Research Article

Order Picking Optimization in Carousels Storage System

Xiong-zhi Wang, 1 Guo-qing Wang, 2 and Xiao-chun Li 3

1 College of Economics and Management, South China Agriculture University, Guangzhou 510642, China
2 Department of Business Administration, Jinan University, Guangzhou 510632, China
3 Department of Information & Engineering, South Chinese Normal University (Nanhai Campus), Nanhai, Guangdong 528225, China

Correspondence should be addressed to Xiong-zhi Wang; flyhawkren@scau.edu.cn

Received 28 March 2013; Accepted 2 September 2013

1. Introduction

Carousels storage system is commonly referred to as an automated computer controlled system which is widely used to store and pick small, light, and highly demanded items, as an effective warehousing facility. It has been used in links of the supply chain: by manufacturers, distributors, and retailers. Standard carousel system consists of a carousel which may hold many different products stored in bins that rotate along a closed loop and a picker (either human or robotic) who occupies a fixed position at the front of the carousel (see Figure 1). To retrieve a product, the carousel system automatically rotates (clockwise or counterclockwise) the bin with the requested product to the position of the picker. Accordingly, it is referred to as a product-to-picker system. The operator may effectively use the rotation time of the carousel for activities such as sorting, packaging, or labeling of the retrieved goods. To expand the merits and further reduce the waiting time for a picker, multiple carousels are integrated so that while a carousel is for picking, others can convey items simultaneously. Some of its advantages include increased efficiency derived from bringing parts to picker instead of sending picker to parts, high speed storage and retrieval, low installation and operating cost, and efficient space utilization.

While companies are finding it important to optimize the operation of their carousel systems to gain a competitive edge in such environments, there are quite a few technical publications on the planning and control of carousels. Bartholdi III and Platzman [1] consider sequencing of picks in a single order. They assume that the time needed by a (robotic) picker to move between bins within the same carrier (or shelf) is negligible compared to the time to rotate the carousel to the next carrier (or shelf). This assumption reduces the problem to finding the shortest Hamiltonian path on a circle. They present a linear time algorithm that finds an optimum solution. Wen and Chang [2] also consider sequencing picks in a single order. They assume that the time to move between bins within the same carrier or shelf may not be neglected. They present three heuristics for this situation, based upon the algorithm in Bartholdi III and Platzman [1]. Several authors have considered the situation where the order-picker consecutively picks multiple orders thereby completing all picks in an order before commencing...
2 Mathematical Problems in Engineering

In this paper, we focus on the deterministic single order picking problem in a multiple carousels system with a single order picker in which carousels are arrayed in the picking area, each with one picking station, and are independently controlled. We will present a formal problem description in Section 2. In Section 3, the problem under study is proved to be strongly NP-hard. Some classical characteristics are put forward in Section 4 and an approximation algorithm is constructed for a special case (double carousel system) of the problem in Section 5. In Section 6, an improved nearest neighbor heuristics (INIH) is designed. We present our conclusions in the last section.

2. Problem Description

In the multiple carousels system depicted in Figure 2, there are multiple identical carousels, in which each one rotates individually either clockwise or counterclockwise. There is only one picker moving among the front of each carousel performing picking operations so as to collect all items from the picking points to the I/O (Input/Output) station. It is obvious to find that the picking time of the picker has to be considered explicitly, which is different from single carousel order picking problems. We are given an order with \( m \) items randomly placed in the system. The objective is to find a picking scheduling to minimize the total picking time. We assumethe following.

1. Each bin stores only one type of item without shortage during picking operation. There may be a few bins in each column, but the moving time among bin levels is negligible.

2. Rotation speed of the carousels is constant. It means that there is no acceleration or retarding time considered.

3. The capacity of the picker is large enough to contain the total number of items of any order.

The notation used throughout this paper is defined as follows:

- \( q \): number of the carousels;
- \( a \): the distance between any two adjacent carousels;
- \( n \): number of columns in each carousel (assuming that the distance between any two adjacent columns is 1);
- \( m_i \): number of items to be picked in carousel \( i \) in one order, \( i = 1, 2, \ldots, q \) and \( m_1 + m_2 + \cdots + m_q = m \);
- \( I_i = \{I_{i1}, I_{i2}, \ldots, I_{im_i}\} \): set of items in carousel \( i \);
- \( I_{ij} \): location of item \( I_{ij} \) in carousel \( i \);
- \( p_{ij} \): picking time of item \( I_{ij} \);
- \( \sigma = \{[1], [2], \ldots, [m]\} \): a feasible sequence of order picking, in which \([i]\) is denoted as the \( i \)th item picked;
- \([j^*(k)]\): the last item picked in the \( k \)th carousel before item \([j]\) is picked;
- \( t_{[i]j} \): the travel time from item \([i]\) to item \([j]\);
- \( T \): the total picking time when all items required are picked;
- \( v_1 \): moving speed of the picker along the picking points (it is easy to know that the moving time \( \tau \) is \( a/v_1 \) between two adjacent carousels);
- \( v_2 \): relative rotation speed of carousels, without loss of generality we normalize \( v_2 \) as 1.

By using a mapping process, we can obtain an exact two-dimensional array \((x, y)\) for all storage units in carousels where \( x \) indicates the carousel number and \( y \) represents which column the unit locates in that carousel, \( x = 1, 2, \ldots, q; y = 1, 2, \ldots, n \). Therefore, item \( j \) location is denoted as \((x_j, y_j)\) in the system.

As each carousel rotates individually either clockwise or counterclockwise, the travel time from item \( i \) to item \( j \) in the same carousel (denoted as \( H_{ij} \)) is

\[
H_{ij} = \min \left\{ \left| y_j - y_i \right|, n - \left| y_j - y_i \right| \right\}.
\]
And if item \( i \) and item \( j \) do not exist in the same carousel, the travel time from carousel where item \( i \) locates to another carousel where item \( j \) locates (denoted as \( V_{ij} \)) is

\[
V_{ij} = |x_i - x_j| \cdot r. \tag{2}
\]

Note that the array \((x, y)\) will change dynamically after an item is picked by rotating to the picking point. For example, if one item located at \((x, y)\) is picked, another item location at \((x, y')\) in the same carousel will become \((x, y''=y'+n-y)\) or \((y''=y'-y)\) if \( y' < y \) or \( y' > y \).

Assume there is a feasible sequence \( \sigma = [[1], [2], \ldots, [l-1], [l], \ldots, [m]] \), the operation time of picking item \([k]\) after \([k-1]\) (denoted as \( t_{[l-1][l]} \)) is

\[
t_{[l-1][l]} = \max \left\{ V_{[l-1][l]}, H_{i^*}[k_{(0)}][l][l] \right\} = \max \left\{ |x_{[l-1][l]} - x_{[l][l]}| \cdot r, \right. \\
\left. \min \left( \left( y_{[l-1][l]} - y_i[1], n - \left( y_{[l][l]} - y_i[1] \right) \right) \right) \right\}. \tag{3}
\]

And the total operation time until all items are picked (denoted as \( T \)) is \( T = \sum_{l=1}^{m} t_{[l-1][l]} \). Hence, the problem studied in the paper is to minimize \( T \) by finding the best feasible sequence \( \sigma^* \) of the order required. It can be simply modeled as follows:

\[
\begin{align*}
\min \quad & T = \sum_{l=1}^{m} t_{[l-1][l]} \\
\text{s.t.} \quad & t_{[l-1][l]} = \max \left\{ |x_{[l-1][l]} - x_{[l][l]}| \cdot r, \right. \\
& \left. \min \left( \left( y_{[l-1][l]} - y_i[1], n - \left( y_{[l][l]} - y_i[1] \right) \right) \right) \right\} \tag{4}
\end{align*}
\]

When \( q = 2 \), it is a special case of the problem. We denote the problem under study by \( P \) and the special case by \( SP \).

### 3. NP-Hardness

In this section we first prove that the special case \( SP \) is strongly NP-hard by a reduction from the 3-partition problem, which is known to be NP-complete in the strong sense. Then, it is easy to know that the general problem \( P \) is NP-hard because \( SP \) is a case of \( P \).

#### 3.1. 3-Partition

Given positive integers \( r, B \) and a set of integers \( A = \{b_1, b_2, \ldots, b_p\} \) with \( \sum_{k=1}^{p} b_k = rB \) and \( B/3 < b_k < B/2 \) for \( k = 1, 2, \ldots, r \). Does there exist a partition \( \{A_1, A_2, \ldots, A_r\} \) such that \( |A_i| = 3, \sum_{b_j \in A_i} b_j = B \) and \( l = 1, 2, \ldots, r \)?

**Theorem 1.** Problem \( SP \) is strongly NP-hard.

**Proof.** For any given instance of 3-partition, construct an instance of \( SP \) as follows:

\[
n = 3r^2B, \quad m_1 = 4r, \quad m_2 = r, \]
\[
\tau = 0, \quad v = 0, \quad y = r(2rB + B + 1), \]
\[
l_{ij} = 0, \quad t_{ij} = b_j, \quad j = 1, 2, \ldots, 3r, \]
\[
l_{ij} = 1, \quad t_{ij} = 2rB, \quad j = 3r + 1, \ldots, 4r, \]
\[
l_{ij} = j(2rB + B + 1), \quad t_{ij} = B, \quad j = 1, 2, \ldots, r.
\]

Without loss of generality, assume that \( r < \min_{1 \leq i \leq 3r} |b_i| \). Call items \( I_{ij} (j = 1, 2, \ldots, r) \) partition items and remaining items enforcing items. We now show that there exists a solution such that \( T(\sigma) \leq y \) for this instance if and only if there exists a desired 3-partition.

**If Part.** Suppose that there exists a desired solution \( \{A_1, A_2, \ldots, A_r\} \) for 3-partition. Let \( \{Q_1, Q_2, \ldots, Q_s\} \) be the corresponding partition item sets. Construct a picking sequence \( \sigma = \{Q_1, I_{1,3r+1}, I_{2,3}, \ldots, Q_s, I_{1,4r}, I_{2,4r}\} \). It is easy to see that \( T(\sigma) = y \).

**Only If Part.** Let \( \sigma \) be a solution such that \( T(\sigma) \leq y \). Let \( R_3 \) and \( T_3 \) be the minimum total rotating time and total picking time of carousel 2, respectively. It is easy to see that \( R_3 + T_3 = y \), which means that carousel 2 has no idle time in \( \sigma \). Let \( A_i \) be the set of all items picked on carousel 1 during time period while carousel 2 rotates from \( I_{2, j-1} \) to \( I_{2, j} \) in \( \sigma \). To keep carousel 2 from waiting for the picker, we know that \( \sum_{i \in A_j} P_{lk} \leq rB + B + 1 \).

Let \( T_1 \) be the total picking time on carousel 1. Then it is clear that there are one enforcing item and at most 3 partition items in \( A_j \), and the total picking time of the partition items \( P_{lk} \leq B \). On the other hand, if \( P_{lk} < B \), for any \( j = 1, 2, \ldots, r \), it means that at least one partition item is picked after picking the last item in carousel 2. Then \( T(\sigma) > y \), a contradiction. \( \square \)

**Theorem 2.** Problem \( P \) is strongly NP-hard.

\( SP \) is a case of \( P \), and \( SP \) is strongly NP-hard. It is easy to know that problem \( P \) is strongly NP-hard.

### 4. Classical Characteristics

**Lemma 3.** Considering the bidirectional single carousel, the optimal picking sequence switched direction at most once \( [1] \).

Supposing there are \( k \) items to be picked in a certain carousel, three types of picking paths will exist as below (see Figure 3). It is obvious that the result of Lemma 3 can facilitate depriving the optimal sequence (linear time complexity method) for a single carousel system.
(1) Travel in clockwise or counter clockwise direction until all $k$ items are picked (see Figures 3(a) and 3(b)).

(2) Travel from location 0 to $i$ in a counterclockwise direction; then backward pass location from 0 to $i + 1$ in a clockwise direction (see Figure 3(c)).

(3) Travel from location 0 to $i + 1$ in a clockwise direction; then backward pass location from 0 to $i$ in a counterclockwise direction (see Figure 3(d)).

**Lemma 4.** If there is an optimal sequence in multiple carousels system, the subsequence of one certain carousel is also optimal.

**Proof.** Prove it by reduction to absurdity. Let sequence $\sigma$ be a feasible solution of the problem in which any subsequence on the specified carousel is optimal with the total operation time $T_\sigma$ in the problem. Consider

$$\sigma = \{ [1], [2], \ldots , [l_{i-1}^k], [l_i^k], [l_{i+1}^k], \ldots , [m^k] \}. \quad (6)$$

And the relevant total operation time $T_\sigma$ is

$$T_\sigma = \sum_{[j]=1}^{[m]} t_{[j-1][j]}$$

$$= \sum_{[j]=1}^{[l_{i-1}^k]} \max \left( \left| x_{[j-1]} - x_{[j]} \right| * \tau, H_{[j^k(k_{i-1})][j]} \right)$$

$$+ t_{[l_{i-1}^k][l_i^k]}$$

$$+ \sum_{[j]=1}^{[l_i^k]} \max \left( \left| x_{[j-1]} - x_{[j]} \right| * \tau, H_{[j^k(k_{i+1})][j]} \right)$$

$$+ t_{[l_i^k][l_{i+1}^k]}$$

$$+ \sum_{[j]=1}^{[l_{i+1}^k]} \max \left( \left| x_{[j-1]} - x_{[j]} \right| * \tau, H_{[j^k(k_{i+1})][j]} \right), \quad (7)$$

where the superscript $k$ on letter $l$ (the same below) indicates the carousel number.

The subsequence $\sigma^k$ on the $k$th carousel is

$$\sigma^k = \{ [l_1^k], [l_2^k], \ldots , [l_i^k], [l_{i+1}^k], \ldots , [l_{m^k}] \}. \quad (8)$$

Assume that there exist an optimal solution in which the subsequence on the specified carousel is not optimal, such as the subsequence on the $k$th carousel is $(\sigma^k)'$. Consider

$$\left( \sigma^k \right)' = \{ [l_1^k], [l_2^k], \ldots , [l_{i-1}^k], [l_i^k], \ldots , [l_{m^k}] \}. \quad (9)$$

And the overall optimal solution $\sigma'$ is

$$\sigma' = \{ [1], [2], \ldots , [l_{i-1}^k], [l_i^k], [l_{i+1}^k], \ldots , [m] \}, \quad (10)$$
with the total operation time $T_{\sigma^*}$,

$$T_{\sigma^*} = \sum_{j=1}^{[m]} t_{[j-1][j]}$$

where $t_{[j-1][j]}$ is the operation time from position $[j-1]$ to $[j]$. The total operation time is the sum of all operation times.

Through the above analysis of two cases, we can know that this conclusion of Lemma 4 is established.

(1) The position of item $[k_1]$ is adjacent to item $[k_1 + 1]$ in the global optimal sequence.

(1) Items $[k_1]$ and $[k_1 + 1]$ locate in the same carousel. Thus, we can derive that $T_{\sigma^*} \geq T_{\sigma}$ from formulas (7) and (11), assumption contradiction.

(2) The position of item $[k_1]$ is not adjacent to $[k_1 + 1]$ in the global optimal sequence. In other words, $[k_1]$ and $[k_1 + 1]$ are not in the carousel where $[k_1 + 1]$ locates.

(2) The position of item $[k_1]$ is not adjacent to $[k_1 + 1]$ in the global optimal sequence. In other words, $[k_1]$ and $[k_1 + 1]$ are not in the carousel where $[k_1 + 1]$ locates.

Therefore, $T_{\sigma^*}$ is not adjacent to $[k_1 + 1]$ locates.

Hence, $T_{\sigma^*} \geq T_{\sigma}$ from formulas (7) and (11), assumption contradiction.

As a result, we can derive that $T_{\sigma^*} \geq T_{\sigma}$ from formulas (7), (11), (15), and (16), assumption contradiction.

Through the above analysis of two cases, we can know that this conclusion of Lemma 4 is established.
In light of Lemma 4, we notice that the amount of the feasible solutions will decrease sharply to be \( N! \prod_{i=1}^{m} k_i! \) given an order of sequencing \( N \) items, where \( m \) denotes amount of carousels in the system and \( k_i \) represents the item quantity required in the \( i \)th carousel \((i = 1, 2, \ldots, m, \sum_{i=1}^{m} k_i = N)\). As a result, this conclusion can help us devise better or optimal solution by combining \( m \) optimal subsequences of each of the carousels which is easy to find.

5. Heuristics

5.1. A Dynamic Programming Algorithm for a Special Case (Double Carousel System). In the double carousel system, there are two identical carousels. Each carousel rotates individually either clockwise or counterclockwise. There is only a single picker that performs the picking operations. In this section, we develop a dynamic programming algorithm (DPA) for fixed picking sequences on two carousels. Assume that \( \text{arc}(i, i + 1) \) (see Figure 4) is not visited by the optimal path. Then there are two possible forms of the optimal path.

1. Travel from 0 to \( i \) in a counterclockwise direction; then backward pass from 0 to \( i + 1 \) in a clockwise direction (see Figure 4(a)).

2. Travel from 0 to \( i + 1 \) in a clockwise direction; then backward pass from 0 to \( i \) in a counterclockwise direction (see Figure 4(b)).

The optimal path on each carousel can be found by testing each of \( m \) distinct arcs with the minimum distance, which is \( \min(n - \max[l_{i+1} - 2l_i, 2l_i - n - l_{i+1}]) \).

For fixed picking paths on each carousel, an optimal picking path can be obtained with the following dynamic programming method:

1. determine the optimal picking sequence for each carousel;
2. find the dominant carousel with longer picking time, and fix the picking schedule for the carousel;
3. schedule operations for another carousel as soon as possible with the constraint of the dominant schedule.

5.2. INIH for General Problem. Based on the previous analysis of two classical characteristics, we develop an improved nearest items heuristics (INIH) algorithm to solve the general problem in which all items required are stored randomly in \( q \) carousels.

Let ordinal set \( S_i \) \((i = 1, 2, \ldots, m)\) be the optimal sequence of the \( i \)th carousel, where \( |S_i| = k_i, S_i = \{i_1, i_2, \ldots, i_{k_i}\} \), and the coordinate of element \( [i_l] \) is \((x_{[i_l]}, y_{[i_l]}) \) \((l = 1, 2, \ldots, k_i)\) (see Algorithm 1).

The sequence of all elements of the ordinal set \( S_i \) is the order picking solution. It can improve the operation’s efficiency and search space of the general nearest items heuristics effectively so as to get better effect. It is easy to know that the complexity of INIH is linear of its scale according algorithm complexity theory [5]. And it is acceptable on computing time.

6. Performance Study

To explore the computational performance of DPA and INIH algorithms comparing other methods such as conventional search (CS) and nearest item heuristics (NIH), we construct two groups of tests on personal computer PIII 1.5 G Hz by C language. Consider a practical multiple carousels system with the operational characteristics specified as follows:

The two groups of items request: \((10, 15, 20, 30, 50)\) and \((100, 150, 200, 300, 500)\); items located on carousel: \(x\)-coordinate is from \((2, 4, \ldots, 30)\) randomly and \(y\)-coordinate is from function \( \text{Rand}(50) \); the distance of any two adjacent bins on carousel = 0.5 m; \( \tau = 3 \) m/s, \( y = 2 \) m/s, and \( a = 2 \) m; Number of bins per carousel is 100, carousels quantity \( m = 2 \) for INIH, and \( m = 2 \) for DPA. \( R = \text{Random}(10^4), \alpha, \beta \), is from \((0, 1)\) randomly, and \( \alpha + \beta = 1 \).

Three indexes are tested: average order picking time (Ot), standard difference (SD), and algorithm operating time (Ts). The testing results are specified as Tables 1 and 2.

Based on the previous analysis of testing data, we can obtain that DPA and INIH are better than CS and NIH for multiple carousels system. And the quality of INIH and DPA’s solution is well distinctive gradually without no more computing time and standard difference when the problem becomes of larger scale and more complicated. To be summarized, the computing efficiency and effectiveness of INIH and DPA are better accepted and will support strongly in practice.

7. Conclusions

Order picking has long been identified as the most labor-intensive and costly activity for almost every warehouse or distribution center. A bad order picking can lead to unsatisfactory service and high operational cost for the warehouse and consequently for the whole supply chain. In order to operate efficiently, the order picking process needs to be robustly designed and optimally controlled. In
(1) Initialization: Get ordinal set $S_i (i = 1, 2, \ldots, m)$; Define ordinal and general set $S_0$ and $S$, respectively; set variable $q$ 
$q \leftarrow 0$, $S_0 = \emptyset$; 
For $i = 1$ to $m$ to 
Get the first element [$i_1$] of $S_i$, $S \cup [i_1]$; 
End

(2) Compute $t_{q,k} = \max \{||x_q - x_k||/v_1, \min \{y_k, n - y_k\}/v_2\}, k \in S$; 
Get $t_{q,k}^* = \min_{k \in S} (t_{q,k}), S_0 = S_0 \cup k^*$; 
$q \leftarrow k^*$;

(3) Find the ordinal set $S_j$ of the element $k^*$; 
If $S_j = \emptyset$, Return step (2); 
Otherwise $S_j = S_j \backslash k^*, S = S \backslash k^*$; 
Get the first element [$j_l$] of $S_j$, $S \cup [j_l]$;

(4) If $S = \emptyset$, Output the ordinal set $S_0$; Otherwise, Return step (2);

**Algorithm 1: Improved nearest items heuristics (INIH).**

**Table 1: Testing performance of INIH.**

<table>
<thead>
<tr>
<th></th>
<th>$N$</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>30</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
<th>300</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>Ot</td>
<td>12.28</td>
<td>24.56</td>
<td>30.82</td>
<td>60.56</td>
<td>92.25</td>
<td>198.5</td>
<td>326.3</td>
<td>428.2</td>
<td>619.2</td>
<td>902.2</td>
</tr>
<tr>
<td></td>
<td>SD</td>
<td>10.2</td>
<td>12.26</td>
<td>16.64</td>
<td>20.2</td>
<td>26.62</td>
<td>32.36</td>
<td>40.24</td>
<td>50.25</td>
<td>56.2</td>
<td>68.0</td>
</tr>
<tr>
<td></td>
<td>Ts</td>
<td>0.024</td>
<td>0.032</td>
<td>0.081</td>
<td>0.168</td>
<td>0.298</td>
<td>1.212</td>
<td>2.324</td>
<td>4.486</td>
<td>8.02</td>
<td>10.96</td>
</tr>
<tr>
<td>NIH</td>
<td>Ot</td>
<td>11.3</td>
<td>12.21</td>
<td>25.86</td>
<td>48.24</td>
<td>80.56</td>
<td>160.2</td>
<td>281.4</td>
<td>362.4</td>
<td>520.8</td>
<td>783.4</td>
</tr>
<tr>
<td></td>
<td>SD</td>
<td>9.98</td>
<td>11.86</td>
<td>15.68</td>
<td>18.02</td>
<td>24.64</td>
<td>28.97</td>
<td>35.86</td>
<td>42.67</td>
<td>49.2</td>
<td>62.2</td>
</tr>
<tr>
<td></td>
<td>Ts</td>
<td>0.028</td>
<td>0.039</td>
<td>0.088</td>
<td>0.172</td>
<td>0.202</td>
<td>1.221</td>
<td>2.328</td>
<td>4.498</td>
<td>8.701</td>
<td>10.99</td>
</tr>
<tr>
<td>INIH</td>
<td>Ot</td>
<td>8.28</td>
<td>11.12</td>
<td>16.36</td>
<td>33.64</td>
<td>59.41</td>
<td>102.3</td>
<td>242.4</td>
<td>312.8</td>
<td>468.9</td>
<td>689.5</td>
</tr>
<tr>
<td></td>
<td>SD</td>
<td>9.1</td>
<td>10.04</td>
<td>14.98</td>
<td>15.54</td>
<td>19.52</td>
<td>23.24</td>
<td>29.12</td>
<td>36.64</td>
<td>41.1</td>
<td>59.2</td>
</tr>
<tr>
<td></td>
<td>Ts</td>
<td>0.021</td>
<td>0.038</td>
<td>0.079</td>
<td>0.156</td>
<td>0.209</td>
<td>1.208</td>
<td>2.23</td>
<td>4.308</td>
<td>8.021</td>
<td>9.02</td>
</tr>
</tbody>
</table>

**Table 2: Testing performance of DPA.**

<table>
<thead>
<tr>
<th></th>
<th>$N$</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>30</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
<th>300</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>Ot</td>
<td>2.26</td>
<td>4.16</td>
<td>5.52</td>
<td>10.28</td>
<td>15.14</td>
<td>30.48</td>
<td>45.32</td>
<td>66.46</td>
<td>101.1</td>
<td>168.2</td>
</tr>
<tr>
<td></td>
<td>SD</td>
<td>3.21</td>
<td>4.26</td>
<td>5.63</td>
<td>6.21</td>
<td>8.62</td>
<td>10.36</td>
<td>13.24</td>
<td>16.75</td>
<td>18.24</td>
<td>23.01</td>
</tr>
<tr>
<td></td>
<td>Ts</td>
<td>0.003</td>
<td>0.004</td>
<td>0.011</td>
<td>0.038</td>
<td>0.041</td>
<td>0.092</td>
<td>0.201</td>
<td>0.516</td>
<td>1.021</td>
<td>1.261</td>
</tr>
<tr>
<td>NIH</td>
<td>Ot</td>
<td>2.34</td>
<td>3.81</td>
<td>4.26</td>
<td>8.04</td>
<td>10.16</td>
<td>20.21</td>
<td>35.04</td>
<td>43.42</td>
<td>65.20</td>
<td>97.45</td>
</tr>
<tr>
<td></td>
<td>SD</td>
<td>3.18</td>
<td>4.06</td>
<td>5.18</td>
<td>6.02</td>
<td>7.64</td>
<td>9.87</td>
<td>12.26</td>
<td>15.24</td>
<td>16.21</td>
<td>20.21</td>
</tr>
<tr>
<td></td>
<td>Ts</td>
<td>0.003</td>
<td>0.004</td>
<td>0.010</td>
<td>0.032</td>
<td>0.040</td>
<td>0.086</td>
<td>0.197</td>
<td>0.498</td>
<td>0.901</td>
<td>1.209</td>
</tr>
<tr>
<td>DPA</td>
<td>Ot</td>
<td>2.18</td>
<td>3.22</td>
<td>4.06</td>
<td>7.64</td>
<td>9.01</td>
<td>15.32</td>
<td>29.41</td>
<td>33.84</td>
<td>42.89</td>
<td>68.54</td>
</tr>
<tr>
<td></td>
<td>SD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Ts</td>
<td>0.001</td>
<td>0.003</td>
<td>0.009</td>
<td>0.031</td>
<td>0.038</td>
<td>0.084</td>
<td>0.198</td>
<td>0.498</td>
<td>0.998</td>
<td>1.801</td>
</tr>
</tbody>
</table>

In this paper we formulate the order picking optimization of multiple carousels system and give the mathematical model. According to the characteristics, an approximation algorithm (DPA) is presented for the special case of two carousels, and an improved nearest neighbor algorithm (INIH) is constructed for general problem. Finally, simulation tests are developed on the two heuristics. Experimental results verify that the optimum solutions are quickly and steadily achieved and show their better performance.

As a generalization of our problem, there are a number of issues with different layout and rotating modes which are of interest for further research, for example, the multiple orders sequencing problem in the same environment. It is interesting to investigate that.

**Acknowledgments**

This work is supported by Science Foundation of Ministry of Education of China (09YJC630088) and Fund of the third phase (2010B, 20111211QN07) project of "211 Engineering" program on National Key Disciplines Development.
in College of Economics and Management, South China Agricultural University.

References


