Inventory Allocation in Robotic Mobile Fulfillment Systems

T. Lamballais D. Roy M.B.M. De Koster

January 9, 2017

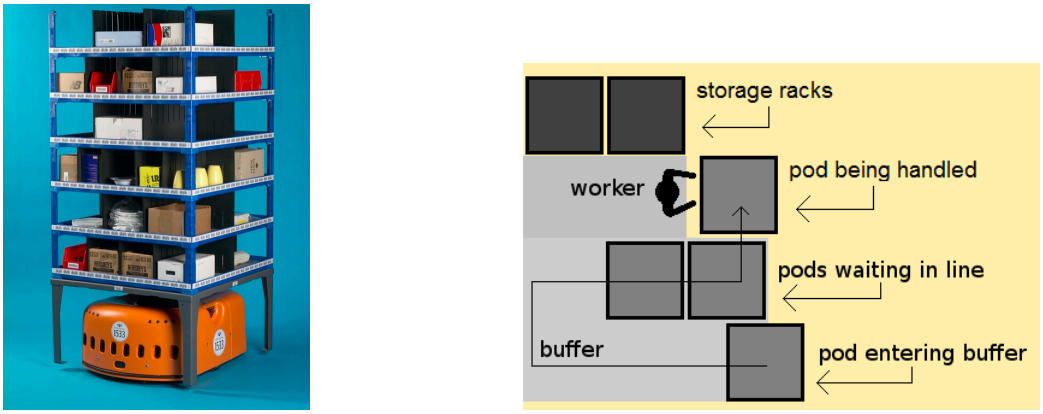
Abstract

A Robotic Mobile Fulfillment System is a recently developed automated, parts-to-picker material handling system. Robots can move storage shelves, also known as inventory pods, between the storage area and the workstations and can continually reposition them during operations. This paper shows how to optimize three key decision variables: (1) the number of pods per product (2) the ratio of the number of pick stations to replenishment stations, and (3) the replenishment level per pod. Our results show that throughput performance improves substantially when inventory is spread across multiple pods, when an optimum ratio between the number of pick stations to replenishment stations is achieved and when a pod is replenished before it is completely empty. This paper contributes methodologically by introducing a new type of Semi-Open Queueing Networks (SOQN): cross-class matching multi-class SOQN, by deriving necessary stability conditions, and by introducing a novel interpretation of the classes.

Introduction

E-commerce order fulfillment can be quite challenging for warehouses. Assortments tend to be large, and the number of daily orders and relative demand across products can fluctuate strongly. Robotic Mobile Fulfillment Systems (RMFS) are a new category of

automated storage and part-to-picker order picking systems developed specifically to fulfill e-commerce orders. These have been brought to the market by companies such as Amazon Robotics (previously known as Kiva Systems, (Business Wire)), Swisslog, Interlink, GreyOrange, Scallog, and Mobile Industrial Robots. Implementations so far suggest that picking rates may double compared to traditional picker-to-parts systems (Wulfraat). The core innovation of an RMFS are the robots that transport the pods, i.e., shelves containing products, to workstations. At a workstation, a pod queues until a worker either picks items from it, or replenishes items on the pod, see Figure 1a and 1b.



(a) Robot carrying a pod (Wurman and Enright, 2011)

(b) Top view of a workstation

Figure 1: Illustration of an inventory pod and close-up of a workstation

An RMFS is flexible in operations. Pods do not need to have a fixed position in the storage area, but can instead be repositioned continually throughout the day, see also Wurman and Enright (2011). Inventory can thus be positioned close to the workstations as needed. In addition, a product can be stored across multiple pods that can be positioned independently from each other. The travel times of the pods decrease if the inventory of a product is spread across multiple pods, because it becomes more likely that a suitable pod is located close to a workstation, and because different pods can be located close to different workstations. If the inventory on a pod drops below a threshold level, that pod is transported to a replenishment station to be fully replenished. In an e-commerce warehouse, one of the main performance metrics is the order throughput time, which improves if the travel times decrease. An interesting, unresolved aspect in the design

of an RMFS is the optimal number of pods over which a product's inventory should be spread to minimize order throughput time. Availability of products is another benefit of spreading inventory across multiple pods. If all inventory of a product is allocated to one pod, there is a risk of temporary unavailability of that product for picking when that pod is waiting for replenishment, or when it is in use for another order. Since in e-commerce environments most orders are single-line, temporary unavailability of that pod directly delays the orders for that SKU. If inventory is spread across multiple pods, it becomes less likely that an order for many units can be fulfilled with inventory from a single pod. In addition, if inventory is allocated across multiple pods, replenishment happens more frequently because the inventory per product on a pod will drop below the replenishment level sooner. This is inefficient as more trips are needed to replenish the same number of units. In both cases, orders for that product are delayed and order throughput time increases. In practice we observe that while some companies divide the inventory of a product over many locations (Amazon for example), others choose to keep products closer together (Timberland for example).

The impact of the trade-off also depends on the replenishment level. A higher replenishment level per product on a pod means that replenishment happens more frequently and may therefore cause additional robot travel time and additional queueing at the workstations. However, it also means that the average inventory on a pod is higher, causing the larger sized orders to wait less. The queueing at the workstations is also influenced by the ratio of the number of pick stations to replenishment stations. A higher replenishment level does not necessarily lead to more queueing if the number of replenishment stations is also higher. If the number of pods per SKU and the replenishment level are not jointly optimized, long and unnecessary delays may occur that can have a large impact on order throughput time. If the ratio of the number of pick stations to replenishment stations is not optimized, pick stations may have unacceptably low utilization while too much queueing occurs at the replenishment stations, or vice versa.

In view of these trade-offs, this paper studies how to minimize the order throughput time by optimizing three decision variables: (1) the number of pods per product, (2) the ratio of the number of pick stations to the number of replenishment stations, and

(3) the replenishment level per pod. We develop a novel queueing approach to jointly analyze the effect of these three variables. The inventory control policies are embedded in the queueing model's state transitions. The analytical model is developed to gain more insights into the system and to solve small instances, but unfortunately it cannot solve large, realistic instances, which are analyzed using simulation instead. Section 1 discusses the literature and motivates why a queueing model is suitable for analyzing these decision variables. To analyze the decisions, Section 2 develops a new type of queueing network, the cross-class matching multi-class Semi-Open Queueing Network (SOQN) and shows several counter-intuitive necessary stability conditions. Section 3 provides the results and insights. Section 4 concludes and presents a future outlook.

1 Literature

Some key features of the system are that robot travel times are stochastic, that queueing at the workstations comprises a substantial part of the order throughput time and that suitable pods have to be retrieved that can fulfill the orders. This implies that traditional inventory models such as (r, Q) or (s, S) policies cannot be used, as these cannot model some or all of these characteristics and their influence on the order throughput time. On the other hand, queueing networks have been used extensively for analyzing the performance of autonomous vehicle storage and retrieval systems (AVS/RS) and automated storage and retrieval systems (AS/RS). These networks can optimize key decision variables, because the low computation time allows evaluation of a large set of parameters. For example, Kuo et al. (2007) use queueing models to predict vehicle utilization and service, waiting and cycle times while varying the number of aisles, storage columns per aisle, storage tiers in the system, vehicles in the system, and the number of lifts in AVS/RS. Fukunari and Malmborg (2009) estimate the expected utilization of resources in an AVS/RS machine using a queueing model that incorporates both single and dual command cycles. Queueing networks can also incorporate the stochasticity of vehicle traveling and worker speed and can capture the resulting congestion effects (see Marchet et al. (2013), Roy et al. (2013), Roy et al. (2015a), Roy et al. (2015b), Roy et al. (2016) and Tappia et al. (2016)). Networks where orders arrive and depart from the system can

be divided into two broad categories: Open Queueing Networks (OQN) (Heragu et al. (2011)) and Semi-Open Queueing Networks (SOQN). SOQNs can capture the matching of different kinds of resources and can therefore include the time an order has to wait before being matched with a vehicle. This could be used to model the matching of orders and pods. For example, Roy et al. (2012) use a multi-class semi-open queueing network to analyze the performance impact of system parameters such as the number of vehicles and lifts, the depth-to-width ratio, and the number of zones. They also study the impact of operational decisions such as vehicle assignment rules on vehicle utilization and order cycle time. A disadvantage of SOQNs is the absence of product form solutions and exact solutions, only approximations exist. Ekren et al. (2014) apply the matrix-geometric method to analyze a SOQN for an AVS/RS, which results in quite accurate approximations for the metrics. Roy et al. (2012) develop a new decomposition technique to evaluate the system.

Lamballais et al. (2017) and Nigam et al. (2014) develop SOQN and CQN queueing networks to estimate the performance of picking operations in an RMFS. Lamballais et al. (2017) optimize the layout of an RMFS warehouse by estimating the expected order cycle time, workstation utilization, and robot utilization for a given layout and by determining the optimal dimensioning of the storage area and the optimal placement of the workstations. They do not consider replenishment but only look at pick operations.

However, a disadvantage of these SOQN models is that only orders and pods of the same type can be matched. In our system, if pods and orders are matched there may be some asymmetry: an order that requires a certain number of units can be fulfilled with a pod that contains that number of units *or more*. In addition, the optimal spreading of the inventory of a product across storage shelves has not been researched yet.

This paper is the first to study how inventory should be spread across pods, how the ratio of pick stations to replenishment stations should be set, and what the replenishment level should be, such that the order throughput time is minimized.

2 Model

This paper determines the optimal number of pods over which the inventory of a product should be spread, the optimal ratio of the number of pick stations to replenishment stations, and the optimal replenishment level of a pod. We construct a queueing network using two main ideas. The first idea is to use the various movements of a pod as queueing stations within the network. The second idea is to capture the number of units of a product as classes within the queueing network. The class of a pod signifies how many units of a product are left on the pod, and the class of an order indicates how many units the order requires. These two ideas are explained in detail in the sections below. In modeling the queueing network, we make the following assumptions to keep the model tractable. (1) Orders are assumed to be single-line orders. This is not too strong a simplification as RMFS were built especially for e-commerce order fulfillment. An order can still require multiple units of an SKU. In our context, a small order thus requires few units of an SKU and a large order requires many units of an SKU. (2) Orders arrive according to a Poisson process. (3) Pods are dedicated to an SKU. i.e., a pod can carry only one SKU at a time and is always replenished with the same SKU. The inventory of an SKU can be spread over multiple pods. (4) The maximum number of units in inventory per product is fixed, as is the number of workstations. (5) The pick or replenishment station at which a pod will be handled is chosen randomly. (6) Multiple pick stations can be modeled as one queueing station with multiple servers and similarly for multiple replenishment stations. (7) Pods are stored at any of the storage locations with equal probabilities; no distinction is made between fast and slow moving SKUs. (8) If multiple, equally suitable pods are available for order picking, the one that is nearest to the workstation will be fetched. Note that, if the inventory of a product is spread across more pods, travel times will become shorter on average, because it becomes more likely that a suitable pod is close to a pick station. This is elaborated in section 2.2. (9) The number of robots is not a constraint and does not delay any of the processes. (10) An order has to be matched with exactly one pod and "order splitting" is not allowed. This implies that if each of the pods carrying a certain SKU have some, but insufficient, units for an order, the order cannot be matched and needs to wait until one of the pods is replenished. (11) A pod can only be matched

with one order at a time and "order merging" is not allowed. This implies that if there are two orders that each require one unit and there is a single pod with two units, the pod will be matched with the orders sequentially.

2.1 Pod Movement

From the perspective of a pod, the following eight processes occur, see Figures 2 and 3. The pod (1) waits to be matched with an order, (2) waits for a robot to come to its storage location, (3) moves to the pick station, (4) queues for its turn at the pick stations and then has items picked from it. If its inventory is not below the replenishment level, (5) the pod returns to the storage area, otherwise it (6) moves to a replenishment station, (7) queues for its turn and is replenished and (8) returns to the storage area.

Each of these processes can be modeled as a queue, where the distribution of the travel times in a situation becomes the distribution of the service time of the corresponding queue. The pick stations are modeled as the servers of the queueing station corresponding to process (4) and the replenishment stations are modeled as the servers of the queueing station corresponding to process (7). The queueing network, which is shown in Figure 3, is a Semi-Open Queueing Network that captures the matching of an order to a pod. The numbers in Figure 2 and Figure 3 show which process corresponds with which queue and Table 1 shows the modeling approach for each of the decision variables. Here S is the number of SKUs, M^s is the number of pods of SKU s, r is the ratio of the number of pick stations to replenishment stations and ξ is the replenishment level, which is expressed as a percentage of the capacity of a pod, U.

2.2 Travel Times and Service Times

For steady state performance analysis, the process in Figure 2 repeats indefinitely and each of these eight processes can be translated one-to-one to a queue, as shown in Figure 3. The handling times at the pick and replenishment stations are assumed to follow an exponential distribution. The number of servers at queue (4) equals the number of pick stations in the RMFS and the number of servers at queue (7) equals the number of replenishment stations. The travel times are modeled as Infinite Server (IS) stations, since

Figure 2: Illustration of pod movement

Table 1: Modeling approach for each of the decision variables

Symbol	Decision variable	Modeling approach
$\sum_{s=1}^{S} M^s$	Total number of pods	Number of "tokens" circulating in the SOQN (see Buitenhek et al. (2000))
r	Ratio of the number of pick stations to replenishment stations	Ratio of the number of servers at the pick queue to the number of servers at the replenishment queue
ξ	Replenishment level	Pod classes in the SOQN

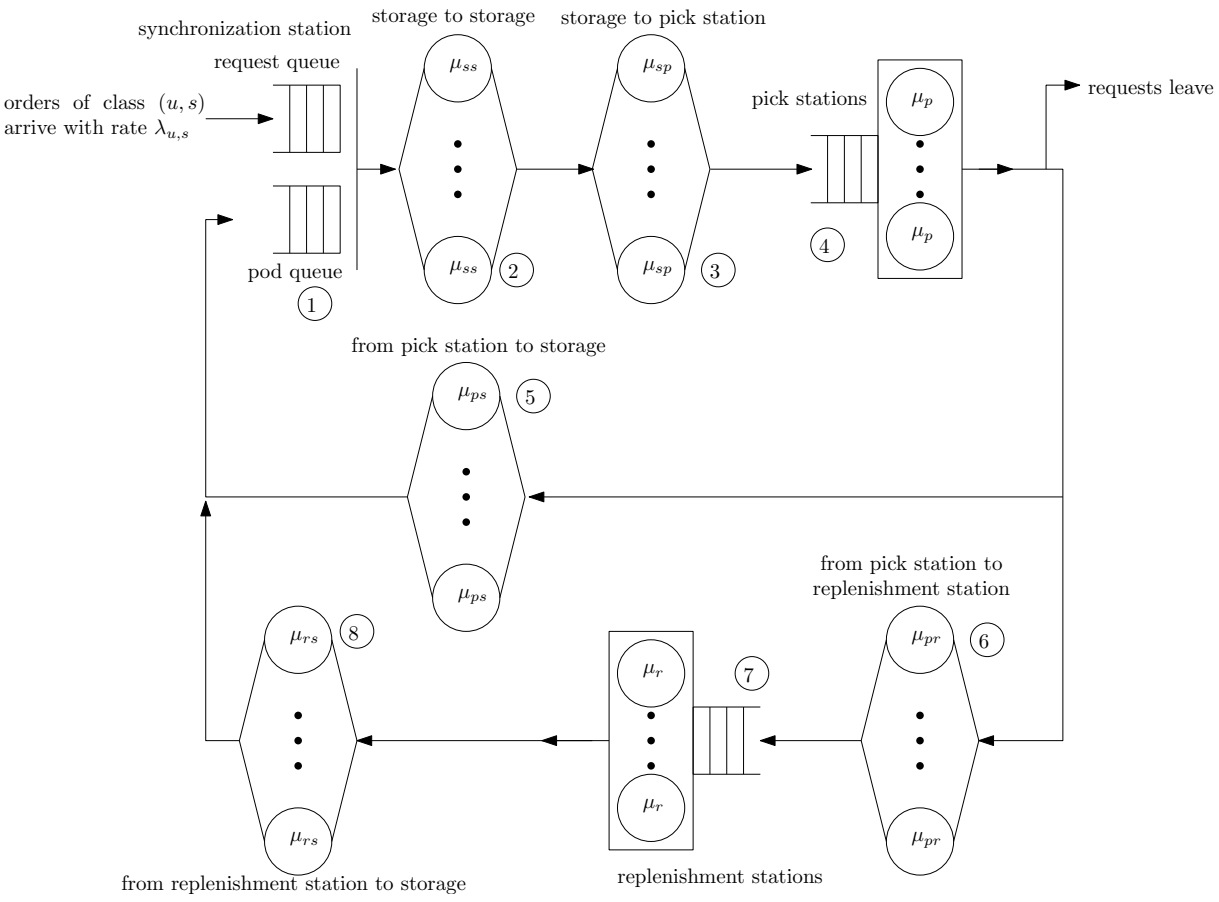


Figure 3: Queueing model of pod movements in the RMFS

pods can start traveling immediately and do not have to wait for other pods to finish. At these IS stations, the average service time corresponds to the average time needed for traveling. The aisles have a single travel direction to prevent deadlock and reduce aisle congestion. Unloaded robots can move underneath the pods and do not need to use the aisles. Movement through an aisle with a pod in the area between the workstations and storage area is also single directional, see also Figure 4.

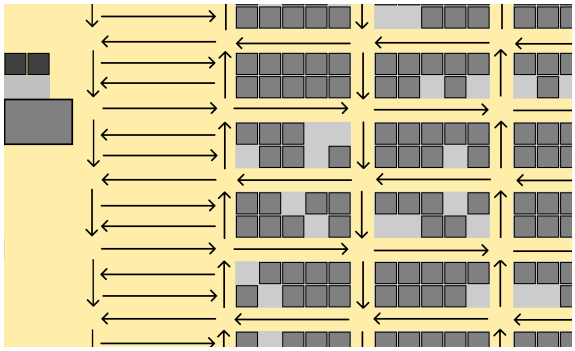


Figure 4: Example of travel directions in a part of the warehouse

This means that the travel distance between any two locations can be calculated exactly (see also Lamballais et al. (2017)). We make the following assumptions when calculating the travel times: (1) Robot velocity is constant. This can be an average velocity that takes acceleration and deceleration into account (see also Lamballais et al. (2017)). (2) Delays due to robot congestion or blocking in aisles, battery recharges, or robot downtime do not occur.

As mentioned earlier, we assume that storage and retrieval occur at a random location. For a single pod, the distribution of the travel time can be created by calculating the travel time from the current location to every possible destination and giving these equal probabilities of occuring. These probabilities can be adjusted to suit the application, for example, to include storage zones for products with different pick frequencies (see Lamballais et al. (2017)).

For multiple pods, the travel time distribution can be created as follows. Suppose that the cumulative distribution function (cdf) of the travel times of a single pod is given by $F(x)$. The travel times decrease if the inventory of a product is spread across multiple pods, because the pod nearest to a workstation can be chosen. The cdf of the travel times

of the nearest pod of SKU s is given by $\tilde{F}(x) = 1 - [1 - F(x)]^{M^s}$. For any IS station modeling travel times, the service time distribution has the cdf $\tilde{F}(x)$ corresponding to those travel times.

2.3 Inventory Levels

If every order line requires only one unit of a product, the queueing model can be solved using the methods in Buitenhek et al. (2000) and Bolch et al. (2006). If an order line requires more than one unit, the behavior of the queueing network becomes more complicated, because a pod with u remaining units cannot fulfill an order line that needs j units if $j > u$. It is therefore necessary to keep track of the number of units per product on each pod and the number of units required by each order. The queueing network achieves this by modeling the number of units remaining on a pod as the class of that pod and the number of units an order requires as the class of that order. In other words, a pod dedicated to SKU s with u units remaining has class (u, s). Since pods are dedicated to SKUs, the SKU class s of a pod cannot change. Similarly, orders for a product s that require u units have class (u, s) and arrive with a rate $\lambda_{u,s}$. Since orders are single-line orders, s simply denotes the product of the order. For a class (u, s), index u refers to the "inventory class", and index s refers to the "SKU class".

The replenishment level, denoted by ξ, is a percentage between 0% and 100%, the maximum number of units on a pod is denoted by U, so $0 \leq u \leq U$, and the replenishment point is ξU. Consider a pod that leaves a pick station. If the pod has inventory class $j, j > \xi U$, it returns to the storage area, but if it has inventory class $j \leq \xi U$, it moves to a replenishment station. In other words, the routing depends on the inventory class of the pod, and the routing probability to go from process (4) to process (6) in Figure 3 is either 0 or 1.

At the synchronization station (see process (1) in Figure 3), a pod of class (j, s) can be matched with an order of class (i, s) if $i \leq j$. We call this kind of matching "cross-class" matching. Due to this cross-class matching, both orders and pods with the same SKU class may be waiting at the synchronization station simultaneously. This happens if $j < i$ for all pods and all orders. If multiple orders are available, the pod is matched with the

order of the highest suitable class available. In other words, the pod is matched with the order that requires the most units, but no more units than the current pod inventory. At the pick station, the pod's inventory class is lowered after picking. If the pod belonged to class (j, s) and was matched with an order of class (i, s), then the pod's class after picking changes to class $(j - i, s)$, because $j - i$ is the number of units left after picking. Figure 5 illustrates these class switches for a situation where the maximum inventory level is six units and the replenishment level is zero.

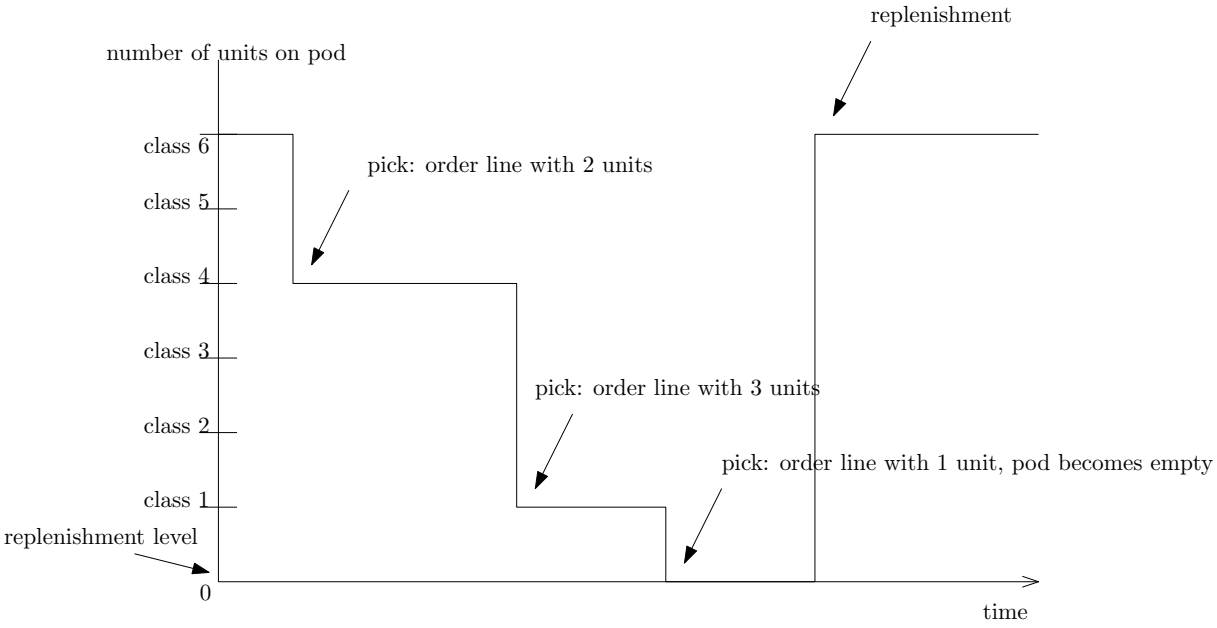


Figure 5: Illustration of a pod's class switching

Since pods are dedicated to a single SKU, a switch from a class $(u,\ s)$ to a class $(u',\ s')$ is not allowed for any $s \neq s'$.

2.4 Creating the Compact Queueing Model

SOQN models do not have a product form, although good approximation methods exist (Buitenhek et al. (2000), Jia (2005)). However, existing methods for analyzing multi-class SOQN models cannot be applied to the model in Figure 3, because these methods cannot analyze a network that uses cross-class matching. Therefore, we construct a novel Continuous Time Markov Chain (CTMC) to analyze the multi-class SOQN with cross-class matching. Since the size of the state space of the Markov Chain corresponding to the

queueing network in Figure 3 grows rapidly in the number of pods and classes, a (nearly) equivalent, compact queueing network is created which state space size grows less rapidly. The Markov Chain of this compact queueing network can then be used to analyze the system. To transform the queueing network in Figure 3 to the compact queueing network requires an additional, intermediate queueing network, which is shown in Figure 6.

The transformation works as follows. In the queueing network in Figure 3, no queueing occurs at the IS stations that model travel. Interaction of the robots via queueing only occurs at the pick and replenishment stations, and at the synchronization queue. The network can therefore be reduced to three queues, a synchronization queue, a pick queue, and a replenishment queue. This reduction is done by grouping the IS stations and the pick and replenishment queues into two groups as shown in Figure 6. Each group can then be transformed into an equivalent load-dependent queue using Norton's theorem (Bolch et al. (2006)). This results in the compact queuing network shown in Figure 7. To create the two groups, the IS station in Figure 3, process 5, needs to be changed because it cannot readily be incorporated into a group with the pick stations or into a group with the replenishment stations. However, in a layout where the pick stations and replenishment stations are next to each other, at the same side of the storage area, processes 5 and 8 modeling travel from pick stations to storage and travel from replenishment stations to storage will have about identical service time distributions. Process 5 can be directly after the pick stations, so in other words, even if the pod goes for replenishment, it first has to visit this IS station. However, by omitting process 8 in the other group of queues, the network consisting of two groups of queues still contains the same underlying processes. This results in two groups of queues as shown in Figure 6.

In the compact queueing model in Figure 7, the service rate at the picking queue with i pods in the queue is $\tilde{\mu}_p(i)$ and the service rate at the replenishment queue with i pods is $\tilde{\mu}_r(i)$. As the number of pick stations is equal to or larger than one, $\tilde{\mu}_p(1) \leq \tilde{\mu}_p(i)$ for $i > 1$. A similar statement holds for $\tilde{\mu}_r(i)$. To further reduce the size of the state space, we change the class switching mechanism. Class switching of the pods happens before rather than after the picking queue. This means that the information about which class of orders was matched with each pod does not have to be included in the state space of

the Markov Chain. This reduces the state space without affecting the results that can be obtained from the queueing network.

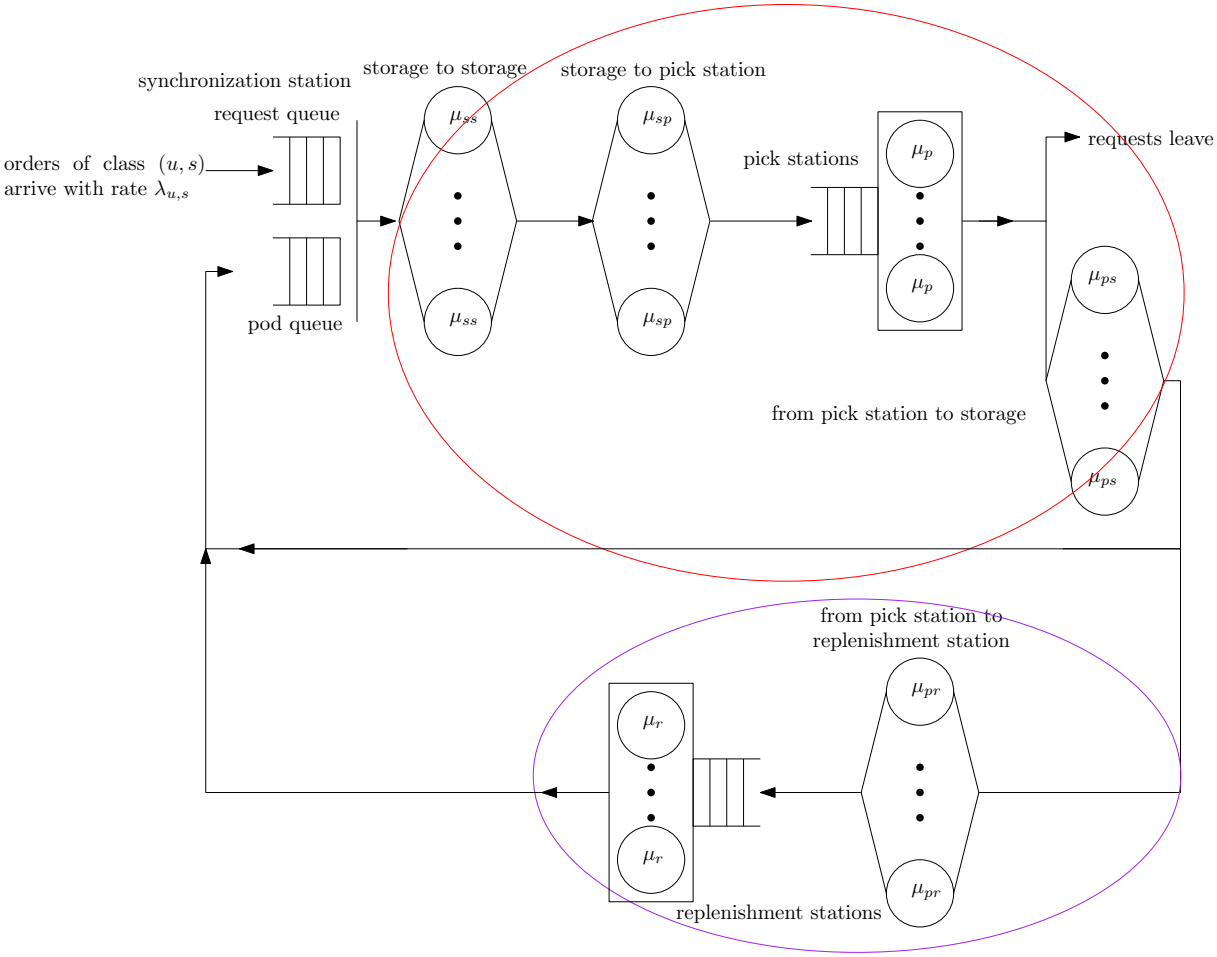


Figure 6: Intermediate queueing model

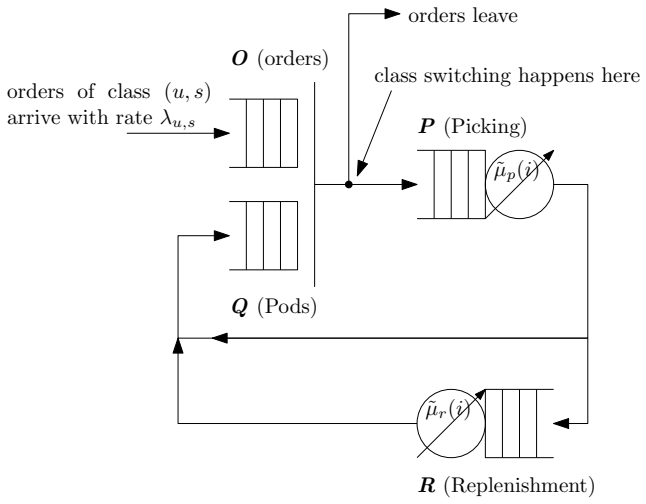


Figure 7: The compact model

2.5 Model States and Transitions

This section explains the states and transitions of the compact queueing model in Figure 7. Let S be the number of SKUs in the system and let U be the maximum number of units a pod can contain. Then the total number of inventory classes per SKU is $U + 1$, since a pod can also be empty (inventory class zero). The state space of the Markov Chain for the compact queueing network in Figure 7 consists of four parts: (1) A "waiting orders part", denoted by $\boldsymbol{O}$, which is a vector that contains the number of orders of each class that are waiting at the synchronization station. Since orders for zero units cannot arrive, the dimension of $\boldsymbol{O}$ equals $U \times S$. (2) A "waiting pods part", denoted by $\boldsymbol{Q}$, which is a vector that contains the number of pods of each class that are waiting at the synchronization station. Since pods with fewer units than replenishment level ξ go for replenishment and not to the synchronization station, the dimension of $\boldsymbol{Q}$ equals $(U - \lfloor \xi U \rfloor)S$. (3) A "picking part", denoted by $\boldsymbol{P}$, that contains the number of pods of each class that are waiting to be picked. As discussed earlier, to reduce the size of the state space class switching happens before this queue and hence the order leaves the system before this queue. This reduction trick does not change the average order throughput time, since the average service time for the pods at the picking queue is known and is the same as it would be for the orders. This also means that the dimension of $\boldsymbol{P}$ equals $U \times S$ rather than $(U + 1)S$, because the number of units of the order have already been subtracted from the class of the pod. Hence a pod at the pick queue cannot be full. Switching class before the pick queue avoids including information about the orders and order assignment to pods in the Markov Chain, which strongly reduces the size of the Markov Chain. (4) A "replenishment part", denoted by vector $\boldsymbol{R}$, which contains the number of pods of each class that are waiting to be replenished at the replenishment queue. Since only pods with $\lfloor \xi U \rfloor$ units or less go for replenishment, including pods that are empty, the dimension of $\boldsymbol{R}$ equals $(\lfloor \xi U \rfloor + 1)S$. The state can hence be described as $(\boldsymbol{O}, \boldsymbol{Q}, \boldsymbol{P}, \boldsymbol{R})$.

Let $\boldsymbol{O}^{\psi}_{u,s}$ denote the number of orders of class (u, s) waiting at the synchronization station in state ψ, $\boldsymbol{Q}^{\psi}_{u,s}$ the number of pods of class (u, s) waiting at the synchronization station in state ψ, $\boldsymbol{P}^{\psi}_{u,s}$ the number of pods of class (u, s) at the pick queue in state ψ, and $\boldsymbol{R}^{\psi}_{u,s}$ the number of pods of class (u, s) at the replenishment queue in state ψ. To limit the

number of states, the total number of orders allowed to wait at the synchronization station is limited to K, in the sense that for any state ψ it must hold that $\sum_{u,s} \boldsymbol{O}^{\psi}_{u,s} \leq K$. In a stable system, the number of orders waiting at the synchronization station should not grow too large. Therefore, as long as K is not too small, its effect on the performance metrics is negligible. The size of the state space is also limited by the number of pods per SKU, denoted by M^s, in the sense that $\sum_u \left(\boldsymbol{Q}^{\psi}_{u,s} + \boldsymbol{P}^{\psi}_{u,s} + \boldsymbol{R}^{\psi}_{u,s} \right) = M^s, \forall s, \psi$. Appendix A shows how to calculate the size of the state space for various parameters.

Three categories of transitions exist: (1) The arrival of an order, which either (1a) matches with a pod, (1b) queues at the synchronization station, or (1c) is rejected from the system. (2) The completion of picking a pod, after which either (2a) the pod matches with an order, (2b) the pod queues at the synchronization station, or (2c) the pod goes for replenishment. (3) The completion of replenishing a pod, after which either (3a) the pod matches with an order, or (3b) the pod queues at the synchronization station. In other words, for each category of transition, the entity involved (an order or a pod) either matches or does not match with its complement (pod or order). In addition to this, the transitions also include order rejection (1c) and replenishment (2c). Order rejection occurs when the arriving order cannot be matched with a pod and K orders are already queueing at the synchronization station.

If the inventory class of a pod falls at or below $\lfloor \xi U \rfloor$ after picking, the pod goes to the replenishment station to be replenished. After replenishment, the pod has class (U, s) so it can match with any waiting order for SKU s. Table 2 shows for each transition: the transition rate, the requirements that the state must meet for the transition to be possible, and how the state changes due to the transition. No information about the sequence of arrival is stored in the state, so at the pick and replenishment queue the queueing discipline is Random rather than FCFS. The number of states is finite and all the transitions are known, a generator matrix $\boldsymbol{\Gamma}$ and the state probabilities $\boldsymbol{\pi}$ can be calculated via $\boldsymbol{\pi\Gamma} = 0$ under the condition that $\boldsymbol{\pi 1} = 1$ (Bolch et al. (2006), page 69). Once the state probabilities have been calculated, they can be used to derive any metric of interest for the system.

As an example, suppose that there is just one SKU, so $s = 1$, and that $U = 2$ and $\xi = 0$,

Table 2: Transitions from a state ψ to a state σ

No	Transition rate	Requirements for transition of current state ψ to next state σ	Change of current state ψ to next state σ after transition
1a	$\sum_{u,s} \lambda_{u,s}$	$\exists j \vert j \geq u, \boldsymbol{Q}^{\psi}_{j,s} > 0$	$h = \underset{j}{\operatorname{argmin}}\{j \geq u, \boldsymbol{Q}^{\psi}_{j,s} > 0\}$
			$\boldsymbol{Q}^{\sigma}_{h,s} \coloneqq \boldsymbol{Q}^{\psi}_{h,s} - 1,\ \boldsymbol{P}^{\sigma}_{h-u,s} \coloneqq \boldsymbol{P}^{\psi}_{h-u,s} + 1$
1b	$\sum_{u,s} \lambda_{u,s}$	$\nexists j \vert j \geq u, \boldsymbol{Q}^{\psi}_{j,s} > 0$	$\boldsymbol{O}^{\sigma}_{u,s} \coloneqq \boldsymbol{O}^{\psi}_{u,s} + 1$
		$\sum_{u,s} \boldsymbol{O}^{\psi}_{u,s} < K$	
1c	$\sum_{u,s} \lambda_{u,s}$	$\nexists j \vert j \geq u, \boldsymbol{Q}^{\psi}_{j,s} > 0$	State remains the same, $\sigma = \psi$
		$\sum_{u} \boldsymbol{O}^{\psi}_{u,s} = K$	
2a	$\tilde{\mu}_p(\sum_{u,s} \boldsymbol{P}^{\psi}_{u,s})$	$\boldsymbol{P}^{\psi}_{u,s} > 0,\ u > \xi U$	$h = \underset{j}{\operatorname{argmax}}\{j \leq u, \boldsymbol{O}^{\psi}_{j,s} > 0\}$
		$\exists j \vert u \leq j, \boldsymbol{O}^{\psi}_{j,s} > 0$	$\boldsymbol{O}^{\sigma}_{h,s} \coloneqq \boldsymbol{O}^{\psi}_{h,s} - 1,\ \boldsymbol{P}^{\sigma}_{u,s} \coloneqq \boldsymbol{P}^{\psi}_{u,s} - 1$
			$\boldsymbol{P}^{\sigma}_{u-h,s} \coloneqq \boldsymbol{P}^{\psi}_{u-h,s} + 1$
2b	$\tilde{\mu}_p(\sum_{u,s} \boldsymbol{P}^{\psi}_{u,s})$	$\boldsymbol{P}^{\psi}_{u,s} > 0,\ u > \xi U$	$\boldsymbol{P}^{\sigma}_{u,s} \coloneqq \boldsymbol{P}^{\psi}_{u,s} - 1$
		$\nexists j \vert u \leq j, \boldsymbol{O}^{\psi}_{j,s} > 0$	$\boldsymbol{Q}^{\sigma}_{u,s} \coloneqq \boldsymbol{Q}^{\psi}_{u,s} + 1$
2c	$\tilde{\mu}_p(\sum_{u,s} \boldsymbol{P}^{\psi}_{u,s})$	$\boldsymbol{P}^{\psi}_{u,s} > 0,\ u \leq \xi U$	$\boldsymbol{P}^{\sigma}_{u,s} \coloneqq \boldsymbol{P}^{\psi}_{u,s} - 1,\ \boldsymbol{R}^{\sigma}_{u,s} \coloneqq \boldsymbol{R}^{\psi}_{u,s} + 1$
3a	$\tilde{\mu}_r(\sum_{u,s} \boldsymbol{R}^{\psi}_{u,s})$	$\boldsymbol{R}^{\psi}_{u,s} > 0$	$h = \underset{j}{\operatorname{argmax}}\{\boldsymbol{Q}^{\psi}_{j,s} > 0\}$
		$\exists j \vert \boldsymbol{O}^{\psi}_{j,s} > 0$	$\boldsymbol{O}^{\sigma}_{h,s} \coloneqq \boldsymbol{O}^{\psi}_{h,s} - 1,\ \boldsymbol{R}^{\sigma}_{u,s} \coloneqq \boldsymbol{R}^{\psi}_{u,s} - 1$
			$\boldsymbol{P}^{\sigma}_{U-h,s} \coloneqq \boldsymbol{P}^{\psi}_{U-h,s} + 1$
3b	$\tilde{\mu}_r(\sum_{u,s} \boldsymbol{R}^{\psi}_{u,s})$	$\boldsymbol{R}^{\psi}_{u,s} > 0$	$\boldsymbol{R}^{\sigma}_{u,s} \coloneqq \boldsymbol{R}^{\psi}_{u,s} - 1,\ \boldsymbol{Q}^{\sigma}_{U,s} \coloneqq \boldsymbol{Q}^{\psi}_{U,s} + 1$
		$\nexists j \vert \boldsymbol{O}^{\psi}_{j,s} > 0$	

then $\boldsymbol{O} = (\boldsymbol{O}_{1,1}, \boldsymbol{O}_{2,1})$, $\boldsymbol{Q} = (\boldsymbol{Q}_{1,1}, \boldsymbol{Q}_{2,1})$, $\boldsymbol{P} = (\boldsymbol{P}_{0,1}, \boldsymbol{P}_{1,1})$ and $\boldsymbol{R} = (\boldsymbol{R}_{0,1})$. Suppose furthermore that there are two pods in the system. One is waiting at the pick queue with one unit left, the other is at waiting the replenishment queue and an order is waiting for one unit at the synchronization station. The state would then be $\psi = ((1,0),\ (0,0),\ (0,1),\ (1))$ and Figure 8 shows the transitions into and out of this state. In Figure 8, $\lambda_{1,1}$ is the arrival rate of orders requiring just one unit, $\lambda_{2,1}$ is the arrival rate of orders requiring two units, $\tilde{\mu}_p(1)$ is the service rate at the pick station with one pod, $\tilde{\mu}_p(2)$ is the service rate with two pods, $\tilde{\mu}_r(1)$ is the service rate at the replenishment queue with one pod, and $\tilde{\mu}_r(2)$ is the service rate with two pods.

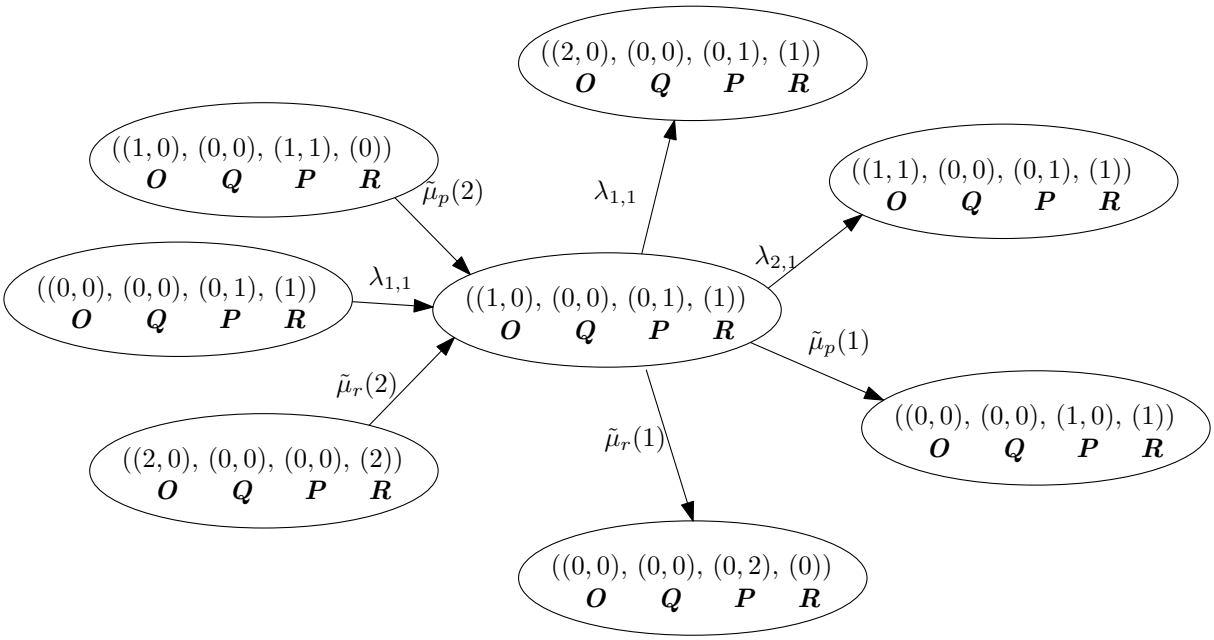


Figure 8: Example of transitions for $U = 2$ and $\xi = 0$

2.6 Aggregate Classes

A disadvantage of using classes to indicate the number of units is that the number of classes can grow large quickly. To mitigate this disadvantage, "aggregate classes" can be constructed for each SKU s. An aggregate class contains a range of m inventory classes of SKU s. An example would be to create three aggregate classes for each SKU, where aggregate class number 1 is labeled "low", number 2 is labeled "medium" and number 3 is labeled "high". If $U = 29$, and $m = 10$, then aggregate class low would contain inventory classes 0 to 9, aggregate class medium would contain inventory classes 10 to 19,

and aggregate class high would contain inventory classes 20 to 29. So a pod with 14 units would belong to aggregate class medium. The main difference that comes with using aggregate classes is that the way in which a pod changes class is no longer deterministic, but stochastic. As described earlier, when using inventory classes, the inventory class of the pod after picking equals the inventory class of the pod before picking minus the inventory class of the order that the pod is matched with, so in other words the class changes of the pod are completely deterministic. This is not the case when using aggregate classes. For example, if a pod with 14 units is matched with an order for 2 units, then the pod has 12 units left after picking. When using inventory classes this means a class change from class 14 to class 12, but when using aggregate classes the pod retains its aggregate class medium. In other words, a pod of aggregate class medium matched with an order of aggregate class low and after picking the pod had aggregate class medium. However, had the order required 6 units, the pod would have been left with 8 units after picking and would have changed from aggregate class medium to aggregate class low. In other words, a pod of aggregate class medium matched with an order of aggregate class low and after picking the pod had aggregate class low. As this example shows, when using aggregate classes the way in which a pod changes class is stochastic rather than fixed.

The main assumptions for adopting aggregate classes are the following. (1) Both pods and orders have aggregate classes, i.e., it is not possible that pods are designated with aggregate classes, whereas orders are designated with inventory classes. (2) When aggregate classes are used, the exact number of units needed by orders and left on pods is not tracked. (3) Every aggregate class contains the same number of m inventory classes. (4) The replenishment level is an inventory class and determines the probability of going for replenishment. In the example, if the replenishment level is 12, a pod of aggregate class low goes for replenishment with probability 1, and a pod of class medium goes for replenishment with probability 0.3. (5) Orders match with a pod of the same or higher aggregate class. If an order matches with a pod of the same aggregate class, we assume that the number of units that the order actually requires is less than or equal to the number of units that are actually left on the pod.

With inventory classes, if a pod of class a matches with an order of class b, the class of

the pod would be $a - b$ after picking. As it turns out, under the assumptions above a similar rule holds for aggregate classes, shown in Theorem 2.1.

Theorem 2.1. *If a pod of aggregate class number α, with $\alpha \geq 1$, matches with an order of aggregate class number β, with $1 \leq \beta \leq \alpha$, then with a probability p the pod will have aggregate class number $\alpha - \beta + 1$ after picking and with probability $1 - p$ the pod will have aggregate class number $\alpha - \beta$, where for $\alpha > \beta$ it holds that $p \to 0.5$ as $m \to \infty$ and for $\alpha = \beta$ it holds that $p \to 1$ as $m \to \infty$.*

Proof. Let an aggregate class number δ contain the inventory classes in the range $[\underline{n}_\delta, \overline{n}_\delta]$, $\underline{n}_\delta \equiv (\delta - 1)m$ and $\overline{n}_\delta \equiv \delta m - 1$, so $m = \overline{n}_\delta - \overline{n}_{\delta-1} = \overline{n}_\delta - (\underline{n}_\delta - 1) = \overline{n}_\delta - \underline{n}_\delta + 1$. For an inventory class d, $d \in [\underline{n}_\delta, \overline{n}_\delta]$, define $\hat{d}$ as $\hat{d} \equiv d - \underline{n}_\delta$, which implies that $0 \leq \hat{d} < m$. A pod of aggregate class number α will, in reality, have a units remaining, $a \in [\underline{n}_\alpha, \overline{n}_\alpha]$, and an order of aggregate class number β will, in reality, require b units, $b \in [\underline{n}_\beta, \overline{n}_\beta]$. After picking, the pod will have $a - b$ units left. Subsequently, equations (1) to (4) hold.

$$
\begin{aligned}
a - b &= \hat{a} + \underline{n}_\alpha - \hat{b} - \underline{n}_\beta && (1)\\
&= \hat{a} + (\alpha - 1)m - \hat{b} - (\beta - 1)m && (2)\\
&= \hat{a} - \hat{b} + ((\alpha - \beta + 1) - 1)m && (3)\\
&= \hat{a} - \hat{b} + \underline{n}_{(\alpha-\beta+1)} && (4)
\end{aligned}
$$

From equation (4) it can be seen that if $\hat{a} - \hat{b} < 0$, the pod will be in aggregate class number $\alpha - \beta$ as the number of its units falls below the lower boundary of class $\alpha - \beta + 1$. However, if $\hat{a} - \hat{b} \geq 0$, the number of units on the pod will be at or above the lower boundary of aggregate class number $\alpha - \beta + 1$, so the pod will be in aggregate class number $\alpha - \beta + 1$. If $\alpha > \beta$, then the probability that $\hat{a} > \hat{b}$ equals the probability that $\hat{a} < \hat{b}$ and as $m \to \infty$ the probability that $\hat{a} = \hat{b}$ will go to zero, hence $p \to 0.5$. If $\alpha = \beta$, then as stated earlier, it is assumed that $\hat{a} \geq \hat{b}$ and hence $p \to 1$ as $m \to \infty$ as the probability that $\hat{a} = \hat{b}$ goes to zero. □

2.7 Simulation and Validation

As can be seen from Table 9 in Appendix A, the number of states in the Markov Chain grows rapidly in the design parameters. Therefore we also develop a queueing network simulation, built in Java for analyzing large test cases. The network simulated is the compact queueing network depicted in Figure 7, where the warehouse layout in Figure 11 was used to create the travel times. To validate the results from the simulation, Table 3 shows the results for three instances. Each instance considers one SKU, one pick station, one replenishment station, and a replenishment level of zero for both the Markov Chain (MC) and the simulation (Sim), with ten runs each of ten weeks of working hours simulated, of which one third serves as the warm-up period. The order arrival rate equals 18 orders per hour per inventory class. The resulting confidence intervals were less than 1 percent of the averages and have therefore been omitted. For the first instance $U = 2$, $M^s = 2$, $K = 2$ resulting in 53 states in the MC, for the second instance $U = 3$, $M^s = 6$, $K = 2$ resulting in 2814 states in the MC, and for the third instance $U = 3$, $M^s = 6$, $K = 5$ resulting in 9324 states in the MC, see also Table 9. The pod utilization, denoted by ρ_{pod}, is the percentage of time that a pod on average is being carried by a robot. ρ_{pick} denotes the utilization of the pick stations, ρ_{repl} the utilization of the replenishment stations, t_{ot} the average order throughput time, L_s the average number of orders in the system, L_o the average number of orders waiting at the synchronization station, and L_p the average number of pods waiting at the synchronization station. These measures can be calculated exactly from the Markov Chain as shown in Equations (5) to (11). Here $\boldsymbol{\pi}_\psi$ denotes the steady state probability to be in state ψ and $\mathbb{1}_{[x]}$ denotes the indicator function, which equals 1 if condition x is true and equals 0 is condition x is false. The order throughput time is calculated as the average number of orders in the system (L_s) divided by the total arrival rate. As no more than K orders can be present at the synchronization station, orders arriving at the system may be rejected, which means that the actual arrival rate will be lower than $\sum_{u,s} \lambda_{u,s}$. This is taken into account via the indicator function in Equation (11).

$$\rho_{\text{pod}} = \sum_{\psi} \left(\boldsymbol{\pi}_{\psi} \times \frac{\sum_{u,s} \boldsymbol{P}^{\psi}_{u,s} + \boldsymbol{R}^{\psi}_{u,s}}{\sum_{s} M^{s}} \right) \tag{5}$$

$$\rho_{\text{pick}} = \sum_{\psi} \boldsymbol{\pi}_{\psi} \mathbb{1}_{\left[\sum_{u,s} \boldsymbol{P}^{\psi}_{u,s} > 0\right]} \tag{6}$$

$$\rho_{\text{repl}} = \sum_{\psi} \boldsymbol{\pi}_{\psi} \mathbb{1}_{\left[\sum_{u,s} \boldsymbol{R}^{\psi}_{u,s} > 0\right]} \tag{7}$$

$$L_{s} = \sum_{\psi} \boldsymbol{\pi}_{\psi} \sum_{u,s} (\boldsymbol{O}^{\psi}_{u,s} + \boldsymbol{P}^{\psi}_{u,s}) \tag{8}$$

$$L_{o} = \sum_{\psi} \boldsymbol{\pi}_{\psi} \sum_{u,s} \boldsymbol{O}^{\psi}_{u,s} \tag{9}$$

$$L_{p} = \sum_{\psi} \boldsymbol{\pi}_{\psi} \sum_{u,s} \boldsymbol{Q}^{\psi}_{u,s} \tag{10}$$

$$t_{ot} = \frac{L_{s}}{\sum_{\psi} \boldsymbol{\pi}_{\psi} \mathbb{1}_{\left[\sum_{u,s} \boldsymbol{O}^{\psi}_{u,s} < K\right]} \sum_{u,s} \lambda_{u,s}} \tag{11}$$

Table 3 shows that the outcomes of the simulations agree with the results from the exact calculations based on the Markov Chain. Table 4 shows the validation for using the aggregate classes. The results all come from simulation and show two experiments. In the first experiment, each SKU is spread over two pods, whereas in the second experiment each SKU is spread over six pods. In both experiments, the simulations with aggregate classes, indicated with "Agg.", had four aggregate classes. For the simulation with normal inventory classes, case (a) has 20 inventory classes, case (b) 40 inventory classes and case (c) 80 inventory classes. As can be seen, the results for the cases with aggregate classes are close to the results for the cases with normal inventory classes. In other words, using the aggregate classes does not reduce the accuracy of the performance estimates while simultaneously complexity is greatly reduced since far fewer classes are needed to model the system.

Table 3: Validation of the simulation with the Markov Chain

Metric	MC 1	Sim 1	MC 2	Sim 2	MC 3	Sim 3
$\rho_{\text{pod}}(\%)$	74.1	74.1	41.0	41.2	42.6	42.8
$\rho_{\text{pick}}(\%)$	70.3	70.0	81.8	81.2	82.9	82.2
$\rho_{\text{repl}}(\%)$	46.1	45.9	56.7	56.6	58.2	58.0
$t_{ot}(s)$	225.6	226.7	158.7	157.0	238.0	233.4
L_s(# orders)	1.736	1.744	2.116	2.106	3.291	3.237
L_o(# orders)	0.795	0.805	0.478	0.460	1.592	1.535
L_p(# pods)	0.517	0.519	3.542	3.527	3.443	3.434

Table 4: Validation of the use of aggregate class via simulation

Metric	Agg.1	1a	1b	1c	Agg.2	2a	2b	2c
$\rho_{\text{pod}}(\%)$	74.5	75.2	74.4	74.1	24.9	24.9	24.7	24.7
$\rho_{\text{pick}}(\%)$	70.3	70.8	70.6	70.6	62.0	62.0	62.1	62.1
$\rho_{\text{repl}}(\%)$	43.5	44.1	43.1	42.8	40.1	40.3	39.8	39.5
$t_{ot}(s)$	358.7	379.6	376.7	370.9	122.6	122.8	122.3	122.7
$L_s(\#$ orders)	2.867	3.037	3.012	2.961	0.982	0.981	0.981	0.984
$L_o(\#$ orders)	1.893	2.058	2.033	1.983	0.002	0.002	0.002	0.002
$L_p(\#$ pods)	0.510	0.496	0.512	0.518	4.506	4.505	4.516	4.517

2.8 Necessary Stability Conditions

Finding order arrival rates for which the queueing network is stable is much more interesting for cross-class matching multi-class SOQNs than for other types of SOQNs, and leads to some surprising results. For example, suppose that a pod can carry a maximum of 6 units of an SKU, the replenishment level is zero, and orders only arrive for either 1 or 5 units. Then the system is unstable if the arrival rate for orders of 1 unit is lower than the arrival rate for orders of 5 units, because each time an order of 5 units is matched to a pod, an order of 1 unit must also be matched to that pod before it goes to replenishment. In other words, the system can become unstable if the arrival rates of certain inventory classes become *too low*, which is an uncommon condition. There are three necessary stability conditions that the system must satisfy for each SKU separately, otherwise it will become unstable. These are the *replenishment level reachability* condition, the *combinatorial matching* condition, and the *maximum capacity* condition, which

are explained below. Fulfilling these conditions does not mean that the system will be stable, but fulfilling the sufficient condition mentioned at the end of this section does.

2.8.1 Condition 1: Replenishment Level Reachability

As an example, suppose that all inventory for a certain SKU is put on only one pod, $U = 6$ and $\lambda_{u,s} = 0, \forall u, s$ with the exception that $\lambda_{3,s} > 0$ and $\lambda_{4,s} > 0$ and that $\xi = 0\%$ so that a pod only goes for replenishment if it has zero units remaining. The system is unstable, because after a full pod synchronizes with an order of inventory class 4, the pod will change to an inventory class 2 and can no longer match with an order. It will then never reach the replenishment level and never replenish to inventory class 6.

The replenishment level reachability condition checks whether a pod can be trapped at an inventory level where it can no longer match with an order and at the same time cannot go to replenishment. This also means that for this condition the number of pods per SKU does not matter. Let H be the set of u for which $\lambda_{u,s} > 0$, then the feasible arrival rates for an SKU s must be in the set Λ_s^1:

$$\Lambda_s^1 = \left\{ \lambda_{u,s} \mid \exists n \in \mathbb{N}\,, \quad 0 \leq U - \sum_{u \in H} nu \leq \lfloor \xi U \rfloor \right\} \tag{12}$$

2.8.2 Condition 2: Combinatorial Matching

Suppose that the necessary stability condition 1 of Equation (12) is met. A counterintuitive feature of the necessary stability conditions for a cross-class matching multi-class SOQN is that there can also be lower bounds to the arrival rate of certain classes. An example would be if $\xi = 0$ and $\lambda_{u,s} = 0$ except $\lambda_{1,s} > 0$ and $\lambda_{U-1,s} > 0$ and $\lambda_{1,s} < \lambda_{U-1,s}$. Each time an order of class $U-1$ is matched with the pod, the order before or after must belong to inventory class 1 so that the pod can go for replenishment. This means that if fewer class 1 orders than class $U-1$ orders arrive, class $U-1$ orders cannot all be matched and will build up to infinity. Hence in this case $\lambda_{U-1,s}$ is a strict lower bound for $\lambda_{1,s}$.

More generally, the combinatorial matching condition essentially looks at the order arrival rates relative of each other. Matching orders of certain sizes to a pod may put constraints

on which orders can be matched to that same pod in between replenishments. In the example, matching a pod with an order of size $U-1$ means that the pod must be matched with an order of size 1 before or after, otherwise it cannot go for replenishment. Certain order sizes can only be matched to a pod in combination(s) with orders of other sizes, which constrains the arrival rates. This means that for this condition, the number of pods per SKU does not matter.

If $\frac{U}{u}$ is an integer, then orders of inventory class u alone can empty the pod and can therefore always bring the pod below the replenishment level ξ. If $\frac{U}{u}$ is not an integer, an additional condition is needed to ascertain that the pod can be brought below ξ. Define the set G as all u for which $\frac{U}{u} \notin \mathbb{N}$, then the feasible arrival rates must be in the set Λ_s^2.

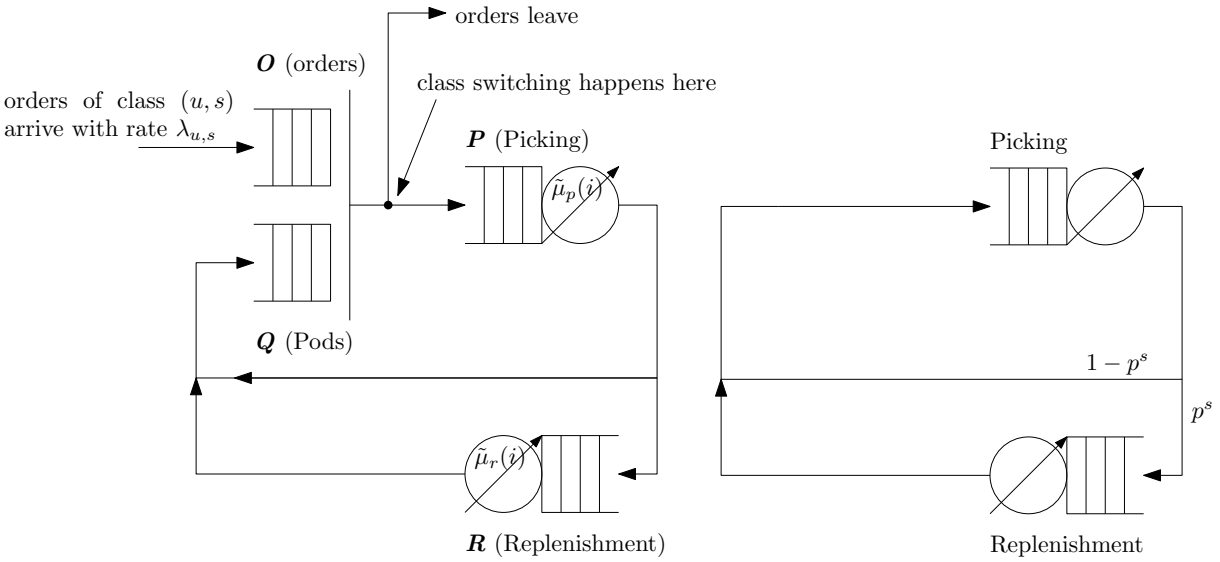
$$\Lambda_s^2 = \Bigg\{ \lambda_{u,s} | \exists w_{g,u} \in \mathbb{N},\, U - \lfloor \xi U \rfloor \leq g + \sum_{u=1}^{U} (u \times w_{g,u}) \leq U \; \forall g \in G, \\ \sum_{g \in G} w_{g,u} \lambda_{g,s} \leq \lambda_{u,s} \; \forall u \notin G \Bigg\} \tag{13}$$

Here $U - \lfloor \xi U \rfloor \leq g + \sum_{u=1}^{U} (u \times w_{g,u}) \leq U$ means that it must be possible to match an order of inventory class g to a pod together with an integer number ($w_{g,u}$) of orders of other inventory classes u, such that the amount $g + \sum_{u=1}^{U} (u \times w_{g,u})$ is equal to or larger than the number $U - \lfloor \xi U \rfloor$ required to bring the pod below the replenishment level, but equal to or less than the maximum number U on a pod. As an example, assume again that $\xi = 0$, $\lambda_{u,s} = 0$ except $\lambda_{1,s} > 0$ and $\lambda_{U-1,s} > 0$, and $\lambda_{1,s} < \lambda_{U-1,s}$. Then $G = \{U-1\}$ and $w_{U-1,u} = 1$, so that the equation (13) becomes $U \leq U - 1 + 1 \times 1 \leq U$, $1 \times \lambda_{U-1,s} \leq \lambda_{1,s}$

2.8.3 Condition 3: Maximum Capacity

The maximum capacity necessary condition examines the number of trips to the pick stations and replenishment stations that are made given the order arrival rates. If the amount of time needed to do picking and replenishment per order times the arrival rate exceeds one, then the system cannot be stable. Here we disregard the time a pod is queueing at the stations, which means that this condition provides an upper bound on the order arrival rates. The system operates at maximum capacity when the pods do not have to wait for an order, but instead immediately synchronize with an order as

soon as they reach the synchronization station. For each SKU s, this corresponds to a CQN network where the pods go to replenishment with a probability p^s. Figure 9a shows the compact queueing network, a SOQN model, and Figure 9b the CQN model that corresponds with the compact model working at full capacity, i.e., the orders are always waiting.



(a) SOQN model, multiple inventory classes (b) CQN model, single class

Figure 9: The compact queueing model and the corresponding CQN model

Suppose that ϕ^s is the average number of times per time unit that the pod goes for replenishment for SKU s. For each SKU s, $\lambda_{u,s}$ orders of inventory class u arrive per time unit that each need u units. In a stable system, all these orders across all inventory classes are fulfilled and the units needed to fill these orders come from replenishment. This implies that the total number of units needed per time unit, divided by the number of units put on the pod during replenishment, equals ϕ^s, see also equation (14). Suppose furthermore that θ^s is the average number of times per SKU s and per time unit that the pod goes through the CQN. A pod carries one order at any point in time. In a stable system, it needs to synchronize with every order and for every order the pod carries it goes through the network once. θ^s can then be calculated as shown in equation (15). Assuming that the replenishment level is zero, probability p^s can be calculated as shown in equation (16). If $p^s \notin [0, 1]$, the network is not stable.

$$\phi^s = \frac{\sum_{u=1}^{U} u\lambda_{u,s}}{M^s U} \qquad \forall s \tag{14}$$

$$\theta^s = \sum_{u=1}^{U} \frac{\lambda_{u,s}}{M^s} \qquad \forall s \tag{15}$$

$$p^s = \frac{\phi^s}{\theta^s} \qquad \forall s \tag{16}$$

The pick and replenishment queue are load-dependent queues and for both queues it holds that the largest service time happens when one pod is at the queue, since then only one workstation is operating at a time. The service time is denoted by $\tilde{\mu}_p^{-1}(i)$ for the pick queue and by $\tilde{\mu}_r^{-1}(i)$ for the replenishment queue. As described earlier in section 2.4, $\tilde{\mu}_p(1) \leq \tilde{\mu}_p(i)$ for $i > 1$, therefore using $\tilde{\mu}_p^{-1}(1)$ as the time picking takes provides an upper bound on the picking time needed. $\tilde{\mu}_r^{-1}(1)$ gives an upper bound for replenishment. Per time unit and per SKU s, the pod goes to the pick queue θ^s number of times, where each time it will spend on average at most $\tilde{\mu}_p^{-1}(1)$ time units. This means that per time unit, a pod is, on average, busy with picking at most $\theta^s \tilde{\mu}_p^{-1}(1)$ time. Furthermore, per time unit the pod goes to the replenishment queue ϕ^s number of times, where each time it spends, on average, at most $\tilde{\mu}_r^{-1}(1)$ time units. This means that per time unit, a pod will on average be busy with replenishment at most $\phi^s \tilde{\mu}_r^{-1}(1)$ time. Let t_{busy} denote the percentage of a time unit that a pod is used. In a stable system, this all needs to fit within one time unit, therefore the order arrival rates needs to be in the set Λ_s^3 as depicted in equation (18).

$$t_{\text{busy}} = \theta^s \tilde{\mu}_p^{-1}(1) + \phi^s \tilde{\mu}_r^{-1}(1)) \tag{17}$$

$$\Lambda_s^3 = \{\lambda_{u,s} | p^s \in [0,1], t_{\text{busy}} \in [0,1]\} \tag{18}$$

2.8.4 Sufficiency Condition

For the necessary stability conditions it holds that $\Lambda_s^2 \subset \Lambda_s^1$ but neither Λ_s^1 and Λ_s^3, nor Λ_s^2 and Λ_s^3 are subsets of each other. Figure 10 shows an example of these sets for a situation with one pod, $U = 6$, $\xi = 0\%$ and $\lambda_{u,s} = 0$ except $\lambda_{1,s} > 0$ and $\lambda_{5,s} > 0$.

The three necessary stability conditions do not guarantee that the system will be stable.

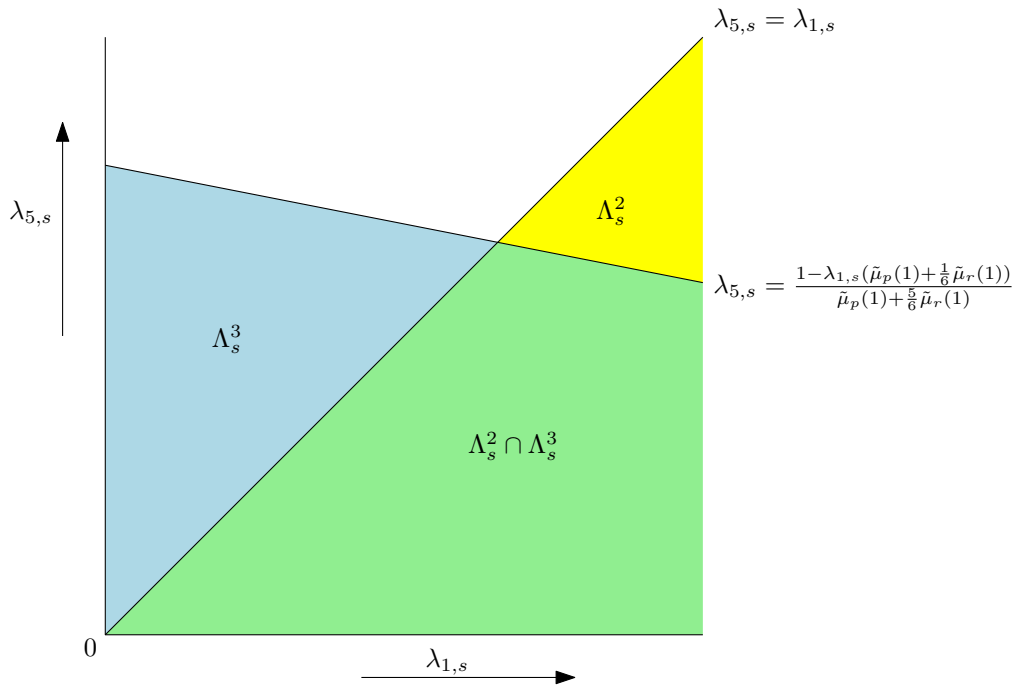


Figure 10: Example of arrival rates that meet the combinatorial matching and the maximum capacity necessary stability conditions

To see this, consider a system with one SKU and one pod, $U = 3, \xi = 0\%$ and $\lambda_{1,s} = \lambda_{2,s} > 0, \lambda_{3,s} = 0$, where the arrival rates are such that the maximum capacity necessary stability condition is met, i.e., they are in Λ_s^3. These arrival rates would be in Λ_s^1, as the replenishment level can clearly be reached, and would be in Λ_s^2, for example, by setting $w_{g,u} = 1 \; \forall g, u$. This system would be stable if every time the pod comes back from replenishment, it would first match with an order of inventory class 1 and later with an order of inventory class 2 or vice versa. However, after replenishment the pod could match twice with an order of inventory class 1, after which it will need to match again with an order of inventory class 1, else it cannot go to replenishment. Each time this happens, three orders of inventory class 2 cannot be matched. This means that the stock of inventory class 2 orders builds up and thus that the system is unstable, despite the fact that the necessary stability conditions are met.

The problem is that the pod will not reach the replenishment level *sufficiently* often to prevent the build-up of orders in the queue. A sufficient condition to prevent this situation would be that $\lambda_{1,s}$ orders for each SKU s is very large relative to the arrival rates of the

other inventory classes. If there are enough orders for 1 unit, the pod will reach the replenishment level sufficiently often. Theorem 2.2 shows a sufficient condition.

Theorem 2.2. *Assuming that* $\lambda_{u,s} \in \Lambda_s^2 \cap \Lambda_s^3$ *for* $1 \leq u \leq U$, $1 \leq s \leq S$, *it holds that* $\lambda_{1,s} \geq \sum_{u=2}^{U} ((U - \lfloor \xi U \rfloor) - u)\, \lambda_{u,s}$ *is a sufficient condition for stability of the network*

Proof. Let C be a collection of orders that such $\sum_{o_{u,s} \in C} u = U - (\lfloor \xi U \rfloor + 1)$, with $o_{u,s}$ denoting an order of class (u, s). Here C can have multiple orders of the same class (u, s), but they all have to be for the same SKU s. If a pod of class (U, s) would be sequentially matched with all orders in C and only with the orders in C, the pod would attain class $(\lfloor \xi U \rfloor + 1, s)$. Let $\mathfrak{C}$ be the set of all such collections C that meet this condition. Let η_C be the rate at which a pod of class (U, s) is matched with a sequence of orders that contains orders of the same classes in the same number as C. $\lambda_{1,s} \geq \sum_{C \in \mathfrak{C}} \eta_C$ is not a sufficient condition, because inventory class $(1, s)$ may also appear one or multiple times in a collection C. For an inventory class u, let C_u be the set of all collections C that contain at least one order of inventory class u. Define $L_u = \sum_{C \in C_u} \eta_C$, then L_u is the rate with which a pod attains class $(\lfloor \xi U \rfloor + 1, s)$ due to a collection C that contained an order of inventory class u. Besides an order of inventory class u, it could happen that this collection C consists only of orders of inventory class $(1, s)$. If a pod is matched with an order of inventory class u, it is certain that the pod will reach the replenishment level if it is also matched with $(U - \lfloor \xi U \rfloor) - u$ number of orders of inventory class $(1, s)$. Therefore, the pod will reach the replenishment level *sufficiently* often if $\lambda_{1,s} \geq \sum_u ((U - \lfloor \xi U \rfloor) - u)\, L_u$. As $\lambda_{u,s} \geq L_u$, $\lambda_{1,s} \geq \sum_{u=2}^{U} ((U - \lfloor \xi U \rfloor) - u)\, \lambda_{u,s}$ is a sufficient condition. □

3 Results

Figure 11 shows the layout and Table 5 shows the parameters used to generate the results in this section. In Figure 11 on the left side, there are six workstations, separated by a traffic corridor from the storage area. In the storage area, stored pods are shown as dark grey squares and empty storage locations as light grey squares. The time needed for pod lifting and storing, the robot speed, and the distribution of the pick and replenishment time are based on Lamballais et al. (2017). However, the Markov Chain would become

far too large when using these parameters. Therefore we simulated the queueing network describing the pod movements as depicted in the Figure 3 to generate the results in this section. The simulation time was 604800 seconds per run, with 10 runs and a warm-up time of 201600 seconds. The width of the 95% confidence intervals were typically about 1% of the average values of the metrics.

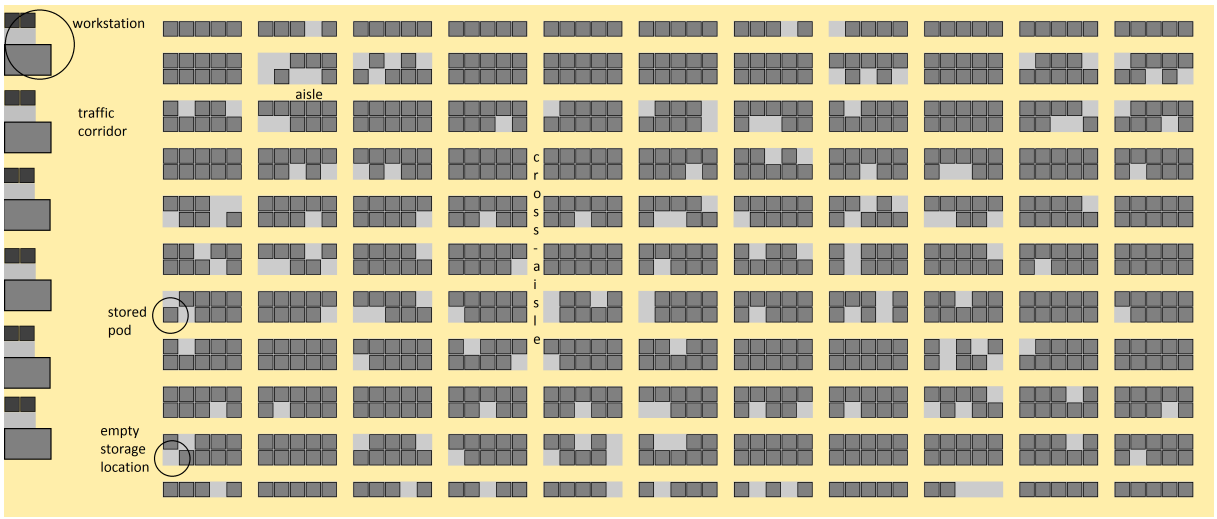


Figure 11: Top view of the warehouse layout used for the experiments

Table 5: Parameters used in all the experiments

Parameter	Value	Parameter	Value
Number of aisles	10	Number of cross-aisles	10
Number of storage locations	1100	Number of workstations	6
Time for pod lifting and storing	1 (s)	Robot speed	1.3 (m/s)
Pick time & replenishment time (Identically distributed)	Exponentially distributed, mean is 8 (s)	$\sum_u \lambda_{u,s}$	2 (h^{-1})
K	1200	Number of SKUs S	100
Layout width	98.8 meters	Layout length	41.6 meters

To investigate the optimal number of pods to use for inventory, we keep the maximum inventory in the experiments at 36 units per SKU and vary the number of pods per SKU, M^s, which can be 1, 2, 3, 4 or 6, while keeping the layout fixed. The number of SKUs S is 100 and the SKUs are identical. Taken together, this means that the total number of pods that are actively used in the system varies from 100 to 600. The total number of pods in the system is always 935, which equals 85% of the number of storage locations. The average travel time to bring a pod to a pick station strongly decreases in the number of pods. For the northernmost workstation, the average travel time is 43.4, 34.1, 29.5, 26.6, and 23.0 seconds for 1, 2, 3, 4, and 6 pods respectively. In the last case with 6 pods, $U = 6$, which means that the largest order that can be matched with a pod has 6 units. Therefore in order to keep the experiments comparable, $\lambda_{u,s} > 0$ for $1 \leq u \leq 6$ and

$\lambda_{u,s} = 0$ for $u > 6$ across all experiments. To meet the sufficient condition for stability, the arrival rates are $\lambda_{2,s} = \lambda_{3,s} = \lambda_{4,s} = \lambda_{5,s} = \lambda_{6,s}$ and $\lambda_{1,s} = 10 \times \lambda_{2,s}$. Having far more orders arrive for 1 unit than for multiple units seems reasonable, as the system is typically used in an e-commerce environment.

To investigate the optimal ratio of the number of pick stations to replenishment stations, we analyze 5 cases. Let the ratio of the number of pick stations to replenishment stations be denoted by r, where $r = (i, j)$ means that in Figure 11 the upper i workstations are pick stations and the j lower workstations are replenishment stations. The 5 cases we consider are $r = (1, 5)$, $r = (2, 4)$, $r = (3, 3)$, $r = (4, 2)$ and $r = (5, 1)$.

Finally, to investigate the optimal replenishment level per pod, we study the following cases: (1) a pod is not replenished unless it is empty, $\xi = 0\%$, (2) a pod is not replenished unless it is at least half empty, $\xi =$50%, and (3) a pod is replenished every time after it has visited a pick station, $\xi = 100\%$.

From Table 6 we can conclude that spreading inventory across as many pods as possible (6 in the experiments) appears to consistently result in the lowest order throughput times. If the inventory of an SKU is contained on a single pod, the replenishment level has a much larger influence on the order throughput time than if the inventory is spread across multiple pods. The order throughput time is about one and a half to two times longer when a single pod is used. This is due to a longer average travel time for a single pod from storage to a pick station and due to a higher probability that no pod is available due to replenishment. We can also conclude that in most cases the optimal ratio is $r = (4, 2)$. In addition, it is clear that $r = (1, 5)$ performs much worse than the other ratios, and the optimal one in terms of lowest order throughput time depends both on M^s and ξ. Lastly, we can conclude that the optimal replenishment level is $\xi = 50\%$ in the majority of experiments. Setting the replenishment level as $\xi = 0\%$ leads to the highest order throughput times in all experiments except 1. The utilization of the replenishment stations is typically very low, unless $\xi = 100\%$, so if the pod goes to replenishment after every pick. Appendix B shows the same results but for equal order arrival rates across inventory classes. From Table 10 we can conclude that the system is not stable if $\xi = 0\%$, even though the necessary stability conditions are met. This happens because insufficient

Table 6: Experiment results, t_{ot} in seconds and the utilizations in percentages

		$\xi = 0\%$				$\xi = 50\%$				$\xi = 100\%$			
M^s , U	r	t_{ot}	ρ_{pod}	ρ_{pick}	ρ_{repl}	t_{ot}	ρ_{pod}	ρ_{pick}	ρ_{repl}	t_{ot}	ρ_{pod}	ρ_{pick}	ρ_{repl}
1 , 36	(1, 5)	414.4	11.1	80.7	0.9	168.3	11.2	80.5	1.7	171.0	13.2	80.4	16.1
1 , 36	(2, 4)	330.2	7.4	40.5	1.1	92.7	7.5	40.3	2.1	97.2	9.7	40.5	20.1
1 , 36	(3, 3)	332.7	7.2	26.9	1.5	89.3	7.3	26.9	2.8	93.8	9.6	27.0	27.0
1 , 36	(4, 2)	328.5	7.1	20.1	2.2	88.6	7.3	20.3	4.2	93.2	9.7	20.1	40.2
1 , 36	(5, 1)	323.7	7.1	16.1	4.5	88.9	7.3	16.2	8.4	103.9	13.1	16.1	80.0
2 , 18	(1, 5)	172.0	5.7	80.9	1.8	148.3	5.6	80.1	3.2	146.3	6.5	80.4	16.1
2 , 18	(2, 4)	89.7	3.5	40.3	2.2	77.0	3.6	40.4	3.9	77.2	4.6	40.4	20.1
2 , 18	(3, 3)	85.6	3.4	26.8	3.0	73.7	3.5	26.7	5.2	74.0	4.5	26.9	27.0
2 , 18	(4, 2)	85.4	3.4	20.1	4.4	73.0	3.5	20.1	7.8	73.3	4.6	20.2	40.4
2 , 18	(5, 1)	86.3	3.4	16.0	8.9	73.3	3.5	16.1	15.9	74.0	6.5	16.0	80.4
3 , 12	(1, 5)	147.3	3.6	80.5	2.7	141.4	3.7	80.4	4.5	147.2	4.4	81.2	16.2
3 , 12	(2, 4)	74.5	2.3	40.1	3.4	72.2	2.4	40.3	5.6	72.2	3.0	40.4	20.2
3 , 12	(3, 3)	72.2	2.2	26.9	4.5	69.1	2.3	26.7	7.6	69.2	3.0	26.9	27.0
3 , 12	(4, 2)	71.5	2.2	20.2	6.7	68.6	2.3	20.3	11.4	68.6	3.0	20.1	40.1
3 , 12	(5, 1)	70.7	2.2	16.1	13.4	68.7	2.4	16.1	22.8	68.8	4.4	16.2	81.0
4 , 9	(1, 5)	158.2	2.8	80.7	3.6	140.0	2.8	80.4	5.4	140.3	3.2	80.7	16.2
4 , 9	(2, 4)	87.5	1.7	40.2	4.5	69.5	1.8	40.4	6.7	69.3	2.2	40.3	20.2
4 , 9	(3, 3)	81.1	1.7	26.8	5.9	66.4	1.7	26.9	8.9	66.3	2.2	26.8	26.7
4 , 9	(4, 2)	83.4	1.7	20.1	9.0	65.7	1.7	20.2	13.4	65.7	2.2	20.3	40.4
4 , 9	(5, 1)	84.8	1.7	16.1	17.9	66.0	1.8	16.1	26.5	65.9	3.2	16.0	80.2
6 , 6	(1, 5)	139.9	1.8	80.5	5.4	137.0	1.9	80.4	7.8	144.5	2.2	80.8	16.2
6 , 6	(2, 4)	66.0	1.2	40.0	6.7	65.9	1.2	40.3	9.7	65.9	1.4	40.3	20.2
6 , 6	(3, 3)	63.3	1.1	26.9	8.9	63.1	1.2	27.0	13.0	62.9	1.4	26.8	26.8
6 , 6	(4, 2)	62.6	1.1	20.1	13.4	62.4	1.2	20.1	19.4	62.5	1.4	20.1	40.1
6 , 6	(5, 1)	63.1	1.1	16.1	26.7	62.6	1.2	16.1	39.0	62.6	2.1	16.1	80.4

orders of inventory class 1 arrive, which means that the pod stays in inventory class 1 far too long and does not go for replenishment sufficiently often.

4 Conclusions and Future Work

The results show that each of the decision variables can be optimized. The order throughput time appears lowest when inventory is spread across as many pods as possible, when the ratio of the number of pick stations to replenishment stations is 4 to 2, and when the replenishment level is 50%. The relationship between order throughput time and the ratio of the number of pick stations to replenishment stations has a skewed U shape. Moreover, the replenishment level has a strong influence on the utilization of the replenishment stations, which can become very low. Tables 7 and 8 summarize some of the additional insights.

Table 7: Managerial insights regarding stability

Stability of pick operations	Small arrival rate of orders of 1 unit	Large arrival rate of orders of 1 unit
Replenish when pod is empty	Not stable	Stable
Replenish when still some units left on pod	Stable	Stable

Table 8: Managerial insights regarding decision variables

Interaction decision variables	Inventory of a product on a single pod	Inventory of a product on multiple pods
Ratio of the number of pick stations to replenishment stations	Choosing the wrong ratio can increase order throughput time by about 25%	Choosing the wrong ratio can increase order throughput time by about 100%
Replenishment level	Large impact on order throughput time, best to replenish a pod before it becomes empty	Small impact on order throughput time, best to replenish a pod before it becomes empty

This paper focused on several important tactical decisions, but there are many promising directions for future research focusing on operational decisions. For example, an RMFS is flexible in capacity as robots can be added quickly and workstations can be opened and closed as needed. The system can thus dynamically increase and decrease the amount of resources used and how much of the resources to dedicate to different types of activities, something that has not yet been researched. Another interesting feature is that the system's decisions can be decentralized to a high degree. Kiva Systems decentralized robot movement and collision detection already in the earliest implementation, but other elements such as route planning, task scheduling, and resource allocation can also be decentralized (see Wurman et al. (2008)). No research has yet been conducted on the interplay between the algorithms and policies for each of these elements.

Appendix A Size of the State Space

This appendix explains how to calculate the total number of states possible in the Markov Chain. The explanation here assumes there is only one SKU, so $S = 1$, but for $S > 1$, the number of states is just the number of states when $S = 1$ multiplied by S. The total number of states is constrained by the maximum number of orders that can wait at the synchronization station and by the number of pods. There can be at most K orders at the synchronization station. Let the total number of possible ways to distribute k orders across the order classes and for all k be denoted by N_O^C. These orders can be of any of U classes (the dimension of $\boldsymbol{O}$ equals U as explained above), so the possible number of combinations is $\binom{U+k-1}{k}$ as this is similar to choosing with repetition k places out of a possible number of U places. Then N_O^C is given by Equation (19). Let N_P^C denote the number of possible ways that M^s pods can be distributed across $\boldsymbol{P}$, $\boldsymbol{Q}$ and $\boldsymbol{R}$. This is again similar to choosing with repetition, and Equation (20) shows how to calculate N_P^C.

$$N_O^C = \sum_{k=0}^{K} \binom{U+k-1}{k} \tag{19}$$

$$N_P^C = \binom{2U+M^s}{M^s} \tag{20}$$

It is not possible to combine every instance of an order part with every instance of the other three parts, because whenever a pod of inventory class u is at the synchronization station, it is not possible to also have orders of inventory class u or lower there. In other words, $\boldsymbol{O}$ constrains the number of inventory classes u in $\boldsymbol{P}$ for which $\boldsymbol{P}_{u,s}$ can be bigger than zero. Let $N_{j,s}^Y$ be the number of combinations in $\boldsymbol{O}$ for which $\boldsymbol{O}_{u,s} = 0, u \leq j$ and $\boldsymbol{O}_{j+1,s} > 0$, so $N_{j,s}^Y = \{\boldsymbol{O} | \boldsymbol{O}_{u,s} = 0, 0 \leq u \leq j, \boldsymbol{O}_{j+1,s} > 0\}, j = 0, \ldots, U-1$. Suppose that there are k orders at the synchronization station, with $k = 1, \ldots, K$. For $N_{j,s}^Y$, this means there can be h orders of class $j+1$ and $k-h$ at classes $j+2$ to $U-1$. Let $N_{j,s,k,h}^Y$ denote all possible situations where k orders are waiting at the synchronization station, with no orders of class j or lower and h orders of class $j+1$, then:

$$N^Y_{j,s,k,h} = \begin{cases} \binom{U+k-h-j-2}{k-h} & \text{if } U+k-h-j-2 \geq 0 \\ 1 & \text{(no classes } > j \text{ left to distribute orders to)} \end{cases} \tag{21}$$

$$N^Y_{j,s} = \sum_{k=1}^{K} \sum_{h=1}^{k} N^Y_{j,s,k,h} \qquad j = 0, \ldots, U-1 \tag{22}$$

In the above, $j \leq U-1$, because $N^Y_{U,s} = 0$ and therefore does not follow the structure and formula above.

Let $N^F_{j,s}$ be the number of combinations in $(\boldsymbol{Q}, \boldsymbol{P}, \boldsymbol{R})$ for which $\boldsymbol{Q}_{u,s} > 0, u \leq j$ and let N^Z be the total number of states in the Markov Chain. Assume that the replenishment level is zero, so pods are only replenished when they are empty.

$$N^F_{j,s} = \binom{2U + M^s - j}{M^s}, \qquad j = 0, \ldots, U \tag{23}$$

$$N^Z = \sum_{s=0}^{S} \sum_{j=0}^{U} N^Y_{j,s} N^F_{j,s} \tag{24}$$

Table 9 shows the number of states for various combinations of K, U and M^s. It seems that if K increases by 10, N^Z roughly doubles. Another pattern that becomes apparent is that the higher U is, the more rapid N^Z grows in M^s.

Table 9: Number of states N^Z, $S = 1$

N^Z	$K=2$	$K=5$	$K=10$	$K=20$	$K=30$	$K=40$
$U=2$, $M^s=2$	53	155	445	1475	3105	5335
$U=2$, $M^s=6$	462	1050	2590	7770	15750	26530
$U=3$, $M^s=2$	155	708	3263	18998	57233	127968
$U=3$, $M^s=6$	2814	9324	35574	183624	529074	1155924
$U=4$, $M^s=2$	360	2430	17290	171810	732105	2116675
$U=4$, $M^s=6$	12105	56313	322773	2802393	11368863	32049183
$U=5$, $M^s=2$	721	6882	73073	1206580	7190337	26714094
$U=5$, $M^s=6$	41283	261030	2186327	31200246	176430265	638136884

Appendix B Results with Equal Order Arrival Rates

Table 10 shows the results for equal order arrival rates. For ξ =50% and ξ =100%, the results are nearly identical to the results in section 3, but for ξ =0%, the results are unstable. The order arrival rates meet all the necessary stability conditions, but these are not sufficient and it turns out that the order throughput time simply becomes longer if the simulation time is larger. Let T_K be the percentage of time the system spends in a state where the number of orders at the synchronization stations equals K. If T_K exceeds 1%, we consider the system unstable and do not show results. The reason the system is unstable is that the pods in the system remain in state $(1, s)$ too often. This happens, because of the following. As $U = 6$, if an order of class $(5, s)$ arrives and matches with a full pod, the pod goes to state $(1, s)$ and can only be matched with an order of class $(1, s)$. This also happens if the pod is matched with an order of class $(2, s)$ and an order of class $(3, s)$ or alternatively an order of class $(4, s)$ and an order of class $(1, s)$. As the arrival rate of orders of class $(1, s)$ equals the arrival rate of orders of class $(5, s)$, orders of class $(1, s)$ do not arrive sufficiently often to bring a pod of class $(1, s)$ to the replenishment level.

Table 10: Results for experiments with equal order arrival rates, t_{ot} in seconds and the utilizations in percentages

		$\xi = 0\%$				$\xi = 50\%$				$\xi = 100\%$			
M^s , U	r	t_{ot}	ρ_{pod}	ρ_{pick}	ρ_{repl}	t_{ot}	ρ_{pod}	ρ_{pick}	ρ_{repl}	t_{ot}	ρ_{pod}	ρ_{pick}	ρ_{repl}
1 , 36	(1, 5)	1192.0	37.8	20.1	0.4	142.8	38.8	20.1	0.7	182.5	48.3	20.1	4.0
1 , 36	(2, 4)	1237.8	36.7	10.0	0.5	135.9	37.9	10.2	0.9	177.0	48.2	10.1	5.0
1 , 36	(3, 3)	1144.6	36.4	6.8	0.7	133.9	37.5	6.7	1.2	170.1	47.2	6.7	6.6
1 , 36	(4, 2)	1166.7	36.0	5.0	1.0	133.0	37.1	5.0	1.8	175.4	48.1	5.0	10.1
1 , 36	(5, 1)	1222.4	36.1	4.0	2.0	132.7	37.1	4.0	3.6	180.8	49.0	4.1	20.2
2 , 18	(1, 5)	973.6	18.5	20.4	0.8	81.8	19.3	20.1	1.3	84.2	23.1	20.0	4.0
2 , 18	(2, 4)	949.6	17.6	10.0	1.0	76.8	18.5	10.1	1.7	79.2	22.6	10.0	5.1
2 , 18	(3, 3)	872.0	17.5	6.7	1.3	76.3	18.3	6.7	2.2	78.8	22.7	6.7	6.7
2 , 18	(4, 2)	907.4	17.4	5.0	2.0	76.2	18.4	5.1	3.3	78.8	22.9	5.1	10.2
2 , 18	(5, 1)	935.1	17.4	4.0	3.9	76.1	18.4	4.1	6.7	79.6	23.4	4.1	20.2
3 , 12	(1, 5)	1048.7	12.2	20.1	1.2	74.1	12.9	20.1	1.9	74.2	15.1	20.1	4.0
3 , 12	(2, 4)	1166.6	11.7	9.8	1.5	69.4	12.5	10.0	2.3	69.9	14.7	10.1	5.0
3 , 12	(3, 3)	1143.8	11.7	6.8	2.0	68.9	12.3	6.7	3.1	69.0	14.5	6.7	6.7
3 , 12	(4, 2)	1088.3	11.6	5.0	2.9	68.7	12.4	5.0	4.6	68.7	14.8	5.1	10.2
3 , 12	(5, 1)	1048.9	11.5	4.0	5.8	68.8	12.4	4.0	9.3	69.4	15.1	4.0	20.2
4 , 9	(1, 5)	1963.8	9.3	20.2	1.6	71.0	9.8	20.2	2.2	70.9	11.1	20.2	4.1
4 , 9	(2, 4)	2013.2	8.9	10.1	2.0	66.2	9.5	10.1	2.8	66.1	10.8	10.1	5.1
4 , 9	(3, 3)	1897.2	8.9	6.7	2.6	65.7	9.3	6.6	3.7	65.8	10.7	6.7	6.7
4 , 9	(4, 2)	1979.7	8.8	5.0	3.9	65.6	9.4	5.1	5.5	65.6	10.8	5.0	10.1
4 , 9	(5, 1)	2022.5	8.8	4.1	7.9	65.7	9.5	4.0	11.1	65.8	11.1	4.0	20.2
6 , 6	(1, 5)	–	–	–	–	67.5	6.7	20.1	3.0	67.4	7.3	20.1	4.0
6 , 6	(2, 4)	–	–	–	–	63.0	6.5	10.0	3.8	62.8	7.0	10.0	5.0
6 , 6	(3, 3)	–	–	–	–	62.6	6.6	6.8	5.1	62.5	7.0	6.7	6.7
6 , 6	(4, 2)	–	–	–	–	62.4	6.6	5.1	7.6	62.2	7.1	5.1	10.1
6 , 6	(5, 1)	–	–	–	–	62.5	6.6	4.0	15.3	62.5	7.2	4.0	20.0

References

Bolch, G., Greiner, S., de Meer, H., and Trivedi, K. S. (2006). *Queueing Networks and Markov Chains*. John Wiley & Sons, New York, 2nd edition.

Buitenhek, R., Van Houtum, G.-J., and Zijm, H. (2000). AMVA-based solution procedures for open queueing networks with population constraints. *Annals of Operations Research*, 93:15–40.

Business Wire. Amazon unveils its eighth generation fulfillment center. `www.businesswire.com/multimedia/home/20141130005031/en`. [Accessed September 23rd, 2016].

Ekren, B. Y., Heragu, S. S., Krishnamurthy, A., and Malmborg, C. J. (2014). Matrix-geometric solution for semi-open queuing network model of autonomous vehicle storage and retrieval system. *Computers & Industrial Engineering*, 68:78–86.

Fukunari, M. and Malmborg, C. J. (2009). A network queuing approach for evaluation of performance measures in autonomous vehicle storage and retrieval systems. *European Journal of Operational Research*, 193:152–167.

Heragu, S. S., Cai, X., Krishnamurthy, A., and Malmborg, C. J. (2011). Analytical models for analysis of automated warehouse material handling systems. *International Journal of Production Research*, 49(22):6833–6861.

Jia, J. (2005). *Solving Semi-Open Queuing Networks*. PhD thesis, Rensselaer Polytechnic Institute.

Kuo, P.-H., Krishnamurthy, A., and Malmborg, C. J. (2007). Design models for unit load storage and retrieval systems using autonomous vehicle technology and resource conserving storage and dwell point policies. *Applied Mathematical Modelling*, 31:2332–2346.

Lamballais, T., Roy, D., and de Koster, M. B. M. (2017). Estimating performance in a robotic mobile fulfillment system. *European Journal of Operations Research*, 256:976–990.

Marchet, G., Melacini, M., Perotti, S., and Tappia, E. (2013). Development of a framework for the design of autonomous vehicle storage and retrieval systems. *International Journal of Production Research*, 51(14):4365–4387.

Nigam, S., Roy, D., de Koster, M. B. M., and Adan, I. J. B. F. (2014). Analysis of class-based storage strategies for the mobile shelf-based order pick system. In *Progress in Material Handling Research: 2014*.

Roy, D., Krishnamurthy, A., Heragu, S. S., and Malmborg, C. J. (2012). Performance analysis and design trade-offs in warehouses with autonomous vehicle technology. *IIE Transactions*, 44:1045–1060.

Roy, D., Krishnamurthy, A., Heragu, S. S., and Malmborg, C. J. (2013). Blocking effects in warehouse systems with autonomous vehicles. *IEEE Transactions on Automation Science and Engineering*, 99:1–13.

Roy, D., Krishnamurthy, A., Heragu, S. S., and Malmborg, C. J. (2015a). Queuing models to analyze dwell-point and cross-aisle location in autonomous vehicle-based warehouse systems. *European Journal of Operational Research*, 242:72–87.

Roy, D., Krishnamurthy, A., Heragu, S. S., and Malmborg, C. J. (2015b). Stochastic models for unit-load operations in warehouse systems with autonomous vehicles. *Annals of Operations Research*, 231:129–155.

Roy, D., Krishnamurthy, A., Heragu, S. S., and Malmborg, C. J. (2016). A simulation framework for studying blocking effects in warehouse systems with autonomous vehicles. *European Journal of Industrial Engineering*, 10(1):51–80.

Tappia, E., Roy, D., De Koster, M. B. M., and Melacini, M. (2016). Modeling, analysis, and design insights for shuttle-based compact storage systems. Forthcoming in Transportation Science.

Wulfraat, M. Is Kiva systems a good fit for your distribution center? an unbiased distribution consultant evaluation. `http://www.mwpvl.com/html/kiva_systems.html`. [Accessed September 23rd, 2016].

Wurman, P. R., D'Andrea, R., and Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–19.

Wurman, P. R. and Enright, J. J. (2011). Optimization and coordinated autonomy in mobile fulfillment systems. Working paper.