantees deadlock-free system operation within route planning as all routes are planned sequentially. Executing these routes occurs independently of planning procedures (cf. Section III-B3). At any time, there is exactly one authorized module within the system. Only

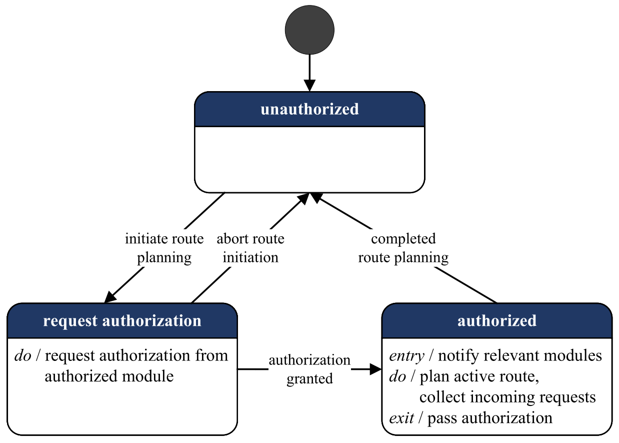
unauthorized
initiate route planning
abort route initiation
completed route planning
request authorization
do / request authorization from authorized module
authorization granted
authorized
entry / notify relevant modules
do / plan active route, collect incoming requests
exit / pass authorization

Fig. 5. Authorization procedure.

this module is entitled to plan an active route. At system configuration, it is initialized with the first input module of the system. Whenever the authorization is passed, all input and sequencing modules are notified as only these initiate active routes.

Each time a module intends to start planning an active route, it first sends a request to the currently authorized module to be authorized next. As soon as this completed planning its active route, it passes the authorization to the next module among the incoming requests. Active routes for unloading are prioritized over those for buffering, as the former directly contribute to system throughput. If an authorization request is not granted, the notification of changed authorization triggers a new request to the currently authorized module.

b) Path selection: Path selection aims to identify the path of modules for an active route based on the criteria defined in Section III-A3. We develop an adapted decentralized A* algorithm, which starts at the initiating module of the active route. Every search iteration occurs locally at the currently best module to be explored. This module is notified to continue the A* search providing all currently available path information to it. It updates this path information, while continuing the open list of known modules by its adjacent modules, and identifies the best module to be explored next. The tentative path cost t^n for an adjacent module n proceeding from module m is calculated as follows:

$$t^n = t^m + d_e^n - d_e^* + \begin{cases} p_b & \text{if } m \text{ is buffer module} \\ 0 & \text{otherwise} \end{cases} + \begin{cases} p_c & \text{if } n \text{ implies directional change} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The current tentative value t^m represents the sum of penalty terms, resulting from relocations and directional changes from the start module to the path predecessor of module m so far. Added to this is the length of the detour $(d_e^n - d_e^*)$ measured in module lengths when routing via neighbor n. The penalty term for buffer module relocation p_b results from the current module state, while the penalty term for directional change p_c results from the transmitted path predecessor of module m $(p_b, p_c > 0)$. p_b and p_c can be parameterized to

TABLE IV
NOTATION FOR DEFINING RESERVATION CONDITIONS (BASED ON [14])

Symbol	Physical equivalent
Processes P_a, P_b	Transport routes of TUs a and b
Process P_{s^a}	Transport route of a successor of TU a within their unloading sequence
Resource R_i, R_j	Conveyor modules i and j
Resource R_o	Output module o
Logical clock C_i	Logical clock of module i
Event $T_{in}\,(P_a, R_i)$	Incoming transfer of TU a at module i
Event $T_{out}\,(P_a, R_i)$	Outgoing transfer of TU a at module i

specifically influence path selection. However, optimality is guaranteed only if the tentative cost estimation is admissible according to the A* requirements, i.e. path costs must never be overestimated (cf. [20]). Adding p_c depends on the path predecessor and may retroactively change the cost estimate of different paths. To satisfy admissibility, p_c cannot be set arbitrarily, which we will discuss in more detail in Section VI-B. Path selection is finished, when the route destination module is notified. The identified path is fixed by confirming the corresponding predecessor modules backward up to the start module of the active route.

c) Route reservation: We apply a time window-based approach using the concept of logical time. It builds upon the preliminary work of [21] and [14]. Each conveyor module is assigned a local logical clock. Transferring a TU requires reserving a logical time window at all modules of its route. Therefore, each module of the system locally holds a reservation table in which all of its transfers are scheduled. These are executed in ascending order of their reserved logical time windows. When completing a transfer, the involved modules update their logical clocks accordingly. Thus, clock times refer to events rather than physical time points.

For route reservation within the developed decentralized sequencing algorithm, the involved modules negotiate logical time windows matching their local reservation tables. Sequencing requires the following *reservation conditions* (R1) to (R6) to be satisfied. We use the notation according to [14] (cf. Table IV).

$$T_{in}(P_a, R_i) < T_{out}(P_a, R_i) \tag{R1}$$

The outgoing transport of each TU at each module is scheduled after its incoming transport there.

$$T_{out}(P_a, R_i) = T_{in}\left(P_a, R_j\right) \tag{R2}$$

Each pair of adjacent modules i and j along the transport route of a TU agrees on the timing regarding their common event. Combining (R1) and (R2) yields ascending logical time windows within the event sequence of each process.

$$T_{out}(P_a, R_i) < T_{in}(P_b, R_i) \tag{R3}$$

Each module can hold only one TU at a time. Thus, the respective logical time windows according to (R1) for two routes scheduled at the same module may not overlap.

$$C_i < T_{in}(P_a, R_i) \tag{R4}$$

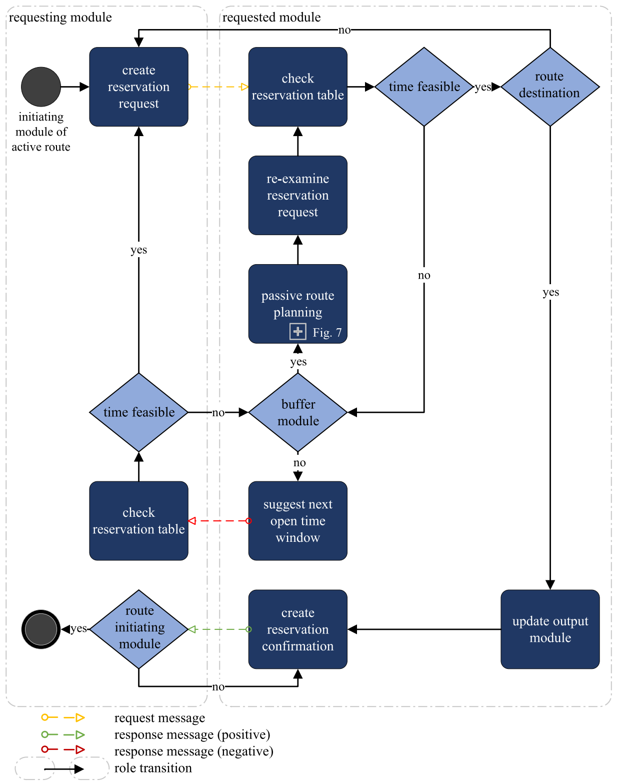

Fig. 6. Reservation of active routes.

Upcoming events are scheduled in future logical time regarding the reserved module to satisfy the causal event dependencies (cf. [21]).

$$T_{in}(P_a, R_o) < T_{in}(P_{s^a}, R_o) \tag{R5}$$

Each succeeding TU may not enter at the assigned output module before any of its predecessors. This upholds the predefined unloading sequences.

$$T_{out}(P_a, R_i) = \infty \tag{R6}$$

Each buffered TU blocks its allocated buffer module for an unlimited time period until its predecessor is planned or it is relocated (cf. Figure 3). Thus, (R6) defines a buffer module.

Establishing a set of feasible logical time windows adhering to reservation conditions (R1) to (R6) is realized via decentralized negotiations according to Figure 6. Adjacent modules iteratively suggest time windows based on their existing reservations until an agreement is found. Due to reservation condition (R6), this requires a passive route planning subroutine for buffer modules of active routes to relocate their blocking TUs.

To negotiate a finite logical time window on buffer modules after the buffered TU enters, the indefinite reservation of reservation condition (R6) needs to be resolved first, i.e. it becomes a non-buffering module. According to Table III, we apply a step-wise relocation approach which moves a buffered TU from its currently allocated buffer module to an adjacent one. As we operate in highest-density systems, this might involve relocating several other buffer modules (cf. Figure 4).

Figure 7 shows the passive route planning. The assigned buffer module of the blocking TU first identifies the closest

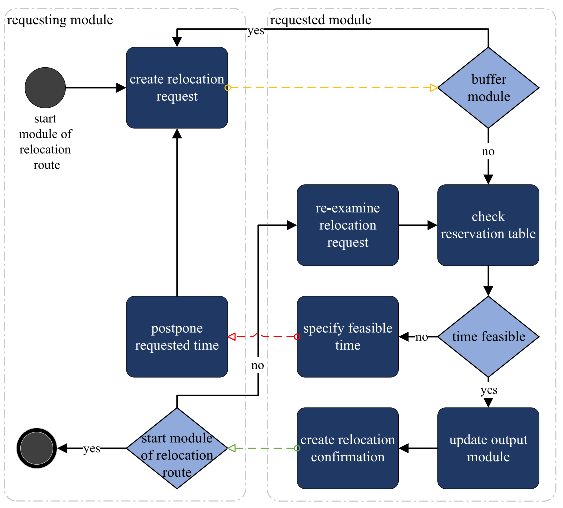

Fig. 7. Passive route reservation.[4]

non-buffering module using the distance-based buffer selection rule for passive routes. Then, relocation requests are successively sent via the port of a shortest path directing to the identified module. To end up with feasible logical time windows at all involved modules according to the defined reservation conditions, the logical time at which a non-buffering module can take over an indefinite buffer reservation dictates the relocation negotiations. After receiving the relocation confirmation at the initially blocked module of the active route, it is now able to respond to its received reservation request (cf. Figure 6).

3) Transport Execution: After confirming the reservation at the start module of an active or passive route, TU movement is initiated. In the decentralized system, transferring a TU occurs if and only if both involved modules agree on the next logical time event within their respective reservation tables. This is coordinated by exchanging transport request and confirmation messages. Whenever a transport event is executed, the involved modules forward their logical clocks: The sending module updates its logical clock to T_{out}, the receiving one to T_{in} of their reserved logical time windows for the corresponding TU. This guarantees an unambiguous transport execution sequence for every module in the system providing deadlock-free system operation during transport execution (cf. Section IV-A).

C. Showcase System

The presented decentralized sequencing algorithm is implemented within a showcase system using right-angle-transfer conveyor modules, such as the *FlexConveyor* of [8]. They can be flexibly combined via *Plug-&Play* technology, which allows creating any kind of customized sequencing system. Each module uses its own control unit for decision making based on a local database. At each of the four module edges, there is an interface for communication with the adjacent modules. Communication with non-adjacent modules within the network requires a corresponding message transmission mechanism, which forwards messages via a series of adjacent modules to the respective recipient. Therefore, any communication protocol is valid which allows specifying the payload as well as the sender and recipient of a message, e.g. TCP/IP.

[4]The legend of Figure 6 also applies to Figure 7.

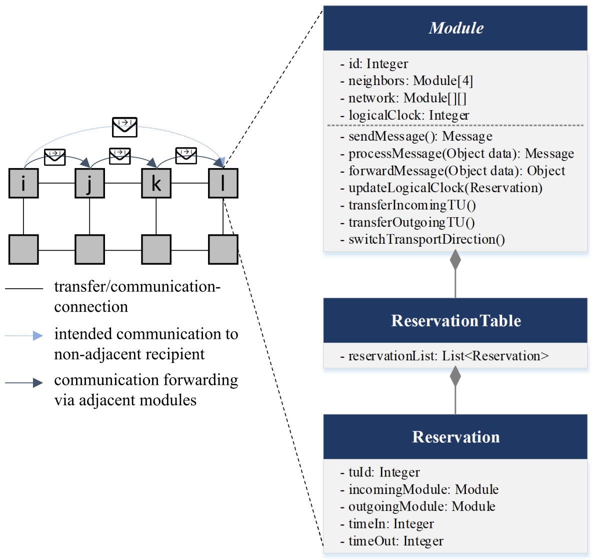
i
j
k
l
transfer/communication-connection
intended communication to non-adjacent recipient
communication forwarding via adjacent modules
Module
- id: Integer
- neighbors: Module[4]
- network: Module[][]
- logicalClock: Integer
- sendMessage(): Message
- processMessage(Object data): Message
- forwardMessage(Object data): Object
- updateLogicalClock(Reservation)
- transferIncomingTU()
- transferOutgoingTU()
- switchTransportDirection()
ReservationTable
- reservationList: List<Reservation>
Reservation
- tuId: Integer
- incomingModule: Module
- outgoingModule: Module
- timeIn: Integer
- timeOut: Integer

Fig. 8. Decentralized system design.

As we presented the algorithmic operations assuming faultless communication and uninterrupted availability of conveyor modules, suitable error handling is vital within real-world applications. The decentralized system architecture based on the local attributes and methods of the conveyor modules to enable sequencing are summarized in Figure 8.

Each module can handle one TU at a time (cf. [8]). Transferring a TU between two adjacent modules requires occupying both of these modules until the TU is entirely located on the next module along its route. We divide the set of all installed conveyor modules into input, output and sequencing modules based on their role within the network. The input modules link from the upstream process to the sequencing system to introduce arriving TUs. Output modules provide them to the downstream process after processing by the intermediate sequencing modules.

For demonstrating our showcase system, we implemented an agent-based simulation model using the simulation software AnyLogic. Each conveyor module is represented by an own agent instance. These agents interact by events, such as sending or receiving messages.

Generally, stable system operation can only be guaranteed if the inflow of TUs arriving at the system can be physically processed using the available buffer capacity of the network. TU movements necessarily require at least one unoccupied sequencing module in the system. Thus, when introducing new TUs, the input modules need to ensure that the sum of TUs within the system does not exceed $(c-1)$ for a network comprising c sequencing modules. Within our simulation model, we generate the input data such that sequencing is always feasible for the set of arriving TUs by restricting the maximum processable batch size to $(c-1)$ as well as the mixing of batches within the arrival characteristics at the input modules.

IV. Preventing Deadlocks, Livelocks and Starvation

Deadlocks, livelocks, and starvation represent a major risk in decentralized controlled systems, as the states of all processes and resources are only locally available. In the following, we prove that the presented sequencing algorithm structurally prevents those due to the specified reservation conditions and communication procedures. We subdivide our investigations into the algorithmic operations as well as the allocation of resources for physically moving TUs, as these both are processed at decentralized module level.

A. Deadlocks

A deadlock arises within a set of processes if each process is waiting for an event of another process in this set [7].

1) Algorithm Operations: The algorithmic operations are deadlock-free if we can guarantee that for each arriving TU a complete route from its input to its output module is initiated in finite time, observing the predefined predecessor-successor dependencies. Assuming that there is always at least one unoccupied sequencing module within the system (cf. Section III-C), a feasible solution exists for the set of all necessary active and passive routes. To ensure that this is found, all modules need to use consistent information. Outdated or incomplete local information may create requests which are mutually exclusive.

Local module information is updated when receiving confirmations. The decentralized authorization concept (cf. Section III-B2.a) guarantees that active routes are planned sequentially based on updated system information. Each request is confirmed before receiving a new request concerning another active route. This ensures that each module is in the correct state to respond to an incoming request at any time. As passive route planning interrupts route reservation on the active route, multiple relocation routes are never planned at the same time. Therefore, one non-buffering sequencing module is generally sufficient within the network. The algorithmic operations for sequencing can continuously proceed preventing the system or parts of it from becoming deadlocked.

2) Resource Allocation: Resource allocation for physically transferring the corresponding TUs implies coordinating the utilization of the common system resources. According to [22], a resource deadlock requires the following four conditions to coexist:

(D1) *Mutual exclusion:* Processes claim exclusive control of their required resources.
(D2) *Hold and wait:* Processes hold allocated resources while waiting for further resources.
(D3) *No preemption:* Allocated resources are used to completion and released by the process holding them.
(D4) *Circular waiting:* There is a circular chain of processes, each of which is requesting a resource held by the next process in the chain.

To prevent deadlocks, at least one of these four conditions needs to be excluded [22]. In modular conveyor systems, conditions (D1) to (D3) are intrinsically satisfied [8]. Thus, we need to demonstrate that condition (D4) is prevented.

Referring to Table IV, the overall route of each TU through the system corresponds to a cohesive process starting at an input module and ending at an output module. To observe the necessary predecessor-successor dependencies, planning the overall process of a TU is decomposed by defining active and passive routes. For each buffered TU, an indefinite buffer reservation is created at the allocated resource for buffering according to reservation condition (R6). It is resolved entirely when initiating the active route from the buffer module to the output module. Thus, we obtain a finite overall process for each TU ending up at its output module. It represents a series of ascending logical time windows according to reservation conditions (R1) and (R2). This is used to demonstrate deadlock prevention for resource allocation similar to [14].

The circular wait condition (D4) for a resource deadlock implies a set of TUs holding a chain of conveyor modules within a closed loop. Assuming process P_1 is currently holding resource R_1, process P_2 resource R_2 and so on, within a circle of adjacent resources $R_1 \dots R_n$. Due to reservation conditions (R1) and (R3), satisfying circular waiting results in the following reservation dependencies for processes $P_1 \dots P_n$:

$$\begin{aligned}
T_{in}(P_1, R_1) < T_{out}(P_1, R_1) < T_{in}(P_n, R_1)\\
T_{in}(P_2, R_2) < T_{out}(P_2, R_2) < T_{in}(P_1, R_2)\\
\vdots\\
T_{in}(P_n, R_n) < T_{out}(P_n, R_n) < T_{in}(P_{n-1}, R_n)
\end{aligned} \tag{2}$$

Based on reservation condition (R2), we obtain for the circular chain of adjacent resources $R_1 \dots R_n$:

$$\begin{aligned}
T_{out}(P_1, R_1) &= T_{in}(P_1, R_2)\\
&\vdots\\
T_{out}(P_{n-1}, R_{n-1}) &= T_{in}(P_{n-1}, R_n)\\
T_{out}(P_n, R_n) &= T_{in}(P_n, R_1)
\end{aligned} \tag{3}$$

From this, it follows:

$$\begin{aligned}
T_{out}(P_1, R_1) < T_{out}(P_n, R_n) < T_{out}(P_{n-1}, R_{n-1})\\
< \quad \dots \quad < T_{out}(P_2, R_2) < T_{out}(P_1, R_1)
\end{aligned} \tag{4}$$

which is a contradiction. Thus, circular waiting is excluded by observing the defined reservation conditions such that deadlocks during resource allocation are prevented.

B. Livelocks

A livelock refers to a process indefinitely repeating the same execution sequence without progressing [23].

1) Algorithm Operations: Livelocks exist if modules send and receive messages in an endless loop. Due to sequential route planning (cf. Section IV-A1), we can consider each active route individually. The algorithmic operations terminate at its initiating module if path selection, route reservation, as well as all induced relocation operations terminate.

In path selection (cf. Section III-B2.b), modules which have already been explored are never selected for re-exploration such that messaging loops are generally excluded.
Route reservation requires all involved modules of active and passive routes to achieve a feasible reservation schedule according to the defined reservation conditions (R1) to (R6). At any time, there are no more than two actively negotiating modules (cf. Figures 6 and 7). All other involved modules

either already received a confirmation or are awaiting a response to their request to continue negotiations. Therefore, we show that each pair of adjacent modules always reaches an agreement on the logical time of transferring the corresponding TU.

Non-buffering modules only hold finite reservations within their local reservation tables. Thus, reservation requests to non-buffering modules are feasible at the latest when the new reservation is scheduled last at both negotiating modules. Repeating identical reservation requests is impossible as the next open time window, the requested module suggests, is later than the requested one. The initiating module of the active route does not hold any reservation scheduled later than that of its allocated TU such that postponing reservations is always possible. Reservation requests to buffer modules induce passive route planning for relocation before continuing reservation on the active route. This resolves the indefinite buffer reservation according to reservation condition (R6) such that it becomes a non-buffering module.

When negotiating passive routes, the non-buffering module sets the time at which it confirms the indefinite buffer reservation. At the requesting module of the passive route, reservation condition (R6) holds. Therefore, it can postpone the scheduled outgoing transport of the relocated buffered TU such that the relocation request is accepted.

2) Resource Allocation: In Section IV-A we showed that dispatching a TU from its input to its output module represents a coherent process composed of active and passive routes. Transport execution follows the set of logical time window reservations in ascending order and defines the order of resource utilization. As input and output modules are distinct, this never creates a closed loop, even if the same resources may be reused within the overall TU process.

C. Starvation

Starvation occurs if a process – not being deadlocked – needs to wait indefinitely as its requested resource is never assigned to it [24]. Thus, it will never terminate.

1) Algorithm Operations: Requesting modules await the response of the requested module. Assuming faultless communication, each request returns a valid response in finite time, so starvation is excluded within the algorithmic operations. In Section IV-B, we showed that route planning always terminates. Likewise, the authorization is granted to each requesting input module in finite time. Due to the limited system size and the predecessor-successor relations, an initiating module of an active route cannot be disadvantaged indefinitely when passing the authorization. After a limited number of requests, there will be no other module awaiting to be authorized.

2) Resource Allocation: Resource allocation for transport execution follows the total ordering of the reserved logical time windows. Thus, any process can use the requested resource in finite time. This excludes starvation within resource allocation.

V. Complexity Analysis

The complexity of our presented decentralized sequencing algorithm is measured by the number of messages N required to sequence a given set of TUs B using the set of conveyor modules M. As the specified routing operations apply to every TU, N linearly depends on $|B|$. Dispatching a TU from its input to its output module is organized via active routes. No more than two active routes are planned for each TU (cf. Figure 2). Therefore, $N = \gamma \cdot |B| \cdot N_a$, where N_a is the average number of messages required per active route as well as the induced passive routes and γ a non-negative constant ($\gamma \ll |M|$). We derive an upper bound $\hat{N}_a$ based on the given conveyor system indicating the maximum complexity of the presented sequencing algorithm. $\hat{N}_a$ results from the sum of the upper bounds of messages $\hat{N}_a^X$ required for every algorithmic sub-part:

- *Allocating buffer modules* $\hat{N}_a^B$: Each active route cannot require more than $|M|$ buffer modules for the planned and blocking TUs, respectively. For allocating a buffer module, no more than $|M|$ requests are necessary to identify an empty module for buffering. This implies
$$\hat{N}_a^B \leq \gamma^B \cdot |M|^2.$$
- *Initiating active routes* $\hat{N}_a^I$: Route initiation is triggered when the output module notifies the module currently assigned to a TU. This requires updating the output module every time the TU is relocated. Each active route incorporates less than $|M|$ interfering buffered TUs, each of which is relocated using less than $|M|$ passive routes. Thus,
$$\hat{N}_a^I \leq \gamma^I \cdot |M|^2.$$
- *Authorization procedure* $\hat{N}_a^A$: Acquiring the authorization for planning an active route requires less than $|M|$ requests to the authorized module. When the authorization is passed all other relevant modules are notified. This gives
$$\hat{N}_a^A \leq \gamma^A \cdot |M|^2.$$
- *Path selection* $\hat{N}_a^S$: Messaging for path selection is driven by the system size, i.e., $|M|$ as each module is notified no more than once for exploration. Thus,
$$\hat{N}_a^S \leq \gamma^S \cdot |M|.$$
- *Active route reservation* $\hat{N}_a^{R_a}$: Rejecting suggested logical time windows within active route reservation increases with the number of existing reservation entries per module. At any time, there cannot be more than $|M|$ TUs present within the system. Each of these induces less than $|M|$ relocation reservations per module when planning an active route. Thus, the number of reservation entries per module scales with $|M|^2$. Each active route incorporates less than $|M|$ negotiating adjacent modules. This results in
$$\hat{N}_a^{R_a} \leq \gamma^{R_a} \cdot |M|^3.$$
- *Passive route reservation* $\hat{N}_a^{R_p}$: Rejecting relocation requests within passive route reservation is finite as due to reservation condition (R6), only reservations scheduled after the last existing outgoing reservation are feasible.

As discussed for active route initiation, each active route causes less than $|M|^2$ relocation steps to be negotiated. This implies,

$$\hat{N}_a^{R_p} \leq \gamma^{R_p} \cdot |M|^2.$$

- *Transport execution* $\hat{N}_a^T$: For each active route, no more than $|M|^2$ reservations are scheduled, as its length as well as the length of its induced relocation routes is limited by $|M|$. Agreeing on a TU transfer between adjacent modules requires a constant number of messages as transport requests are only sent if the two adjacent modules are ready to execute the next reserved TU transfer. Therefore,

$$\hat{N}_a^T \leq \gamma^T \cdot |M|^2.$$

From all that follows the upper bound of the number of messages required for the given sequencing problem as

$$\hat{N} \leq \ \gamma \cdot |M|^3 \cdot |B|. \qquad (5)$$

Due to the decentralized system design, forwarding these messages is necessary via a series of adjacent modules in case the message sender and recipient are non-adjacent. This represents a linear scaling factor and is therefore incorporated in γ. As we always assume $|M|$ involved modules when determining $\hat{N}_a$, this indicates an upper bound at module level as well. The system offers $|M|$ control units for message processing – one installed at every module. Therefore, based on the overall algorithm complexity of $\mathcal{O}(n^3)$ according to (5), the complexity at the level of a single module decreases to $\mathcal{O}(n^2)$ scaling with system size n in terms of the number of installed modules $|M|$.

VI. Performance Analysis and Algorithm Parameterization

In simulation studies, we evaluate the performance of our decentralized sequencing algorithm. We take a 5×5 square arrangement of sequencing modules including five input and five output points as a reference network (cf. Figure 9(a)). Batches are randomly assigned to one of the output points with equal probability. Likewise, arriving TUs of these batches are randomly assigned to input points. The time for transferring a TU between two adjacent modules is denoted by t_{conv}, while t_{lift} represents the time for switching a module between its rectangular transportation modes. We focus on system throughput as key performance indicator.[5]

A. Impact of Conveying Speed and Relocation Penalty p_b

For isolating the effects of conveying speed, i.e., t_{conv}, in relation to varying values of relocation penalty p_b, we initially fix t_{lift} at 0 s. p_c is set to 0 accordingly. The results indicate that system throughput is directly proportional to the conveying speed of the modules (cf. Figure 10). If p_b is within the interval of $(0 \ldots 2]$ the throughput of the given setting is at its maximum. This applies regardless of the batch size k (cf. Figure 11).

[5]For all evaluations of Chapter VI, we start recording the performance indicators after a warm-up phase, which is determined using the MSER-5 method of [25]. The stopping criterion is calculated using an admissible deviation range of 1‰ from the mean system throughput during the total run time for the last 100s. The number of replications is based on a confidence interval with confidence level $\alpha = 5\%$ and error percentage $\epsilon = 0.01$, with a minimum of 10 and a maximum of 30 replications per parameter setting.

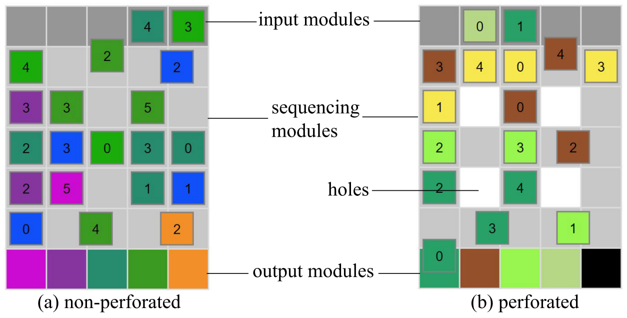

Fig. 9. Reference networks for performance analysis.

Increasing or decreasing system occupation is possible by varying the number of input $|I|$ and output points $|O|$ of a given network. Assuming $|I| \gg |O|$ creates a bottleneck at the unloading process such that system occupation increases. Likewise, when $|I| \ll |O|$, system occupation is reduced as the unloading capacity exceeds the loading capacity of the input points. Varying the reference network with 15 input or output points ceteris paribus yields the results shown in the charts of Figure 12.

This implies that the range of p_b which maximizes system throughput is not affected by varying batch sizes or system occupation. The minimum at $p_b = 0$ results from setting modules with interfering buffered TUs equal to empty modules. The optimal range is within the interval $0 < p_b \leq 2$. $p_b > 2$ causes a major reduction in throughput due to overweighting buffer modules. Within the given grid structure every shortest detour omitting a single buffer module induces two additional steps compared to the shortest route from start to destination. Thus, for $p_b > 2$, this detour is always preferred instead of the shortest path with an induced relocation. This causes interfering buffered TUs to be relocated only if there is no alternative path using detours. Analyzing the average path length of a TU confirms these statements (cf. Figure 13). Transforming the reference network according to Figure 9(b) increases detour lengths within the system. In this case, throughput reduction is observed at $p_b > 2$ as shown in Figure 14. Higher weights for buffer modules are necessary until detours are preferred instead of relocations.

Executing a detour increases solely the transport time of the TU on the active route, while an interfering TU remains positioned at its currently assigned buffer module. However, active and passive routes can be executed simultaneously. While the TU of the active route is moving towards a buffer module, the necessary passive routes for relocation are already processed. To optimize throughput, p_b needs to be set greater than 0 and smaller than the difference in the lengths of the shortest path and the shortest detour path between all modules within the network.

B. Interdependencies of Conveying Speed, Lift Time and Path Selection Parameters p_b and p_c

Due to the linear correlation of t_{conv} and system throughput observed within the previous evaluations, we fix t_{conv} at 1 s for all evaluations of this section. With $p_b = 1$, the effects

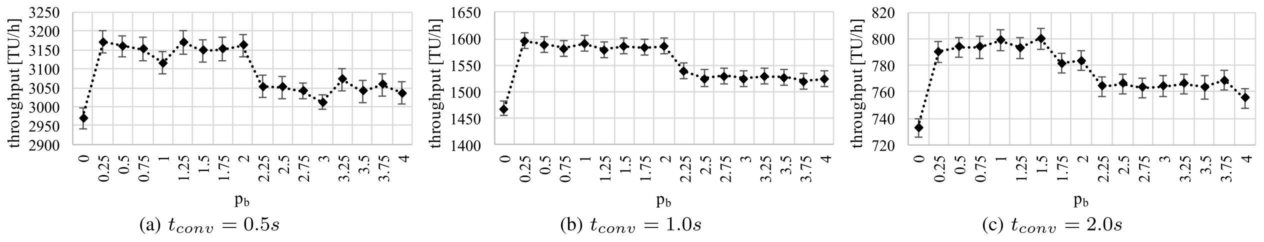

Fig. 10. Impact of p_b on throughput for selected values of conveying speed t_{conv} ($k = 5$).

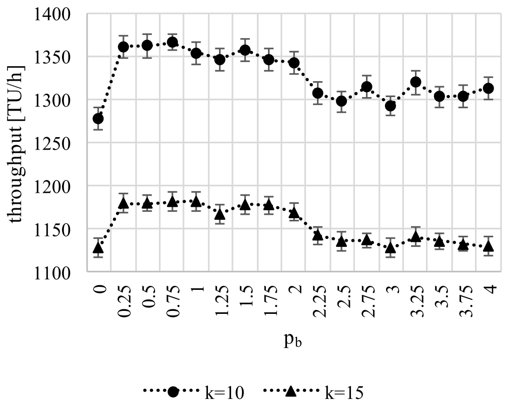

Fig. 11. Impact of p_b on throughput for selected values of batch size k ($t_{conv} = 1s$).

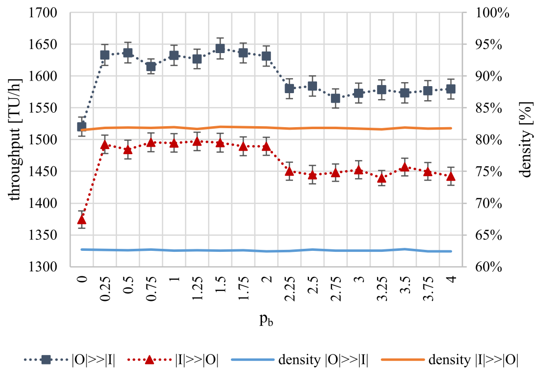

Fig. 12. Impact of p_b on throughput for different system occupations ($t_{conv} = 1s$, $k = 5$).

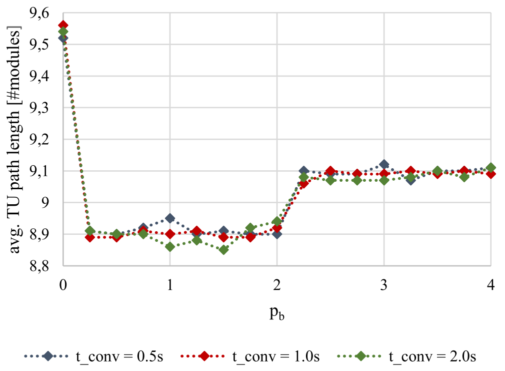

Fig. 13. Average TU path length for selected values of p_b ($k = 5$).

of t_{lift} and p_c arise as shown in Figure 15. The effects on throughput are equal for the given setting independent of t_{lift}, so even if $t_{lift} \gg t_{conv}$ holds. This results from the minimum number of directional changes on a particular path which is predetermined by the given network. With $p_c > 0$, unnecessary directional changes can be avoided. However, the

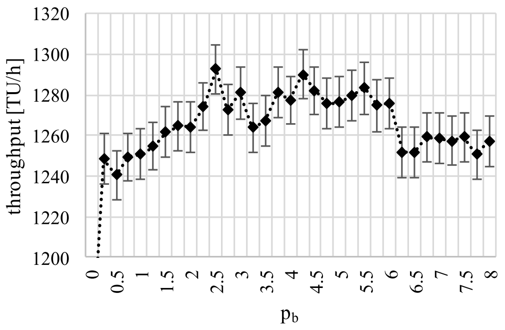

Fig. 14. Throughput results of the perforated 5×5 network for selected values of p_b ($t_{conv} = 1$, $k = 5$).

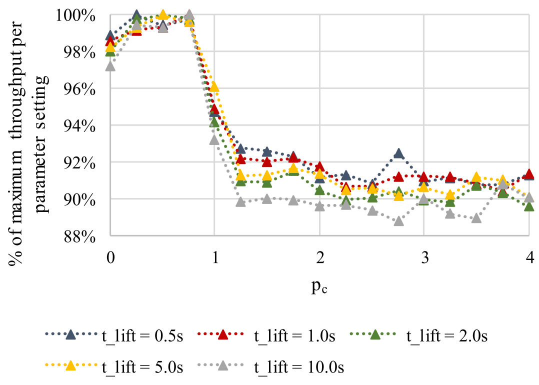

Fig. 15. Impact of p_c on throughput for selected values of lift time t_{lift} ($t_{conv} = 1s$, $k = 5$).

minimum number of direction changes is incurred regardless of t_{lift}.

As indicated in Section III-B2.b, p_c must not be set arbitrarily, as this will prevent path cost estimation from guaranteeing optimality. In this case, the overall optimal path cannot be derived from combining all optimal sub-paths, which is a necessary condition for optimality (cf. Bellman's principle of optimality). Thus, p_c must be chosen sufficiently small such that for all possible sub-paths $\tilde{p}$ and p^* no identical path costs are generated at any explored module, where $\tilde{p}$ is non-optimal and p^* is optimal. For this, $p_c < \epsilon$ must hold, which, however, also includes highly infrequent constellations. From the results shown in Figure 16, we conclude that $p_c < p_b$ represents a reasonable requirement as throughput is increased if $0 < p_c < p_b$.

C. Overall Results to Maximize System Throughput

From all the preceding numerical investigations, we derive that improving system throughput is achieved by:

- increasing conveying speed,
- decreasing batch sizes,
- increasing the number of output points,

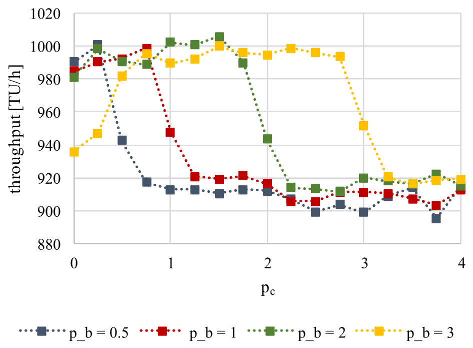

Fig. 16. Impact of p_c on throughput for selected values of p_b ($t_{conv} = 1s$, $k = 5$).

- creating non-perforated network structures, and
- setting $0 < p_c < p_b \leq 2$ for the algorithmic parameterization.

This parameterization is also stable concerning the given batch size, network structure, or resulting system occupation.

VII. Conclusion and Outlook

In this paper, we introduced a decentralized algorithm for sequencing transport units from multiple input to multiple output points in highest-density conveyor systems. The developed system is composed of identical right-angle-transfer conveyor modules, which are combined via Plug-&Play technology. Each of these modules decentrally executes the developed sequencing algorithm and takes decisions locally. The implementation is based on the concept of logical time. Combined with a decentralized authorization procedure for route planning, this prevents deadlocks, livelocks, and starvation as shown. The algorithmic complexity at module level is of order $\mathcal{O}(n^2)$ and scales with system size. Within our simulation studies, we deduce that high conveying speed, small batch sizes, increasing the number of output points, non-perforated network structures, and setting $0 < p_c < p_b \leq 2$ for the algorithmic parameterization are drivers for high system throughput.

Future work will be dedicated to investigations of more variable network configurations as the number of conveyor modules is crucial for the maximum batch size which can be sequenced. Analyzing the impact of the number and position of input and output points reveals profitable system setups in practical use cases. Furthermore, comparing the performance of our decentralized sequencing algorithm to centralized approaches allows to investigate the trade-off between optimality and flexibility, which we aim to address in further contributions.

References

[1] L. Monostori, P. Valckenaers, A. Dolgui, H. Panetto, M. Brdys, and B. C. Csáji, “Cooperative control in production and logistics,” *Annu. Rev. Control*, vol. 39, pp. 12–29, Jan. 2015.

[2] D. Trentesaux, “Distributed control of production systems,” *Eng. Appl. Artif. Intell.*, vol. 22, no. 7, pp. 971–978, Oct. 2009.

[3] K. R. Gue, “Very high density storage systems,” *IIE Trans.*, vol. 38, no. 1, pp. 79–90, Jan. 2006.

[4] Z. Shiller, “Off-line and on-line trajectory planning,” in *Motion and Operation Planning of Robotic Systems* (Mechanisms and Machine Science), G. Carbone and F. Gomez-Bravo, Eds. Cham, Switzerland: Springer, 2015, pp. 29–62.

[5] O. Uludağ, “GridPick: A high density puzzle based order picking system with decentralized control,” Ph.D. dissertation, Dept. Ind. Syst. Eng., Auburn Univ., Auburn, AL, USA, 2014.

[6] G. Hao, “GridHub: A grid-based, high-density material handling system,” Ph.D. dissertation, Dept. Ind. Eng., Univ. Louisville, Louisville, Kentucky, 2020.

[7] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. Boston, MA, USA: Pearson, 2015.

[8] S. Mayer and K. Furmans, “Deadlock prevention in a completely decentralized controlled materials flow systems,” *Logistics Res.*, vol. 2, nos. 3–4, pp. 147–158, Dec. 2010.

[9] T. Kruhn, M. Radosavac, N. Shchekutin, and L. Overmeyer, “Decentralized and dynamic routing for a cognitive conveyor,” in *Proc. IEEE/ASME Int. Conf. Adv. Intell. Mechatronics*, Jul. 2013, pp. 436–441.

[10] M. B. Firvida, H. Thamer, C. Uriarte, and M. Freitag, “Decentralized omnidirectional route planning and reservation for highly flexible material flow systems with small-scaled conveyor modules,” in *Proc. IEEE 23rd Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2018, pp. 685–692.

[11] S. Sohrt and L. Overmeyer, “Decentralized routing algorithm with physical time windows for modular conveyors,” *Logistics Res.*, vol. 13, no. 8, pp. 1–16, 2020.

[12] K. R. Gue, K. Furmans, Z. Seibold, and O. Uludag, “GridStore: A puzzle-based storage system with decentralized control,” *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 429–438, Apr. 2014.

[13] M. S. Ashgzari and K. R. Gue, “A puzzle-based material handling system for order picking,” *Int. Trans. Oper. Res.*, vol. 28, no. 4, pp. 1821–1846, Jul. 2021.

[14] Z. Seibold, K. Furmans, and K. R. Gue, “Using logical time to ensure liveness in material handling systems with decentralized control,” *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 1, pp. 545–552, Jan. 2022.

[15] R. Alahmad and K. Ishii, “A puzzle-based sequencing system for logistics items,” *Logistics*, vol. 5, no. 4, pp. 1–18, 2021.

[16] K. R. Gue, O. Uludağ, and K. Furmans, “A high-density system for carton sequencing,” in *Proc. Int. Mater. Handling Res. Colloq.*, 2012, pp. 1–14.

[17] K. Gue, “A high-density, puzzle-based system for rail-rail container transfers,” in *Proc. 14th IMHRC*, vol. 14, 2016, pp. 1–16.

[18] P. Sittivijan, “Modular warehouse control: Simultaneous rectilinear movement of multiple objects within a limited free space environment,” Ph.D. dissertation, Dept. Ind. Syst. Eng., North Carolina State Univ., Raleigh, NC, USA, 2015.

[19] C. Lieberoth-Leden and J. Fottner, “Deployment of an distributed strategic material flow control for automated material flow systems consisting of autonomous modules,” in *Proc. 15th IMHRC*, Savannah, GA, USA, 2018.

[20] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.

[21] L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[22] E. G. Coffman, M. J. Elphick, and A. Shoshani, “System deadlocks,” *ACM Comput. Surv.*, vol. 3, no. 2, pp. 67–78, Jun. 1971.

[23] G. J. Holzmann, *Design and Validation of Computer Protocols* (Prentice Hall Software Series). Englewood Cliffs, NJ, USA: Prentice-Hall, 1991.

[24] S. V. Ramesh, *Principles of Operating Systems*, 1st ed. New Delhi, Indi: Laxmi Publications, 2010.

[25] K. P. White, “An effective truncation heuristic for bias reduction in simulation output,” *Simulation*, vol. 69, no. 6, pp. 323–334, Dec. 1997.

Julia Fleischmann is currently a Research Associate with the Institute for Material Handling and Logistics, Karlsruhe Institute of Technology, Germany. Her research interests include the algorithms and control of modular and decentralized material handling systems.

Kai Furmans (Member, IEEE) is currently a Professor of Mechanical Engineering and the Head of the Institute for Material Handling and Logistics, Karlsruhe Institute of Technology, Germany. His research interests include automation and robotics in material handling as well as modeling of such systems.