# AUTOMATED ORDER-PICKING WORKSTATION HANDLING OUT-OF-SEQUENCE PRODUCT ARRIVALS

R. Andriansyah, L. F. P. Etman, I. J. B. F. Adan and J. E. Rooda
*Department of Mechanical Engineering, Eindhoven University of Technology, Den Dolech 2, Eindhoven, The Netherlands*

Keywords: End-of-aisle, Warehouse, Order-picking, Workstation, Picking policy, Simulation.

Abstract: A novel design of an automated order-picking workstation processing multiple orders is proposed for a warehouse with an end-of-aisle order-picking system. A typical problem at this workstation is the out-of-sequence arrival of products, assuming the workstation receives products for multiple orders simultaneously. As multiple products are present, the picking sequence at the workstation affects the system throughput. The order throughput performance of three picking policies is compared under different extents of out-of-sequence arrivals. Experimental results show the capability of the workstation to handle an arbitrary extent of out-of-sequence arrival of products. Noteworthy insights for design considerations of such systems are drawn.



## 1 INTRODUCTION

Warehouses nowadays are operating in a more-than-ever challenging environment. Internet orders are forcing warehouses to keep greater varieties of SKUs (Stock Keeping Units) and to deliver low-volume orders more frequently. Moreover, retailers are setting tighter order delivery schedules, so as to avoid out-of-stock situations. These challenges combined with fierce market competition call for a more efficient order-picking operation. After all, order-picking, the process of retrieving products from the storage area to fulfill customer orders, is estimated to account for 55% of the total warehouse operating cost (Tompkins et al., 2003).

Warehouse automation is becoming a common practice to respond to these challenges. This can be seen from the notable growth in sales of automated material handling systems in recent years (Baker and Halim, 2007). Different sorts of automated picking technologies are continuously being introduced to the market. Each technology is typically designed by considering factors such as the number of SKUs and the expected picking volume, among others. A selection methodology (Dallari et al., 2009) can be used to determine the most suitable order-picking system for a given set of warehouse requirements.

The current study focuses on a particular class of order-picking systems, namely an end-of-aisle order-picking system. This system is typically composed of separate processing units including a storage area and an order-picking workstation, which are connected by a transportation unit such as a closed-loop conveyor. Such configuration is capable of processing a significantly large number of SKUs.

For such a system it is desirable to process multiple orders at the same time to gain high throughput. To do so, products for the multiple orders must be retrieved simultaneously from the storage area. This, however, poses a threat to the system performance. Ideally, products required for the earliest order released arrive earlier at the workstation than any other products. However, due to factors such as the number of storage racks, the composition of SKUs across the storage racks, and the retrieval time of products, products may not arrive completely in the same sequence as requested by the workstation. This situation is referred to as *out-of-sequence arrivals* of products. Combined with inefficient picking operations at the workstation, such situation deteriorates the throughput performance of the order-picking system.

In this paper we study an automated order-picking workstation that is able to deal with out-of-sequence arrivals. A novel workstation design with an integrated carrousel mechanism is proposed, where multiple orders can be processed simultaneously. The design is capable of handling arbitrary out-of-sequence arrival distributions. While there are a number of factors that affect the workstation performance, the focus of our study is on the picking policy for the proposed workstation design. We consider three picking policies and show that a significant gain in throughput can

Andriansyah R., F. P. Etman L., J. B. F. Adan I. and E. Rooda J..
AUTOMATED ORDER-PICKING WORKSTATION HANDLING OUT-OF-SEQUENCE PRODUCT ARRIVALS.
DOI: 10.5220/0003599902830292
In *Proceedings of 1st International Conference on Simulation and Modeling Methodologies, Technologies and Applications* (SIMULTECH-2011), pages 283-292
ISBN: 978-989-8425-78-2

be realized by applying a proper picking policy.

The paper is organized as follows. Section 2 elaborates the configuration of the order-picking workstation under study. Section 3 describes the three picking policies. In Section 4 a number of simulation experiments are performed to see the performance of the workstation under different settings. Finally, Section 5 concludes the paper.

## 2 AUTOMATED WORKSTATION

The automated order-picking workstation discussed in this paper is part of a larger order-picking system, which typically also comprises of a pallet storage area and conveyors. In this order-picking system, product pallets are retrieved from the pallet storage area using an AS/RS (Automated Storage/Retrieval System). Each product pallet contains a number of items from one SKU only. These pallets are transported using conveyors from the storage area to one of the picking workstations. At the workstation, items on product pallets are picked onto order pallets to fulfill orders. Product pallets that have been processed but still contain some items left are returned to the storage area. An order pallet corresponds to one order. Multiple orders are present and each order requires a number of SKUs, which is referred to as the order length.

We require the automated order-picking workstation to be capable of handling different extents of out-of-sequence arrivals. This is because the AS/RS at the storage area is assumed to be the bottleneck of the entire order-picking system. Hence, the AS/RS should operate at its maximal capacity without taking into account the issue of out-of-sequence arrivals. The automated order-picking workstation should therefore be configured such that it is able to operate even under a high extent of out-of-sequence arrivals. Furthermore, a well-defined picking policy is required. This is because products for multiple orders are present at the workstation. A picking policy prescribes the sequence in which products should be picked to fulfill orders. The desired picking policy is the one that gives a high order throughput under arbitrary out-of-sequence arrival distribution.

The basic configuration of the automated order-picking workstation is shown in Figure 1. There are a number of conveyors that act as buffer lanes. The layout of Figure 1 has one order buffer and two product buffers. These buffers follow a first-in-first-out principle; only pallets located at the head of the buffers (foremost) can leave the workstation. A robot picks items from a product pallet and drop them onto the corresponding order pallet. This robot has a limited operating area referred to as the pickable area. Only pallets inside the pickable area are accessible to the robot. Once picked, a product pallet may leave the workstation, if possible. An order pallet may only leave if it already contains all SKUs required by the order. Once a pallet leaves the workstation, all remaining pallets in the same buffer shift one position forward. It is possible, however, that a picked pallet cannot leave the workstation because there is another pallet in front of the picked pallet. The picked pallet is then referred to as a *blocked pallet*.

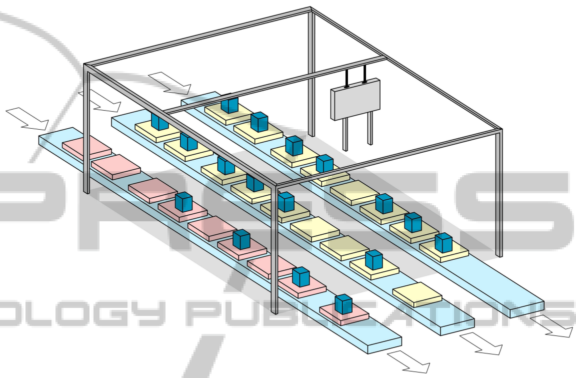


Figure 1: Layout of the automated order-picking workstation. One buffer lane is used for order pallets (red) and two buffer lanes are used for product pallets (yellow). The pickable area is the area inside the frame. The arrows denote the direction of pallets movements.

Other issues relevant to this automated order-picking workstation are the pipeline capacity and system deadlock. The *pipeline capacity* is the maximum number of pallets that can be simultaneously on the way to the workstations. It is crucial that the pipeline is sufficiently filled. This reduces the interarrival time of product pallets at the workstation, hence providing the robot with enough work. Furthermore, the workstation needs an additional mechanism to avoid deadlock. Due to the out-of-sequence arrivals, system deadlock may occur namely when the buffers are full and the pallets at the heads of all product and order buffers cannot be picked. A proper design of such end-of-aisle workstation should therefore take this eminent problem into account.

A conceptual design of the workstation that avoids such a deadlock situation is proposed in Figure 2. The workstation is comprised of four buffers, namely order buffer *OP*, product buffer *PP*, recirculated product buffer *rPP*, and carrousel buffer *cPP*. The buffers *rPP* and *cPP* are actually parts of a carrousel. These two buffers can rotate independently in one direction, allowing product pallets to exchange places between the buffers. The robot can only pick/drop items from/to pallets within the pickable area, which is depicted by the striped line in Figure 2.

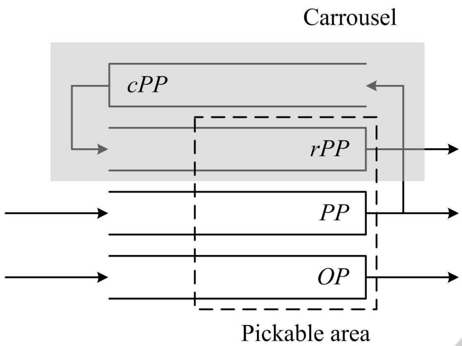



Figure 2: Conceptual design of the automated order-picking workstation.

The system works as follows. Products retrieved from the storage area arrive at the product buffer *PP*. Each product has an identification number (ID) that is equal to the ID of the order to which the product belongs. The robot then selects a product pallet inside the pickable area of product buffers *PP* or *rPP*, picks an item from the product pallet, and drops the item onto the corresponding order pallet in *OP*. Only product pallets with IDs that belong to one of the *pickable orders* can be selected as the next pallet to be picked. A pickable order is an order whose order pallet and at least one of the corresponding product pallet are located inside the pickable area.

There are two operations involving the carrousel:

1. Inserting a Product Pallet into the Carrousel.
   A product pallet is inserted if no product pallet in *rPP*, *cPP* and inside the pickable area of *PP* is required by the order pallets inside the pickable area of *OP*. A product pallet is inserted from *PP* to the carrousel buffer *cPP*. This also means that the product pallet has been recirculated. Subsequently, if there is a space available in *rPP*, the product pallet is immediately moved from *cPP* to *rPP*.
2. Rotating the Carrousel.
   The carrousel is rotated if no product pallet inside the pickable area of *PP* and *rPP* is required by the order pallets inside the pickable area of *OP*, but such a required product pallet is present in *cPP* or outside the pickable area of *rPP*. During the rotation, one product pallet is moved from *rPP* to *cPP* and one product pallet is moved from *cPP* to *rPP*.

## 3 PICKING POLICY

The robot in the workstation selects the next product pallet to be picked based on a predefined picking policy. Three picking policies are distinguished as *nearest-to-the-head, nearest neighbor,* and *dynamic programming* picking policies.

All picking policies use the information about the product and order pallets inside the pickable area of product buffers *PP*, *rPP*, and order buffer *OP*. First, the content of *rPP* and *OP* is evaluated. If there is a product pallet that can be picked in *rPP*, then the robot picks an item from the product pallet and put it into its corresponding order pallet in *OP*. Otherwise, the content of *PP* and *OP* is evaluated. Similarly, if there is a product pallet that can be picked in *PP*, then the robot picks an item from the product pallet and put it into its corresponding order pallet in *OP*. That is, priority is given to picking a product pallet from *rPP*. In case no product pallet can be picked from both *rPP* and *PP*, one of the two carrousel operations is performed. The following section elaborates for each policy the criteria used for selecting the product pallets to be picked.

### 3.1 Nearest-to-the-Head

The nearest-to-the-head picking policy aims at an uninterrupted flow of product pallets at the workstation. Recall that a product pallet can only leave the workstation once it has been picked and is located at the head of the product buffer. Once a product pallet leaves, all remaining product pallets in the buffer shift one position forward and a new product pallet can enter the workstation. Therefore, giving priority to picking the product pallet located at the head of the product buffer supports a continuous flow of product pallets at the workstation.

The nearest-to-the-head picking policy requires the robot to pick the product pallets according to their sequence in the buffer. It evaluates first whether the product pallet at the head of the product buffer can be picked. If this is not possible, then the product pallet located at the second position in the buffer is evaluated next. This evaluation is performed continuously until either a pickable order is found or the end of the pickable area is reached.

### 3.2 Nearest Neighbor

One of the most serious pitfalls of the nearest-to-the-head picking policy is that it may cause the robot to travel without carrying any item from the current position of the robot to the head of the product buffer.

Such travel consumes time, but is a non-value-added process; it is thus detrimental to the order throughput of the workstation. The nearest neighbor picking policy is proposed, which minimizes the distance in which the robot travels without carrying any item.

The nearest neighbor picking policy requires the robot to pick the product pallet located nearest to the current position of the robot. If there is more than one pickable order with the same distance to the robot, then the robot picks the pallet that is located closer to the head of the buffer.

### 3.3 Dynamic Programming

Given an unlimited supply of product and order pallets at the workstation, the order throughput of the workstation depends on the robot processing time; a lower processing time leads to a higher throughput. The robot processing time consists of a travel time, pick time, and drop time. Assuming a relatively constant pick and drop time, one can increase the workstation throughput by reducing the travel time, which is a function of the robot travel distance.

The dynamic programming policy constructs a *picking trip* that minimizes the robot travel distance based on the current content of product and order buffers. The robot then picks a number of orders in a sequence as indicated in the picking trip. Once a product or an order pallet leaves the workstation, the buffer content changes. In this case, a new picking trip is constructed using the dynamic programming approach based on the new content of the product and order buffers.

The robot travel distance depends on the sequence of picking a number of orders. When picking one order, the robot travels from its current position to a product pallet (picking an item) and finally to the corresponding order pallet (dropping the item). Following a picking process, the robot thus always starts a new picking process from the location of the previously served order pallet.

A dynamic programming policy is formulated to minimize travel distance $D$ to reduce the travel time and consequently to increase the workstation throughput, where:

$$D = \sum_{i=1}^{m} (|y_{i-1} - x_i| + |x_i - y_i|), \qquad (1)$$

where:

$x_i$ = position of product pallet of the order picked at the $i$th step.
$y_i$ = position of order pallet of the order picked at the $i$th step.
$y_0$ = current position of the robot.

In the above formulation, $m$ is the number of orders picked within a picking trip, which is referred to as the *trip length*. Note that travel distance $D$ increases with trip length $m$. Therefore, the optimal trip is the one that minimizes travel distance $D$ per order, so the one minimizing $D/m$.

The first step in constructing a picking trip that minimizes travel distance $D$ is evaluating the pallets contained in the product and order buffers. Recall that each product pallet has an identification number represented by the ID of the order to which the product belongs. Let

$P$ be the set of IDs of product pallets within the pickable area
$Q$ be the set of IDs of order pallets within the pickable area

Then $R = P \cap Q$ gives the IDs of *pickable orders*. These are the orders whose product and order pallets are both located inside the pickable area. Let $S \subseteq R$ be the IDs of *target orders*. These are the pickable orders whose product and/or order pallets are located at the head of the buffer. A target order is the last order to be picked in a picking trip because picking a target order causes a product and/or an order pallet to leave the workstation. With this regard, $x_m$ and $y_m$ in (1) are the positions of the product pallet and the order pallet for the target order, respectively. Picking a target order leads to a new content of either a product buffer or the order buffer, or both. Hence, a new picking trip needs to be constructed.

Having a target order is a requirement for constructing a picking trip. Recall that there are two product buffers namely *rPP* and *PP* evaluated against the order buffer *OP*. If both *rPP* and *PP* contain at least one target order, then a picking trip is constructed from *rPP*. That is, *rPP* has priority over *PP*. If only one product buffer contains a target order, then a picking trip is constructed from that product buffer. Otherwise, if none of the two product buffers contains a target order, then one of the two carrousel operations explained previously is performed.

Figure 3 shows an example of the content of product buffer *PP* and order buffer *OP*. Assume now that product buffer *rPP* does not contain a target pallet and therefore is disregarded from the figure. Based on this figure, $P = \{11, 12, 14, 15, 18\}$, $Q = \{11, 12, 13, 14, 15\}$, $R = \{11, 12, 14, 15\}$, and $S = \{11\}$.

Constructing a picking trip can be regarded as a Traveling Salesman Problem (TSP). That is, given trip length $m$, pickable orders $R$, target orders $S$, and current position of robot $y_0$, determine the sequence of picking $m$ pickable orders involving one and only one $s \in S$, which starts at $y_0$ and ends at $s$ such that $D$

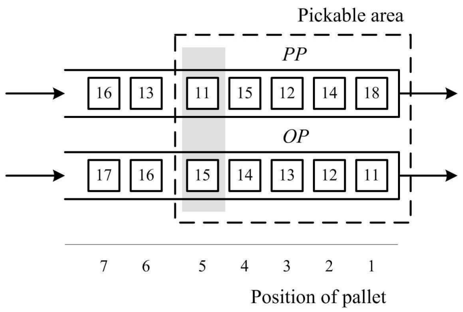



Figure 3: An example of the content of product (*PP*) and order (*OP*) buffers. The numbers represent the IDs of pallets. A product pallet at *PP* belongs to an order pallet at *OP* with the same ID. The robot is currently at position 5.

as given in Equation (1) is minimized.

Note that given $R$ there are many TSPs of length $m$ that contain one $s \in S$. Each TSP has to be separately solved. That is, for $m = 1, ..., m_{\max}$ the TSPs of length $m$ are generated, and subsequently solved. Herein, the maximum trip length $m_{\max}$ follows from the number of elements in $R$ excluding $S$ and subsequently adding *one* element of $S$ in the trip. Hence, the maximum trip length is $m_{\max} = |R| - |S| + 1$, where $|R|$ and $|S|$ denote the number of orders in $R$ and $S$, respectively.

In the example of Figure 3 where $R = \{11, 12, 14, 15\}$ and $S = \{11\}$, the maximum trip length is 4, resulting in eight TSPs that contain one $s \in S$, each of which is a subset of $R$, namely for $m = 1$: {11}; for $m = 2$: {11,12}, {11,14}, {11,15}; for $m = 3$: {11,12,14}, {11,12,15}, {11,14,15}; and for $m = 4$: {11,12,14,15}. The number of TSPs increases exponentially with the number of pickable orders. With $|R| = 10$ and $|S| = 1$, as many as 512 TSPs need to be solved.

The optimal picking trip given $R$, $S$, and $y_0$ is the one that minimizes travel distance per order, i.e., the one minimizing $D/m$. Let $r^*_m$ be the picking trip minimizing the travel distance $D$ from *all* TSP of length $m$ as a subset of $R$. Let $D_{r^*_m}$ be the resulting travel distance for $r^*_m$. The optimal picking trip is

$$r^* = \arg\min\{\frac{D_{r^*_1}}{1}, \frac{D_{r^*_2}}{2}, ..., \frac{D_{r^*_{m_{\max}}}}{m_{\max}}\}. \qquad (2)$$

We use dynamic programming for TSP (Bellman, 1962) to solve each TSP. With dynamic programming, a TSP is handled in smaller parts by solving subproblems sequentially. The solutions to these subproblems are then stored for future use. Larger subproblems are solved by a recursion formula from the smaller subproblems. The complete solution for the TSP is obtained through backtracking the solutions of the subproblems. Note that there are a number of TSP heuristics other than dynamic programming available in the literature. For a review of the performance of various TSP heuristics we refer to (Johnson, 1990). A description of the dynamic programming for TSP is provided in the Appendix. For the example in Figure 3, the optimal picking trip is found to be $y_0 \to 12 \to 14 \to 15 \to 11$.

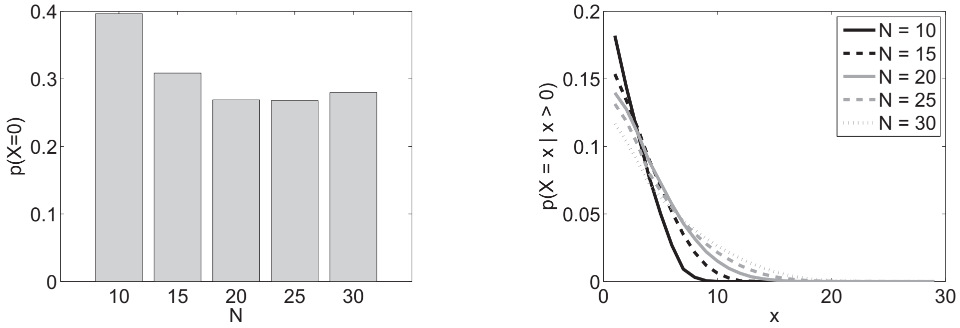
# 4 SIMULATION EXPERIMENTS

We investigate the performance of the automated order-picking workstation under all three picking policies for single and multiple products per order. The performance measures of interest is the order throughput. A method to model the out-of-sequence arrival of products is proposed.

## 4.1 Modeling Out-of-Sequence Arrivals

An example of out-of-sequence arrival of product pallets at the workstation can also be seen from Figure 3. In this example we assume that each order pallet in buffer *OP* requires one product (that is, order length $k = 1$). Each order pallet has an ID, where lower IDs represent orders that are released earlier. A product pallet in *PP* with the same ID as an order pallet in *OP* means that the product pallet is required for the order pallet. Ideally, product pallets arrive in the same sequence as the order pallets, namely products for older orders arrive first. However, products arrive out-of-sequence, where e.g., product pallet 18 arrived earlier than the product pallets for orders 11,12,...,17. That is, product pallet 18 *overtakes* seven product pallets. The number of overtaken products is calculated for each incoming product pallet. This gives an overtaking distribution that characterizes the out-of-sequence arrival of products.

An overtaking distribution is obtained by measuring the overtaking from a detailed simulation model of an AS/RS. The AS/RS system consisted of five storage racks from which products were retrieved. A total of 10,000 SKUs were contained in the storage racks; each SKU was contained only on one product pallet. This setting represents a warehouse that serves slow moving products. The cranes of the storage racks had a retrieval batch size of four pallets. That is, the cranes wait until there is a retrieval command for four SKUs before it starts retrieving the SKUs. The maximal pipeline capacity is $N$ product pallets. These are the product pallets that have been retrieved by the AS/RS and are on their way to (but have not entered) the workstation buffer. Figure 4 shows the overtaking distributions measured from the




(a) Probability of no overtaking. (b) Probability of overtaking $x$ products.

Figure 4: Measured overtaking distribution from a detailed simulation model.

detailed simulation model for different pipeline capacities $N = \{10, 15, 20, 25, 30\}$.

We used Figure 4(b) as inspiration to develop an analytical function to represent the overtaking. Let random variable $X$ be the number of overtaken products, which can take a value of $\{0, 1, 2, 3, ..., N-1\}$. Let

$$P(X > 0) = p,$$
$$P(X = 0) = 1 - p. \tag{3}$$

Here, $P(X = 0)$ is the probability that a certain product does not overtake other products. In the case of overtaking ($X > 0$) the possible number of overtaken products is $\{1, 2, 3, ..., N-1\}$. To obtain a shape of the overtaking distribution similar to Figure 4, we determine the probability of overtaking $x$ products as:

$$P(X = x \mid x \in Y) = f(x) = (ax + b)\, xe^{-cx}, \tag{4}$$

where $Y = \{1, 2, 3, ..., N-1\}$. There are three parameters in the function, namely $a, b$, and $c$. The values of $a$ and $b$ are calculated given $c$. Since the maximal number of overtaken products is $N-1$, it is known that:

$$P(X = N) = (aN + b)\, Ne^{-cN} = 0. \tag{5}$$

Furthermore, the sum of all probabilities of overtaken products is equal to 1. That is:

$$\sum_{x=1}^{N} f(x) = 1. \tag{6}$$

From (5) and applying (5) in (6), we obtain:

$$b = -aN.$$
$$a = \frac{1}{\sum_{x=1}^{N} (x - N)\, xe^{-cx}}.$$

Hence, parameters $p$ and $c$ determine the overtaking distribution given by (3) and (4).

To illustrate the proposed overtaking distribution, we have fitted function (4) to data measured from the detailed simulation model. The value of parameter $c$, also referred to as the shape parameter, is determined using the nonlinear regression fitting method in Matlab. Figure 5 shows the comparison of overtaking from simulation data and from the fitted function for $N = 10$.

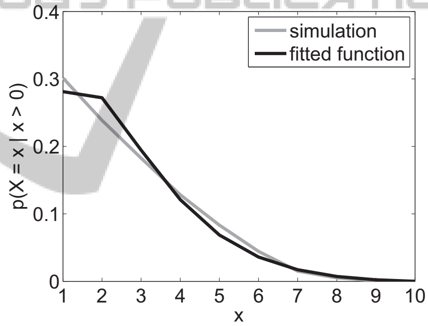



Figure 5: Comparison of overtaking from a detailed simulation and from the fitted overtaking function.

The shape parameter $c$ can be changed for experimental purposes. Figure 6 shows several shapes of the overtaking distribution given various $c$ values for $N = 10$. In general, larger $c$ leads to smaller mean number of overtaken products, given that there is overtaking. By using different settings for overtaking probability $p$ and shape parameter $c$, we are able to analyze the performance of the automated order-picking workstation under different extents of out-of-sequence product arrivals.

## 4.2 Assumptions and Settings

The following assumptions apply to all simulation experiments. The product buffer, the recirculated product buffer, and the order pallet buffer have a finite capacity of 10 pallets. A new product pallet is generated into the product buffer as soon as a space in the

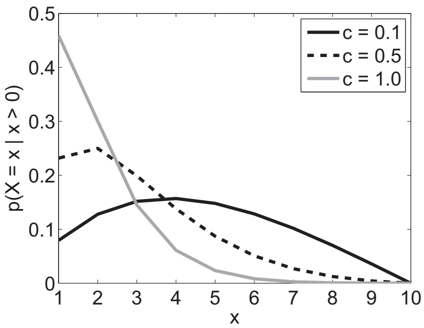



Figure 6: Comparison of overtaking distributions with different values of shape parameter $c$.

buffer becomes available. That is, there is no interarrival time involved in generating a new product pallet. The processing time of the robot is comprised of pick time, travel time, and drop time. The travel time depends on the travel distance $D$ (in pallets) that covers the cycle: current position - product pallet - order pallet. We assume that the robot requires 0.25 seconds to travel a distance of 1 pallet measured in horizontal direction only. That is, the movement from product to order pallet and vice versa is performed during the horizontal movement. The pick and drop time is assumed to be constant at 2 seconds each. Hence, the processing time $t_e$ of the robot is (in seconds)

$$t_e = 0.25\,D + 4. \tag{7}$$

It is assumed that rotating the carrousel costs 2 seconds per rotation and that shifting the pallets forward in the buffer take place while the robot is traveling.

Table 1 gives a list of parameters that are used in the experiments. Overtaking probability $p$ and overtaking shape parameter $c$ are used to create an overtaking distribution (see Section 4.1), representing the out-of-sequence arrivals of products. Order length $k$ represents the number of SKUs required for an order, which equals to the number of required product pallets given that a product pallet contains one SKU only. The maximum number of product pallets that are on their way to the workstation simultaneously is denoted by $N$, which is the pipeline capacity. The size of pickable area $L$ gives the number of pallets that are within reach of the robot.

Table 1: List of parameters.

| Parameter | Notation |
|---|---|
| Overtaking probability | $p$ |
| Overtaking shape parameter | $c$ |
| Order length | $k$ |
| Size of pickable area | $L$ |
| Pipeline capacity | $N$ |

All simulation experiments have been performed using process algebra based discrete-event simulation language $\chi$ (Chi) 1.0. We refer to (Hofkamp and Rooda, 2008) for a definition of this language. A tutorial of the language is provided by (Rooda and Vervoort, 2007).

The following setup is used for all experiments. Each experiment consists of 50 simulation runs excluding a warm-up period of 30,000 time units. This warm up period was determined using Welch's method (Welch, 1983) based on the combination of parameters that leads to the highest overtaking considered, namely $p = 0.8$, $c = 0.1$, $k = 1$, $L = 1$, and $N = 30$. The resulting warm-up period is then used for all experimental settings. A simulation run is terminated after 100,000 product pallets have been picked.

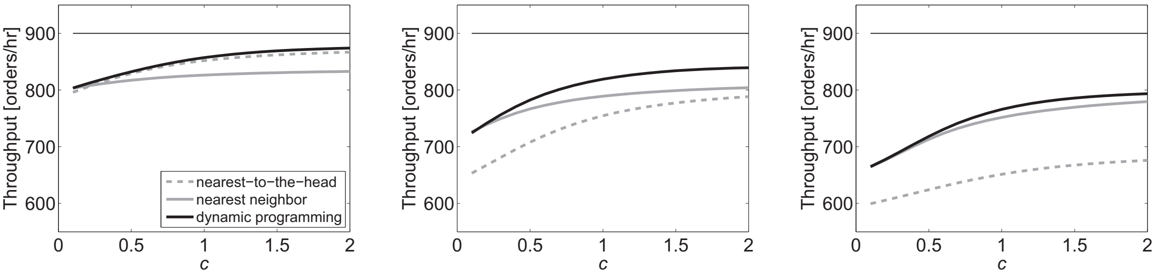
Out-of-sequence arrivals are generated in the simulation model as follows. A generator of product pallet holds a list of product pallets to be generated for orders in order buffer $OP$. For example, the list $[11, 12, 13, 14, 15, 16, 17]$ contains 7 product pallets each for a different order as denoted by the ID numbers. Each time a product pallet is generated, it is determined whether or not the new product pallet will overtake other product pallets based on the value of parameter $p$. In case of overtaking, the number of overtaken pallet $x$ is sampled. Subsequently, the first $x$ product pallets in the list is skipped and the product pallet at position $x+1$ in the list is generated. In the previous list example, if $x = 2$ then product pallet 13 is generated. The updated list excludes this pallet.

## 4.3 Experimental Results

### 4.3.1 Single Product per Order

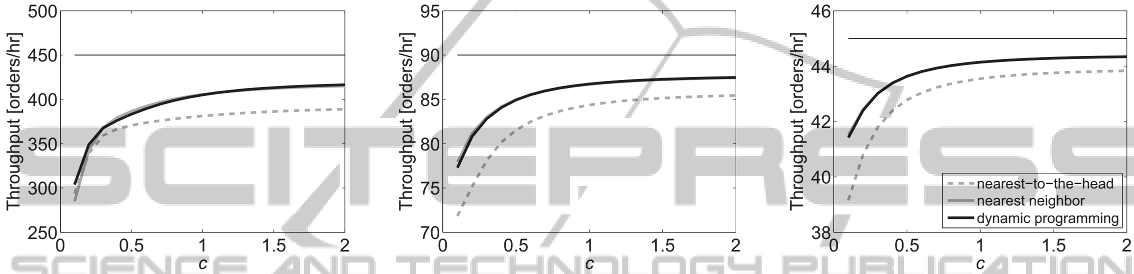
First, we consider the case where each order consists of exactly one product ($k = 1$) and the pipeline capacity $N = 10$. Only one item is required from each product pallet. The effect of the overtaking distribution on the order throughput is investigated. The size of pickable area $L = 10$. Figure 7 depicts the order throughput of the workstation as a function of overtaking shape parameter $c$ given overtaking probability $p = \{0.2, 0.5, 0.8\}$. Note that the straight line in this figure is the maximal order throughput *without* out-of-sequence arrival ($p = 0.0$). In this case, the robot stays at the head of the buffer causing the travel distance to become zero. Thus, the maximal throughput is constant at $\frac{1}{4} \times 3600$ orders/hr $= 900$ orders/hr.

The nearest-to-the-head picking policy performs well only when the product pallets arrive relatively in a good sequence, that is, with a low probability of overtaking. As products arrive more out-of-sequence,




(a) $p = 0.2$. (b) $p = 0.5$. (c) $p = 0.8$.

Figure 7: Order throughput for $k = 1$, $L = 10$, $N = 10$.




(a) $k = 2$. (b) $k = 10$. (c) $k = 20$.

Figure 8: Order throughput for variable order length $k$, $p = 0.8$, $L = 5$, $N = 30$.



the performance of the nearest-to-the-head picking policy deteriorates compared to the other policies. With high overtaking, the resulting order throughput from the nearest-to-the-head policy may even be 17% lower than that of the dynamic programming policy. The reason is that with high overtaking, the nearest-to-the-head policy requires the robot to return to the first position in the buffer after picking. This return trip is performed without picking any other product pallets. Hence, the extra traveling without picking deteriorates order throughput. However, when products arrive at the buffer in a good sequence, most picking under nearest-to-the-head picking policy takes place at the foremost position in the buffer. No extra traveling occurs, which results in a high order throughput.

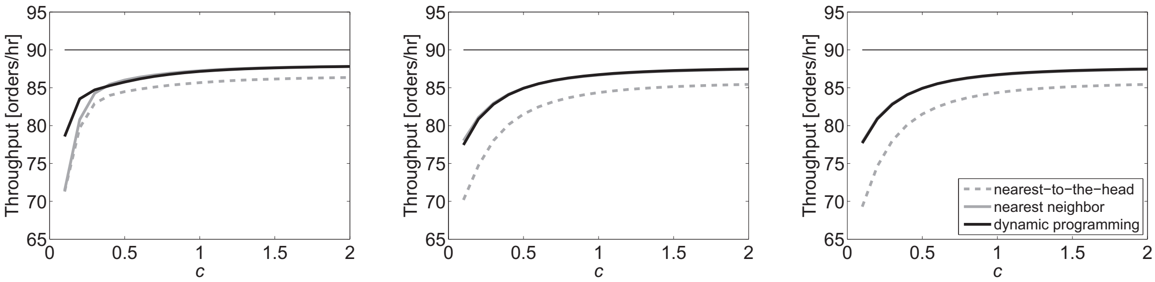
Contrary to the nearest-to-the-head policy, the nearest neighbor policy performs better at a high extent of out-of-sequence arrivals. This policy prevents the robot to travel empty by picking whichever product pallet nearest to the current location of the robot. Such way of working is advantageous particularly when products and orders are not sequenced neatly in the buffer. However, with low overtaking it may happen that the robot skips the product pallet at the foremost position of the product buffer and picks the product pallets in the rest of the buffer. The robot eventually returns to the foremost position of the product buffer. This return trip is performed without picking, causing extra traveling that deteriorates order throughput.

The dynamic programming policy perform superior to the nearest-to-the-head and nearest neighbor policies. In general, the added value of the dynamic programming policy is larger at high overtaking. An optimal picking trip that minimizes the robot travel distance results in a high throughput under arbitrary extent of out-of-sequence arrivals.

Figure 7 also shows the effect of out-of-sequence arrivals. This can be seen by comparing the straight line (the maximal throughput when all pallets arrive in a good sequence) with the other lines in the figure. When 20% of all product pallets arrive out-of-sequence (Figure 7(a)), the decline in order throughput may already reach 11%. It is therefore beneficial to have a mechanism at the AS/RS that prevents out-of-sequence arrivals of products at the workstation. However, considering that the AS/RS is typically the bottleneck of the order-picking system, such mechanism should not compromise AS/RS throughput.

### 4.3.2 Multiple Products per Order

Now we assume that each order requires more than one product. Multiple product pallets are needed for an order, but each product pallet is only picked once (one item per product). As a worst-case scenario we study the system performance under high overtaking $p = 0.8$ with a large pipeline capacity $N = 30$.




(a) $L = 2$. (b) $L = 6$. (c) $L = 10$.

Figure 9: Order throughput for variable pickable area size $L$, $p = 0.8$, $k = 10$, $N = 30$.

Figure 8 shows the resulting order throughput when the order lengths are fixed at $k = \{2, 10, 20\}$ for all three policies. The dynamic programming picking policy consistently gives a high order throughput, which is not the case for the other two policies. This is also true given different sizes of pickable area $L = \{2, 6, 10\}$, as depicted in Figure 9. That is, the dynamic programming policy is robust to the order length and the pickable area size. Note that in Figures 8 and 9 the straight lines give the maximal order throughput when all product pallets arrive in a good sequence ($p = 0.0$). In this case, the maximal throughput is constant at $\frac{1}{4} \times \frac{3600}{k}$ orders/hr.

## 5 CONCLUSIONS

The performance of a novel conceptual design for an automated order-picking workstation processing multiple orders has been studied. We highlighted the typical problem of out-of-sequence arrival of products at the workstation. This has been modeled using a so-called overtaking function. Furthermore, three picking policies have been evaluated. The picking policy based on dynamic programming gives the highest order throughput. Therefore, the use of dynamic programming picking policy in such a workstation is advocated.

The conceptual design of the workstation includes a carrousel. This is actually a built-in solution of the workstation to overcome deadlock situation due to the out-of-sequence arrival of products. Having this carrousel, we do not require products to be delivered in a good sequence, nor do we require a sequencer (e.g., vertical buffer) in front of the workstation to rearrange product arrivals at the workstation. Such an extra hardware is typically expensive.

The proposed overtaking function is a simple function that requires only two parameters. This function is capable of modeling different extents of out-of-sequence arrivals. Such a function is practical for performance evaluation when factors influencing out-of-sequence arrivals change.

A significant improvement in throughput can be gained by applying a smart picking policy at the workstation processing multiple orders. This is particularly the case when the number of pickable orders in the buffer at any time is large and there is a high extent of out-of-sequence arrival of products. Experimental results showed that the dynamic programming policy, which strives to minimize the travel distance of the robot during a sequence of picking, perform well under low, medium, and high extent of out-of-sequence arrivals.

## ACKNOWLEDGEMENTS

This work has been carried out as part of the FALCON project under the responsibility of the Embedded Systems Institute with Vanderlande Industries as the carrying industrial partner. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Embedded Systems Institute (BSIK03021) program.

## REFERENCES

Baker, P. and Halim, Z. (2007). An exploration of warehouse automation implementations: cost, service and flexibility issues. *Supply Chain Management: An International Journal*, 12.

Bellman, R. (1962). Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63.

Dallari, F., Marchet, G., and Melacini, M. (2009). Design of order picking system. *International Journal of Advanced Manufacturing Technology*, 42(1-2):1–12.

Hofkamp, A. T. and Rooda, J. E. (2008). Chi 1.0 reference manual. Systems Engineering Report 2008-04, Eindhoven University of Technology, Eindhoven.

Johnson, D. S. (1990). Local optimization and the traveling salesman problem. In *Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 446–461. Springer Berlin / Heidelberg.

Rooda, J. E. and Vervoort, J. (2007). *Analysis of Manufacturing Systems*. Eindhoven University of Technology, Eindhoven.

Tompkins, J. A., White, J. A., Bozer, Y. A., Frazelle, E. H., and Tanchoco, J. M. A. (2003). *Facilities Planning*. John Wiley and Sons, New Jersey, 2 edition.

Welch, P. (1983). The statistical analysis of simulation results. In *Computer Performance Modeling Handbook*. Academic Press.

# APPENDIX

## Dynamic Programming for TSP

Let $R$ be the set of pickable orders, $S \subseteq R$ denotes set of target orders. These are pickable orders whose product and/or order pallets are located at the head of the buffer. Let $U$ be a subset of $R$ containing $m$ orders and one $s \in S$; this is a TSP of length $m$. The robot initial position is $y_0$. A distance matrix is constructed based on the position of product and order pallets whose IDs are contained in $U$. Distance $d_{ij}$ is the distance traveled by the robot when picking one order, and is defined as

$$d_{ij} = |y_i - x_j| + |x_j - y_j| \tag{8}$$

for any $i, j \in U$ and $i \neq s$, where:

$x_i$ = position of product pallet with ID $i$
$y_i$ = position of order pallet with ID $i$

This formulation covers the distance from the previous order pallet $i$ (current robot position) to the current product pallet $j$ (pick item) and finally to the current order pallet $j$ (drop item; new robot position). Additionally, we set an artificial distance $d_{sj} > 2 \times L$, where $L$ is the size of the pickable area. This way, we prevent target order $s$ to be picked in the middle of the picking trip and ensure that the target order is the last order picked in the picking trip. The following recursion is used to solve $U$, a TSP of length $m$.

For all $i \in U$:
  $g(i, \emptyset) = 0$
For $n = 1$ to $m - 1$:
  For all subsets $V \subseteq U$ containing $n$ orders:
    For all $i \notin V$ and $i \in U$:
      $g(i,V) = \min_{j \in V} \{d_{ij} + g(j, V \setminus \{j\})\}$

where:
$g(i,V)$ = the shortest total distance for picking all orders in $V$ starting from order $i$.

Once $g(i,V)$ is obtained, $p(i,V)$ is determined as the first order to pick after picking order $i$ that gives $g(i,V)$.

Finally, the optimal picking trip $r_m$ given $m$ orders in $U$ requires a travel distance of:

$$D_{r_m} = g(y_0, U) = \min_{j \in U} \{d_{y_0 j} + g(j, U \setminus \{j\})\} \tag{9}$$

The minimum travel distance is the distance from the current robot position plus the shortest total distance for picking all orders in $U$ starting from order $j$.

The optimal picking trip $D_{r_m}$ for set $U$ is obtained from sequencing the values of $p$. The first order to be picked from the current robot position $y_0$ is $p(y_0, U) = k_1$. This order $k_1$ is actually order $j$ that gives $D_{r_m}$ in (9). The second order to be picked is $p(y_0, U \setminus \{k_1\}) = k_2$, where $U \setminus \{k_1\}$ contains $m - 1$ orders. The third order to be picked is $p(y_0, U \setminus \{k_1, k_2\}) = k_3$, where $U \setminus \{k_1, k_2\}$ contains $m - 2$ orders. This way of working is continued until a picking trip $r_m = \{k_1, k_2, k_3, ..., s\}$ has been constructed, which gives the sequence of order picking for the robot.

The recursion is only used for solving a TSP with length $m > 1$, that is, $U \subseteq R$ and $|U| > 1$. For $m = 1$ the solution is picking target order $s \in U$.

Note that the above description of dynamic programming is based on the assumption that each order requires exactly one product. In this case, each product pallet is uniquely identified by the order ID to which the product belongs. If an order consists of multiple products, each product pallet can be identified uniquely by the order ID and the position of each product pallet for that order at the buffer.