logistics

Article

A Puzzle-Based Sequencing System for Logistics Items

Raji Alahmad * and Kazuo Ishii

Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology, 2-4 Hibikino, Wakamatsu, Kitakyushu 808-0196, Fukuoka, Japan; ishii@brain.kyutech.ac.jp
* Correspondence: alahmad.raji941@mail.kyutech.jp

Abstract: Background: The new demands of the current market including for space should be satisfied by designing modern material flow systems. Designing warehouses using effective material handling equipment significantly supports cost reduction and efficient space utilization. Sequencing of items is an important process that leads to enhanced logistics operations. Current approaches are not capable of fully fulfilling dynamic changes. Methods: In this paper, a puzzle-based sequencing system with a high density and highly efficient floor space utilization was successfully developed. Accordingly, two solving methods were investigated: game tree and pathfinding algorithms. A-star was chosen based on pathfinding algorithms in order to find the shortest solution of the puzzle in which the sequencing time was decreased. The pre-sorting strategy was proposed to overcome the unsolvable configuration issue that cannot be solved by the aforementioned methods. Moreover, the shape of the puzzle was considered. Results: Based on numerical calculations, we found that a square shape was better than a rectangle in terms of solution steps, and we confirmed the direct relationship between the aspect ratio and rectilinear distance, which directly affects the pre-sorting steps. Conclusion: Our results prove that the puzzle-based sequencing system should be highly preferred for effective floor space utilization compared to the current systems.

Keywords: sequencing; 8-puzzle; A-star algorithm

check for updates

Citation: Alahmad, R.; Ishii, K. A Puzzle-Based Sequencing System for Logistics Items. *Logistics* **2021**, *5*, 76. https://doi.org/10.3390/logistics5040076

Academic Editor: Robert Handfield

Received: 1 September 2021
Accepted: 25 October 2021
Published: 28 October 2021

1. Introduction

Logistics operations can be elucidated by several fixed assets: warehouses, depots, transport, and material handling. The number and size of these assets are important factors in effective logistics planning [1]. The capital and operating costs of warehouses embody 23% of logistics costs in the U.S., and 39% in Europe [2]. Two types of warehouses can be categorized: distribution warehouses, where the products are collected from the point of origin for delivery to consumers, and production warehouses, where the raw materials and semi-finished products of production facilities are stored [3]. The proper design of warehouses is one of the most important factors affecting space utilization, efficiency, and cost [4,5]. The effective use of space is a goal for almost every company located near population centers, where high space charges and limited availability of real estate are the main concerns [6]. Smaller warehouse systems decrease the overall costs, since they are less expensive to build [7].

Material handling is the movement of raw materials and semi-finished and finished products to and from productive processes, in warehouses and receiving and shipping zones [3], and its activities consume 20% to 50% of the total operating costs. Effective material transport equipment such as rollers, wheels, and sorting conveyors lead to significant cost reductions and efficient space utilization [8,9]. For efficient warehousing (i.e., put-away, storage, and order picking), an automatic store and retrieval system (ASRS) is typically used [10]. In an ASRS, cranes operate in parallel and feed the pallet building workstation; therefore, the robotic palletizer receives a random sequence of items that should be re-sequenced [11]. Referring to the systems that are applied in real-world warehouses, the items are mostly released from ASRS in random sequence [12,13]. Thus, they need either optimized release (which is still under research and development [13]) or item re-sequencing after retrieval for better performance, especially during peak-hours, where a lack of workforce and other new technologies are highly required at the packing stations for timely release of the lanes.

Furthermore, mixed model assembly lines have become common in the automotive industry, and the efficiency of the final assembly depends on the sequence of vehicles being built [10]. Typically, two types of systems are used to re-sequence the incoming random items: (a) a temporary storage system that uses parallel lanes [10], and (b) a sortation conveyor, where the items keep looping until they are in the desired sequence [14]. Poor floor space utilization is one of the disadvantages of such systems [11]; therefore, a material handling device with a high-density system is required. Generally, the term density in logistics is used for storage density, which is the ratio of the space dedicated for storage to the total storage space including locations such as aisles [15]. However, in this paper, we defined the density as the areal density, which is the ratio of items to the total material handling device space.

Many studies have considered high-density systems in order to enhance the efficiency of logistics processes. The sliding puzzle was invented by Sam Loyd in the 1870s [7], and is also known as the 15-puzzle, and later, the general version ($n^2 - 1$) became a popular and interesting subject for logistics researchers, especially in developing storage systems. Gue [6] developed a new concept based on a puzzle game: a very high-density storage system (HDSS) for physical goods with an efficient algorithm for filling densely rectangular storage areas. Later, Gue and Kim [7] developed an algorithm for the retrieval of items in a puzzle-based storage system (PBSS). They experimentally compared puzzle-based with traditional aisle-based storage. The results showed that the puzzle-based system was superior, with multiple escorts regarding the retrieval time, if the storage density was less than 90%. In [16], Kota et al. extended the analytical results of retrieval time in PBSS to determine the retrieval time performance when multiple escorts are randomly located within the system. The GridStore system was developed by Gue et al. [17] in order to overcomes the inflexibility of automated material handling systems for HDSS by implementing decentralized control. In GridStore, an arbitrary number of requests could be retrieved by allowing simultaneous item moving. The major drawback of this system is the capability of delivering items to only a single side. However, Uludag [18] solved this limitation by developing a puzzle-based order picking system called GridPick. In the GridPick system, the orders can be picked from two sides of the grid, allowing for higher throughput and an improved use of space compared to single-sided systems. A further improvement was achieved by Gue and Hao [19]. They developed a new system called GridHub, which was able to transfer orders in four directions simultaneously within the grid. Subsequently, Hao [20] developed the NU GridHub system to handle bigger boxes in which one box can occupy more than one conveyor module. Further modification of GridHub was conducted by Ashgzari and Gue [21]. In the new method, GridPick+, several limitations of GridPik were addressed. For instance, GridPick+ allowed the requested items to be delivered into specific picking positions on the edge of the grid. Moreover, the use of the sequencing function allowed multiple orders to be processed simultaneously. An algorithm for moving several items at the same time in grid-based storage was designed by Yalcin et al. [15] by avoiding the items' conflict. Their experimental results demonstrated that for storage, the pushback strategy achieved the shortest time and distance, and the puzzle-based retrieval strategy was most efficient. Yalcin et al. [22] also addressed the problem of item retrieval from puzzle-based storage with a minimum number of item moves. In this work, they proposed an exact search algorithm with several search-guiding estimate functions. Additionally, they discussed the configurations with multiple empty cells located in the grid with different grid sizes.

In recent research, Shirazi and Zolghadr [23] developed an algorithm for item retrieval for HDSS. This method guaranteed the deadlock freeness in the algorithm and discussed different puzzle sizes with a dissimilar number of empty cells. It was observed that increasing empty cells up to three cells will increase the average retrieval movement, while increasing the empty cells above three will decrease the average retrieval movement sharply. Further research was carried out to formalize arranging smart boxes into an autonomous delivery vehicle [24]. The authors proposed the snake-line concept utilizing the puzzle arrangement to find the tradeoff between space and access rapidity and were able to guarantee the boxes moving continuously with minimum movement.

The system we proposed in this paper was compared with the high-density systems described in the literature, as illustrated in Table 1. The used system, function, contribution, and system areal-density are listed in the table to distinguish these works.

Table 1. Comparison among the proposed system and other high-density systems from the literature.

Paper	System	Function	Contribution	System Areal-Density for 35 Boxes
Gue and Kim [7]	NAVSTORS system	Storing, retrieval	Describe the relationship between storage density and expected retrieval time	94.4% with two escort
Gue et al. [11]	GridSequence	Sequencing	High density, decentralized control algorithm	72.9%
Kota et al. [16]	Puzzle-Based system	Storing, retrieval	Determine the retrieval time performance for multi-escorts randomly located in the grid.	94.4% with two escorts [1]
Gue et al. [17]	GridStore system	Storing, retrieval	Retrieve several items by allowing simultaneous moving	≤94.4% [1]
Uludag [18]	GridPick	Storing, retrieval	Higher throughput, retrieve items to two sides of the grid	≤94.4% [1]
Gue and Hao [19]	GridHub	Storing, retrieval	Transfer orders in four directions simultaneously within a grid	≤95.45% [2]≤94.44 for 36 boxes
Hao [20]	NU GridHub	Sorting, sequencing	Delivers requested items in the desired sequence to any location	56.25% for 36 boxes
Ashgzari et al. [21]	GridPick+	Storing, retrieval	Increasing in throughput by 77%	-
Yalcin et al. [15]	Grid-based system	Storing, retrieval	Framework for the efficient storage and retrieval of items based on a multi-agent routing algorithm	Up to 97.2% [1]
Yalcin et al. [22]	PBS system	Items retrieval	Retrieve items with a minimum number of items moves	≤94.4% [1]
Shirazi et al. [23]	PBS system	Items retrieval	Deadlock prevention algorithm	Up to 97.2% [1]
Tetouani et al. [24]	Puzzle-based system	Rearrangement while Routing" strategy	Formalize arranging smart boxes in an autonomous delivery vehicle	97.2%
Proposed method	Puzzle-based sequencing system	Sequencing	High-density sequencing system, address unsolvable puzzle configuration	97.2%

[1] Since these systems involve the puzzle-based concept, the areal-density is calculated as $(n_c - e)/n_c$, where n_c is the number of grid cells and e is the number of empty spaces in the grid. [2] One rule of GridHub is that at least one empty module has to be in each column or row, and their experiment was set as a grid with 22 columns and 11 rows.

Although several studies have considered high-density and puzzle-based systems with their applications, most of them have focused on storage and item retrieval. In these systems, the items are retrieved in the desired sequence. However, under batch and/or zoning picking policy, which is applied in most online retailers' warehouses, items necessitate further processes such as consolidation and sequencing [13]. To the authors' best knowledge, very few contributions have been published in the literature that have addressed the issue of item sequencing, for instance, GridSequence, which was developed by Gue et al. [11]. The proposed system could re-sequence incoming items to feed a palletizing robot with the required sequence. The GridSequence system consists of a puzzle grid with (n × m) dimensions, plus one additional row and one additional column; thus, the whole system dimensions are (n + 1) × (m + 1). The authors showed the effect of the aspect ratio on the sequencing time in the experimental results, and suggested that the aspect ratio should be at least 10. Furthermore, adding one more additional column to the center of the grid can positively affect the system. The major drawback of this system is low space utilization, since adding rows and columns will occupy more spaces out of the grid, and decrease the density. A lower density means higher empty spaces in the grid, and an increase in the floor space usage. Thus, the density plays a key role in evaluating the utilization of floor space of warehouses (storage and other functions) in urban areas where the limited space should be utilized efficiently.

The puzzle-based sequencing system was further compared with the GridSequence system developed by Gue et al. [11] with respect to the density ratio, in order to evaluate the utilization of floor space in the sequencing system. The density can be calculated as $(n_c - \text{e})/n_c$, where n_c is the number of grid cells and e is the number of empty spaces in the grid, which is 1 in the case of Sam Loyd's puzzle [7]. In the puzzle-based system, n_c is calculated as $(n_c = i + 1)$, where i is the number of items that need to be sequenced. In the GridSequence system n_c is calculated as (n + 1) × (m + 1), where (I = n * m), and e is calculated as (e = $n_c - i$). Table 2 illustrates the density ratios in the puzzle-based system versus the GridSequence system for sequencing 8, 15, 24, and 35 items.

Table 2. The density ratio in a puzzle-based system vs. a GridSequence system.

Number of Items (i)	**Puzzle-Based System Density = $(n_c - \text{e})/n_c$**		**GridSequence System Density = $(n_c - \text{e})/n_c$**	
8	n_c = 9, e = 1	88.8%	n_c = 15, e = 7	53.3%
15	n_c = 16, e = 1	93.7%	n_c = 24, e = 9	62.5%
24	n_c = 25, e = 1	96%	n_c = 35, e = 11	68.5%
35	n_c = 36, e = 1	97.2%	n_c = 48, e = 13	72.9%

As shown in Table 2, the puzzle-based system can provide a higher density than the GridSequence system. Better space utilization is quantified, with a practical example; to sequence 35 boxes with sizes of 35 cm × 35 cm = 0.1225 m^2, GridSequence would occupy 5.88 m^2, while the proposed puzzle-based would occupy 4.41 m^2. Therefore, a puzzle-based sequencing system is recommended to reduce the space as well as reduce the cost.

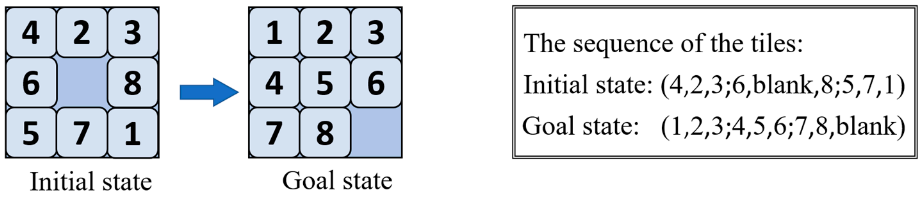
The contribution of this paper is twofold. First, we realized a high-density sequencing system based on the puzzle movement concept, with highly efficient floor space utilization concerning the minimum item movements. These points are directly related to better energy efficiency and, consequently, to lower operational costs. Second, we propose a pre-sorting strategy to overcome the unsolvable configurations of the puzzle, which obstruct the operational process. The analysis here carried out represents a tool for improving the warehouse activities in terms of both space utilization and time-consumption, in addition to minimizing the workforce.

The rest of this paper is structured as follows. Section 2 presents a puzzle-based system and solving methods, followed by a description of the A-star algorithm and puzzle solvability condition. In addition, a proposal to overcome unsolvable scenarios is presented in this section. A comparison between different shapes of the puzzle, with a discussion of the factors that affect the number of solution steps, is presented in Section 2. The results and discussion are provided in Section 3. The conclusions are given in Section 4.

2. Puzzle-Based System

2.1. Sliding Puzzle

A sliding puzzle is a single-agent sliding game consisting of (n × m) − 1 tiles and one blank, distributed in an (n × m) grid. The process for solving this is to rearrange a random configuration of numbers in the initial state by sliding the blank tile in one of four allowable moves (Up, Down, Right, and Left) to reach the goal state, which is the proper sequence of numbers [25], as shown in Figure 1.



The sequence of the tiles:
Initial state: (4,2,3;6,blank,8;5,7,1)
Goal state: (1,2,3;4,5,6;7,8,blank)

Figure 1. A 3 × 3 puzzle (8-puzzle), random configuration (**left**), goal state (**right**).

There are different shapes for such a puzzle. The ($n^2 - 1$) puzzle is a specific type, where the board is square (n × n) with ($n^2 - 1$) numbered tiles and one blank [26]. The 8-puzzle is one of the most famous ($n^2 - 1$) puzzles. The 15-puzzle and 24-puzzle are extended versions of the 8-puzzle. In the 8-puzzle, there are 9! = 362,880 different configurations, and only half of them are solvable [27]. Many researchers are interested in methods to solve such puzzles with the fewest moves (the shortest path to the solution).

2.2. Puzzle Solving Methods

There are two typical methods for finding the shortest path to the solution: game tree and pathfinding algorithms.

2.2.1. Game Tree

This method creates a tree of all configurations (states) that can be generated for the puzzle, and finds the target configuration in this tree. In the game tree, all states are represented by nodes, and the depth of the tree denotes the number of solution steps. The procedure is as follows:

1. Start tree creation from the target state;
2. Find the input node (the initial configuration) in this tree;
3. Track the path which leads to the initial node.

The game tree method could guarantee finding the shortest path to the solution. However, we might face two problems: the huge number of states that could be generated, and the scenario of searching for different targets (specific configurations).

(i) The huge number of states

Equation (1) provides the total number of nodes that could be generated in the tree for the 8-puzzle:

$$N_{States} = 1 + \sum_{i=1}^{d} b^i \,, \tag{1}$$

where N_{States} is the total number of states in the tree; b is the branching factor; and d is the depth of the tree. The branching factor is the number of nodes that could be expanded from the previous node in the tree. Figure 2 shows the concept of branching factor of the 8-puzzle.

From Figure 2, the branching factor was about 3 (when the blank tile is in the corner, there are two possible moves; when it is along edges, there are three; and when it is in the middle, there are four). Regarding the depth, Figure 3 illustrates the histogram of the solution steps for all solvable configurations of the 8-puzzle as well as the Probability

Density Function (PDF) for a normal distribution. We obtained an average solution depth of 22. The same result was confirmed with the work by Reinefeld [28].

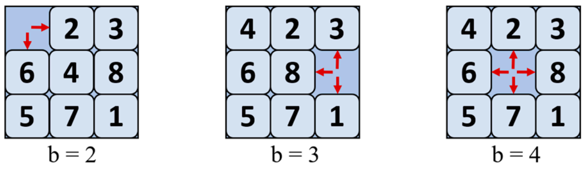


Figure 2. The 8-puzzle branching factor b.

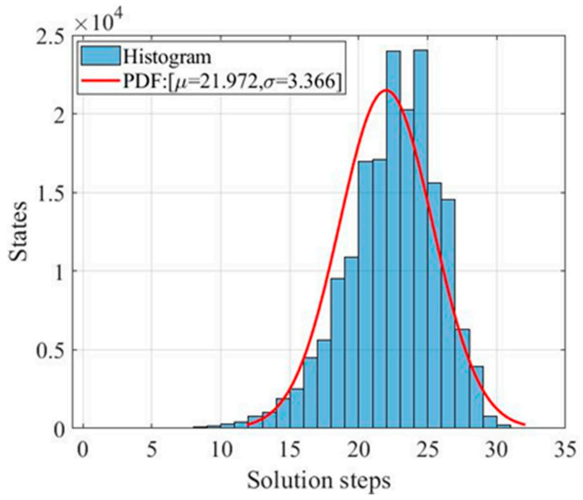


Figure 3. Histogram of solution steps for the 8-puzzle.

Referring to Equation (1), the number of nodes that could be generated for depth 22 and branching factor 3 is 3.13×10^{10} nodes. This huge number of nodes not only requires time to be generated, but is also inefficient in terms of memory [29]. By tracking the repeated states, we cut the tree down drastically into $9!/2 = 181{,}440$ nodes.

(ii) Searching for different targets

In the case of different targets, we needed to generate a tree of nodes for each goal. Thus, we had to generate $9! = 362{,}880$ trees and about 13.16×10^{10} nodes in total. One proposal to overcome the problem associated with generating such a huge number is to search for input state in the current tree using the following steps:

1. Change the desired target to the target in the current tree;
2. Apply the same changes to the input;
3. Find the new input in the current tree.

Figure 4 shows the proposal for searching in the current tree.

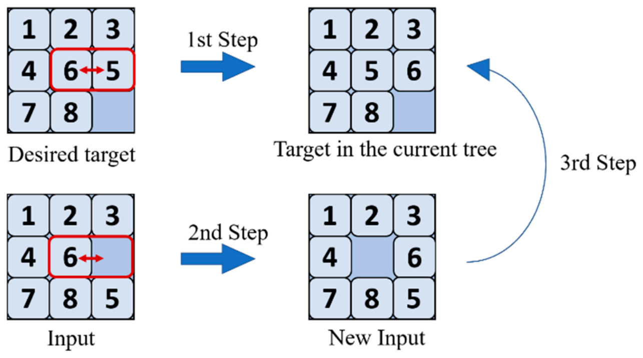


Figure 4. Concept of searching in the current tree.

Since tiles changing might give unsolvable configurations, this method will not work for all the cases in our system. The solvability of the puzzle is an important concept, therefore, the solvability condition will be discussed in Section 2.3.

2.2.2. Pathfinding Algorithms

To reach the puzzle solution, pathfinding algorithms can be applied by creating a tree of puzzle configurations (nodes), starting from the initial state until the goal state is matched, and then tracking back to the path, which leads to the goal. When reaching the goal state (node), the process of node creation will stop; therefore, generating a huge number of nodes can be avoided. There are two different types of pathfinding algorithms:

(i) Uninformed algorithms (blind algorithms)

Such algorithms work without using any external information to guide the agent to reach the goal state. Breadth-first search (BFS), depth-first search (DFS), and iterative deepening depth-first (IDDF) are some such algorithms [29,30].

(ii) Informed algorithms

In these algorithms, some information can be used to lead the algorithm and direct it to achieve a better performance. Greedy, A-star (A*), and the iterative deeping A-star (IDA*) algorithm are the most common pathfinding algorithms [29,30].

Among the algorithms that extend search paths from the root, A-star is optimally efficient [30,31]. Hence, A-star was the core algorithm in this study.

2.3. A-Star Algorithm

In the A-star algorithm (A*), the nodes can be evaluated using the cost function (function (Equation (2)), which is the sum of two factors: the heuristic function, which estimates how close the current node is to the goal, and the cost from the initial node to the current one [32].

$$f(n) = g(n) + h(n), \tag{2}$$

where f(n) is the evaluation function for the A* algorithm; g(n) is the cost from the initial node to the current node n; and h(n) is the estimated cost from the node n to the target.

The estimation function used in this research was the Manhattan distance, since it showed better performance for the informed search techniques [29,31]. The Manhattan distance or city block distance is the absolute vertical and horizontal distance between the tile in the current configuration and its appearance in the goal configuration. Figure 5 shows the layout of the A-star algorithm for solving the n-puzzle with the fewest solution steps.

The A-star algorithm allows us to avoid many nodes that should not be selected, avoiding the waste of time caused by searching a large number of useless nodes. The whole search process has strong directionality [33]. Figure 6 illustrates the implementation of the A-star algorithm for the 8-puzzle.

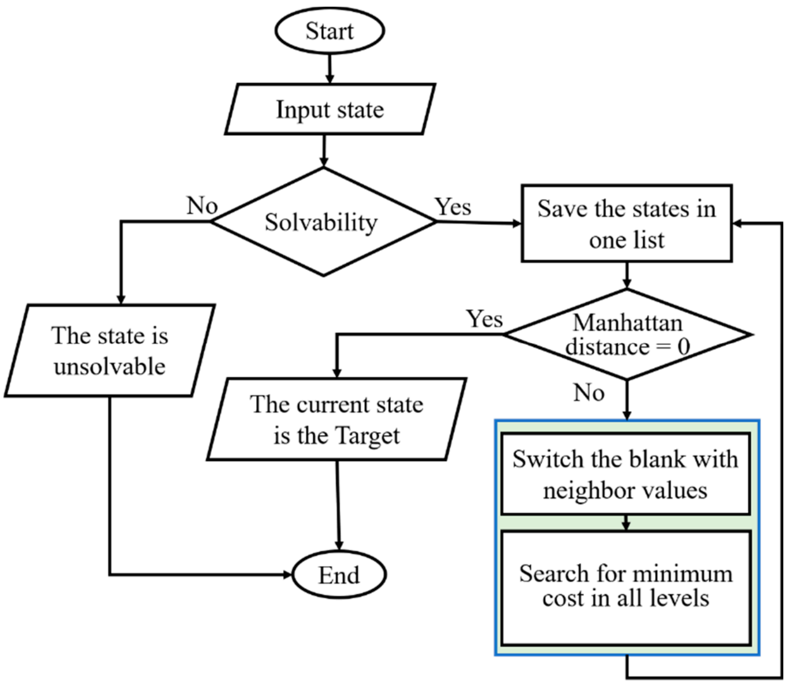

Start
Input state
Solvability
No
Yes
Save the states in one list
The state is unsolvable
Yes
Manhattan distance = 0
No
The current state is the Target
Switch the blank with neighbor values
Search for minimum cost in all levels
End

Figure 5. A-star algorithm for the n-puzzle.

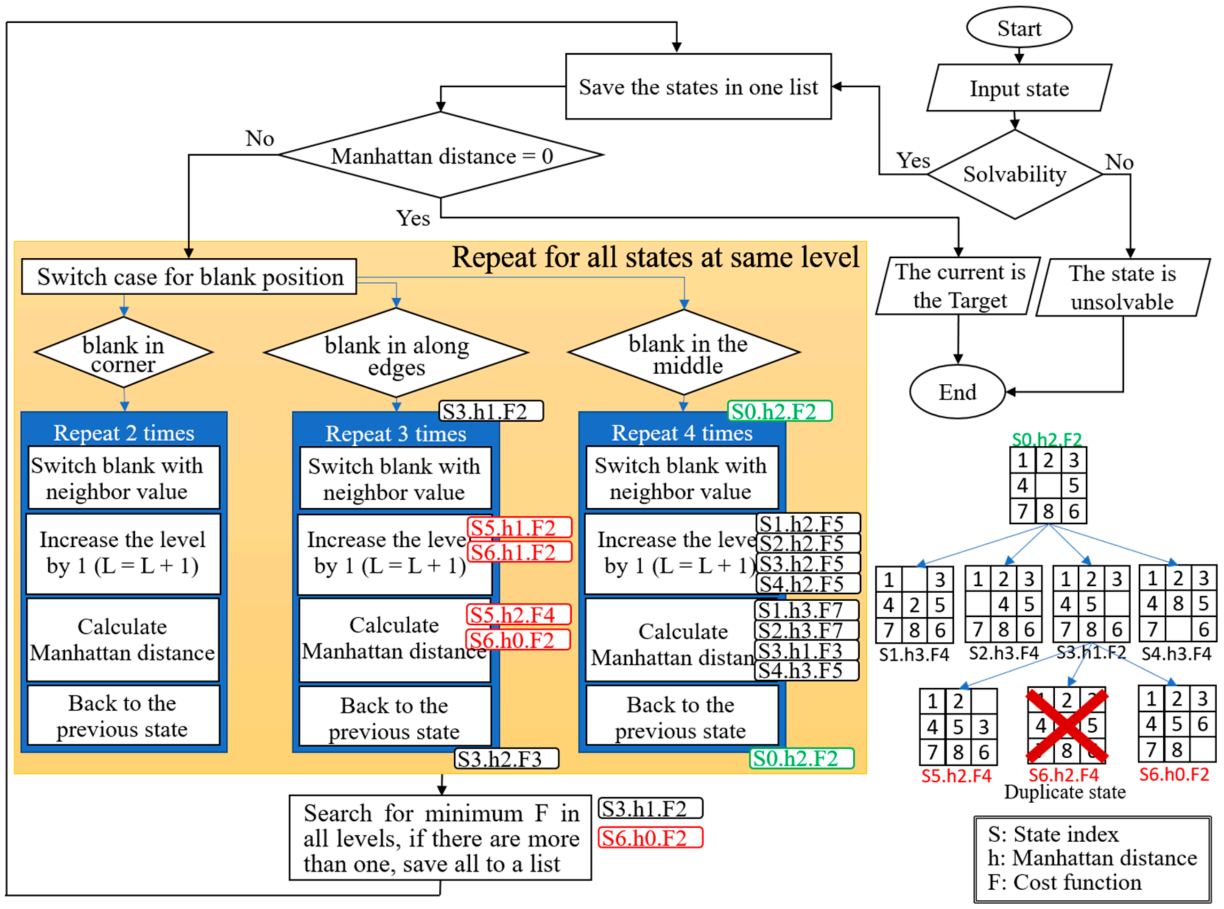

Start
Input state
Solvability
Yes
No
Save the states in one list
Manhattan distance = 0
No
Yes
The current is the Target
The state is unsolvable
End
Repeat for all states at same level
Switch case for blank position
blank in corner
blank in along edges
blank in the middle
Repeat 2 times
Switch blank with neighbor value
Increase the level by 1 (L = L + 1)
Calculate Manhattan distance
Back to the previous state
Repeat 3 times
Switch blank with neighbor value
Increase the level by 1 (L = L + 1)
Calculate Manhattan distance
Back to the previous state
S3.h1.F2
S5.h1.F2
S6.h1.F2
S5.h2.F4
S6.h0.F2
S3.h2.F3
Repeat 4 times
Switch blank with neighbor value
Increase the level by 1 (L = L + 1)
Calculate Manhattan distance
Back to the previous state
S0.h2.F2
S1.h2.F5
S2.h2.F5
S3.h2.F5
S4.h2.F5
S1.h3.F7
S2.h3.F7
S3.h1.F3
S4.h3.F5
S0.h2.F2
Search for minimum F in all levels, if there are more than one, save all to a list
S3.h1.F2
S6.h0.F2
S0.h2.F2
S1.h3.F4
S2.h3.F4
S3.h1.F2
S4.h3.F4
S5.h2.F4
S6.h2.F4
S6.h0.F2
Duplicate state
S: State index
h: Manhattan distance
F: Cost function

Figure 6. The implementation of the A-star algorithm for the 8-puzzle.

The conditional sentences in Algorithm 1 describe the implementation of the A* algorithm.

Algorithm 1: A* Implementation for 8-puzzle

```
1:  if solvable then
2:      Check Manhattan distance
3:  else
4:   End Algorithm
5:  Repeat until finding the target
6:      if Manhattan $\neq 0$ then
7:          Find a blank
8:          Perform Procedure switching blank
9:          Search for minimum cost
10:     else
11:         Input is the target
12:     end if
13: end repeat
```

The procedure of switching the blank with neighbors in order to generate branch nodes is described as Algorithm 2:

Algorithm 2: Procedure: Switching blank

```
1: if blank in a corner then
2:     Repeat 2 times: switch blank
3: else
4: if blank in along edges then
5:     Repeat 3 times: switch blank
6: else
7: if blank in the middle then
8:     Repeat 4 times: switch blank
9: end if
```

Switching blank involves three steps:

- Switch blank with a neighbor;
- Increase the depth (level in the tree which denotes the solution steps) by 1;
- Recalculate Manhattan distance.

2.4. Solvability Condition

The solvability can be checked by the inversion, which indicates that a pair of tiles in the current state is in reverse order of their places in the goal state. When the number of inversions is even, the puzzle is solvable; otherwise, it is unsolvable [34]. For example, if we have an 8-puzzle with the following configuration state (2, 1, 5; 4, blank, 3; 8, 6, 7), regardless of the blank, the inversion is calculated as follows:

The Investigated Tile	Tiles Follow the Investigated Tile	Number of Inversions
2	1	1
1	-	0
5	4 and 3	2
4	3	1
3	-	0
8	6 and 7	2
6	-	0
7	-	0
	Total inversions	6

The total inversions are six, which is an even number. Thus, the example configuration is solvable.

Proposal for the Solvability Problem

As mentioned before, the 8-puzzle has 9! different configurations, and only half of them are solvable. Since the state configurations in practical implementation in the warehouse are random, we will not be able to carry out sorting for unsolvable states (9!/2 states in the case of the 8-puzzle). Therefore, we need a scenario in which all states of the puzzle are solvable. In order to build such a scenario, we provided a pre-sorting strategy.

The products moving to the sorting area enter in a random configuration, which might be an unsolvable configuration. Therefore, we have to pre-sort the products on the sequencing board so that the pre-sorted configuration is a solvable one. The pre-sorting process is as follows:

1. Check the solvability by calculating the inversion number;
2. In case of an odd number of inversions, move the first six tiles to their specific positions on the sequencing board;
3. Switch the last two tiles on the board.

Figure 7 shows a flowchart of the pre-sorting process, and Figure 8 shows an example of the pre-sorting process for an unsolvable input configuration.

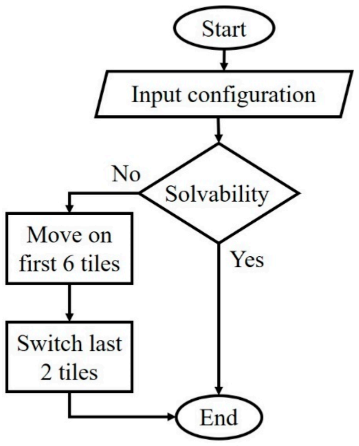


Figure 7. Flowchart of the pre-sorting process.

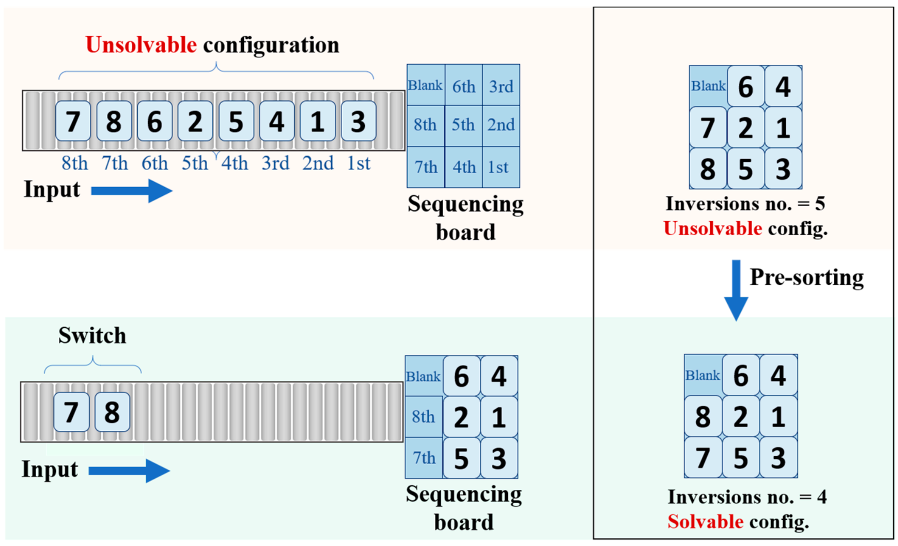


Figure 8. Pre-sorting process.

2.5. Board Shape

Different shaped boards can carry out the sequencing task. Therefore, four different sizes with two shapes were discussed with the same number of tiles. A 2 × 3 puzzle has 6! = 720 states, and half of them are unsolvable. By keeping the blanks in the corner of the puzzle to satisfy the reality of practical implementation in the warehouse, we reduced this to only 60 solvable states. For the same input and output configurations, all 60 states were examined, as shown in Figure 9.

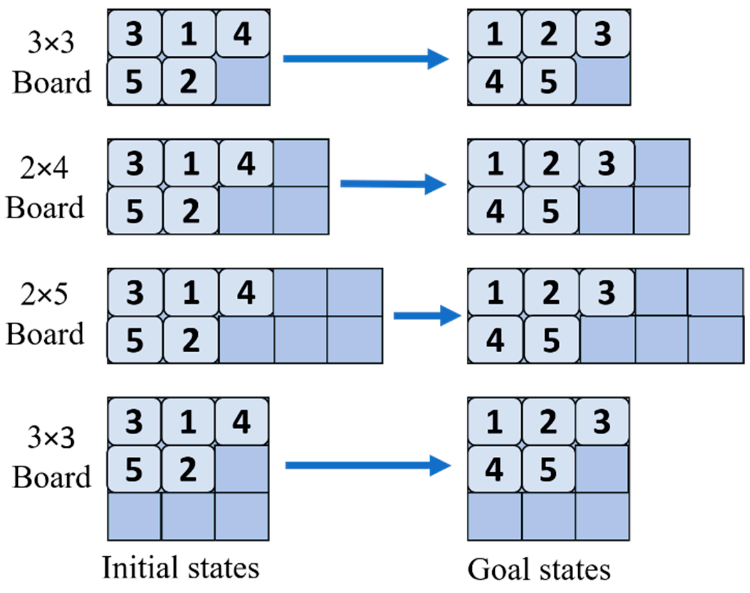


Figure 9. An example of the examination of the same state configurations with different board sizes and shapes.

Figure 10 illustrates the effect of different board shapes and sizes of the puzzle on the solution steps. The results of Figure 10 are summarized in Table 3.

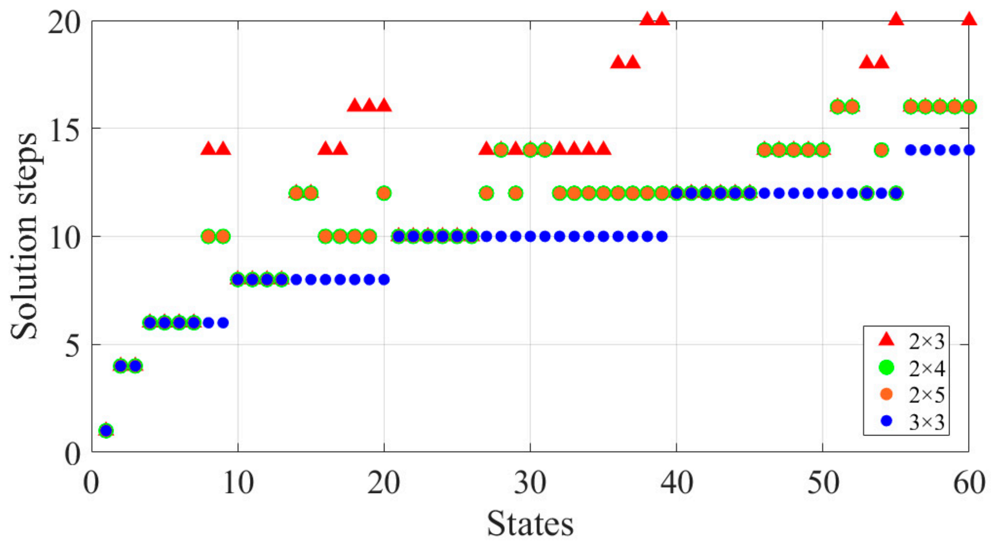


Figure 10. Comparison between different board sizes and shapes for the same number of boxes.

Table 3. Comparison of the performances of a 3 × 3 puzzle with different board sizes and shapes regarding the solution steps.

3 × 3	Better [%]	Same [%]	Worse [%]
vs. 2 × 3	61.6	38.4	0
vs. 2 × 4	58.3	41.7	0
vs. 2 × 5	58.3	41.7	0

From the table, the 3 × 3 board showed a better performance than the 2 × 3, 2 × 4, and 2 × 5 boards by 61.6%, 58.3%, and 58.3%, respectively. One of the reasons for these results is the difference in the number of blanks in the different shapes and sizes of the puzzle. In a scenario where the same size puzzle has the same number of blanks, many factors affect the number of steps for different shapes.

2.5.1. Branching Factor

The branching factor is the number of states that can be generated from each state in the tree. Usually, the branching factor measures the space complexity of the searching algorithm. The higher the branching factor, the lower the overhead of the repeatedly expanded states [31]. In our case, the analyzed data were generated from the target state, where we used the opposite concept of the branching factor. If the branching factor is higher, more states would be generated for a specific level in the tree (the level denotes the solution steps). Figure 11 illustrates an example of the effect of the branching factor on the number of generated states at the same level in the tree.

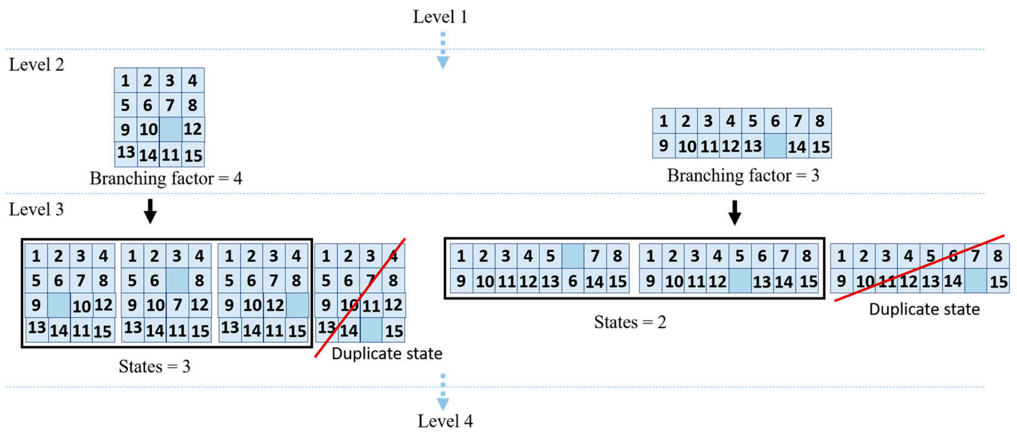


Figure 11. The effect of the branching factor on the number of generated states in the same level.

In Figure 11, two different shapes are illustrated, and we note that in level 3 (three steps to the solution), the square shape had more generated states than the rectangular one due to the difference in the branching factor. Figure 12 shows the average branching factor for both shapes discussed in the previous example.

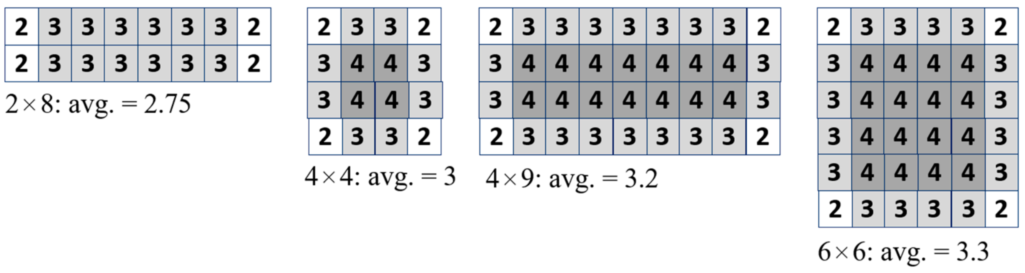


Figure 12. The average branching factor for the different sizes and shapes of boards.

2.5.2. Maximum Rectilinear Distance of One Tile

We suggest Equation (3) for calculating the maximum steps of a tile:

$$r_d = (\mathrm{L} + \mathrm{W}) - 2, \tag{3}$$

where r_d is the maximum rectilinear distance of the tile; L is the length of the board; and W is the width of the board.

A smaller distance for one tile results in a better board, since it decreases the number of initial steps of the pre-sorting process. Figure 13 illustrates the maximum distance that the tile can move.

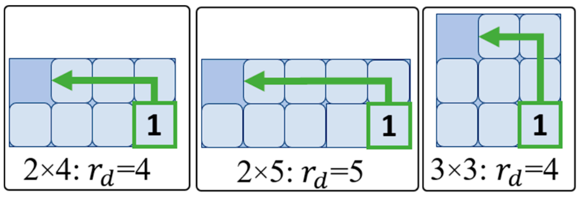


Figure 13. Maximum rectilinear distance of one tile of different board shapes and sizes.

From Figure 13, we noted that different board shapes could have the same r_d. With this in mind, we compared the performance depending on the maximum board capacity, as illustrated in Table 4.

Table 4. Comparison of different board shapes and sizes of puzzles, and the max. capacity in the case of the same r_d.

Max. Rectilinear Distance of One Tile	Max. Capacity	Board Size
4	7	2 × 4
	8	3 × 3
5	9	2 × 5
	11	3 × 4
6	14	3 × 5
	15	4 × 4
7	17	3 × 6
	19	4 × 5
8	20	3 × 7
	23	4 × 6
	24	5 × 5
9	23	3 × 8
10	26	3 × 9

From Table 4, we concluded that in the case of r_d, being the same for different board sizes and shapes, square puzzles provide more capacity than rectangular ones.

3. Results and Discussion

In this section, we numerically show how the shape of the puzzle affects the solution steps. Then, we investigate some factors affecting the pre-sorting steps, which will affect the overall steps of the operation.

In order to generalize the comparison of different shapes of the puzzle, the same size and number of blanks were used. First, we investigated the 16-boxes size of the puzzle. This size can sort 15 boxes with two different shapes (4 × 4 and 2 × 8). As Figure 14 shows, we generated 2×10^5 non-random states for both shapes, starting from the target state. Figure 14 illustrates the performance of the generated states regarding the solution steps.

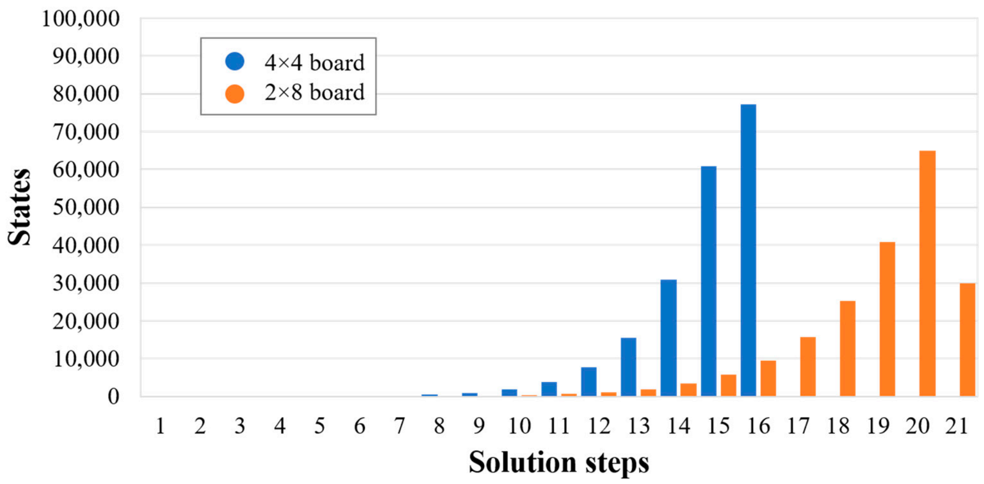


Figure 14. Comparison between the 4 × 4 and 2 × 8 board sizes with non-random states regarding the solution steps.

To verify the validity of our method to generate the state tree, we compared our 8-puzzle tree with other works regarding the following factors: maximum number of states, maximum solution steps, and average solution steps. Table 5 illustrates the comparison between our results for the 8-puzzle with others.

Table 5. Our generated tree for the 8-puzzle vs. other works.

Comparison Factor	Our Generated Tree	Other Works [25,28,35]
Maximum number of states	181,440	181,440
Maximum solution steps	31	31
Average solution steps	21.97	≈22

According to Table 5, we were able to validate our method, and the same program was used to generate the 2×10^5 states for different shapes in this section.

In the generated tree of 16-boxes size of the puzzle, we noticed a clearly significant difference in the state numbers of the two puzzles in the same tree depth (solution steps). In other words, states in one shape of the puzzle need more solution steps than the second shape. The equation that describes the number of states that need more solution steps is as follows:

$$N = \sum_{i=S_{max.1}+1}^{S_{max.2}} N_i, \tag{4}$$

where N is the number of states that need more solution steps; $S_{max.1}$ is the maximum solution steps of the first shape; $S_{max.2}$ is the maximum solution steps of the second shape; and N_i is the number of states in depth i.

According to Equation (4), for all generated states, 88.35% of states could provide fewer solution steps in the 4 × 4 board than in the 2 × 8 for the 2×10^5 states. Furthermore, Equation (5) provides an increasing percentage of solution steps for different shapes.

$$S_{plus} = \frac{|S_{max.1} - S_{max.2}|}{S_{max}} \times 100\%, \tag{5}$$

where S_{plus} is the increasing percentage of solution steps; $S_{max.1}, S_{max.2}$ are the same as in Equation (4); and S_{max} is the total solution steps. Based on Equation (5), the results prove that the 4 × 4 board achieved 23.8% of steps better than the 2 × 8 board at 2×10^5 states. Overall, when increasing the number of states in both given boards, the 4 × 4 board performed better than the 2 × 8 board in terms of the number of steps.

Next, we considered a 36-boxes size of the puzzle, which can sort 35 boxes. In this case, there are four different shapes (6 × 6, 4 × 9, 3 × 12, and 2 × 18). The same analysis as in the previous case with the size of 16-boxes was carried out. Figure 15 shows the solution steps of all states with the same number of boxes for different shapes.

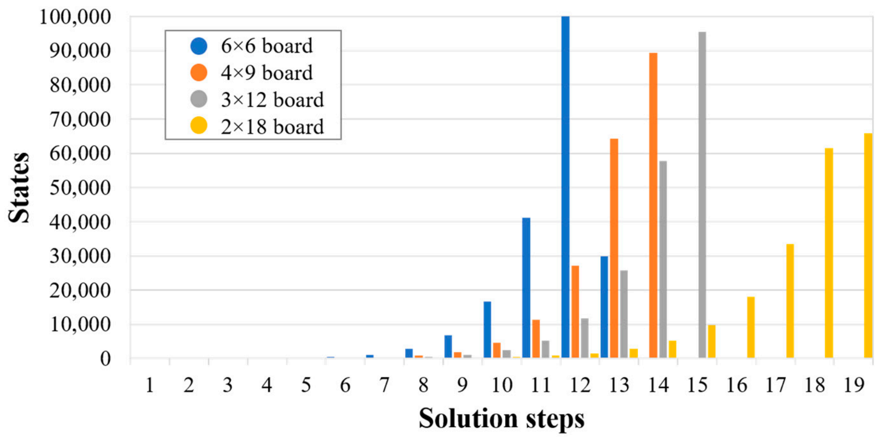


Figure 15. Comparison between the 6 × 6, 4 × 9, 3 × 12, and 2 × 18 board sizes for non-random states.

We observed the same trend in Figure 14 for our 16-boxes size of the puzzle in Figure 15. For all generated states, and referring to Equation (4), we confirmed that 44.63%, 76.60%, and 96.92% of states provided fewer solution steps in the 6 × 6 board than in the 4 × 9, 3 × 12, and 2 × 18 boards, respectively. Moreover, from Equation (5), the 6 × 6 board provided 7.14%, 13.33%, and 31.57% steps fewer than the 4 × 9, 3 × 12, and 2 × 18 boards, respectively. From these results, we deduced that the square shape of the puzzle had a better performance than the rectangular shape. However, more analyses are necessary to verify how the pre-sorting steps for different shapes will affect the overall solution steps.

The pre-sorting process plays a key role in the whole sorting system in practical implementations. Therefore, we considered several factors that can affect the pre-sorting steps, for instance, the rectilinear distance of one tile.

As mentioned in Section 2.5.2, the puzzle shape affects the rectilinear distance of one tile, r_d as well as the number of initial steps in pre-sorting. Figure 16 illustrates the initial steps to fill in the sequencing board with different sizes and shapes concerning r_d.

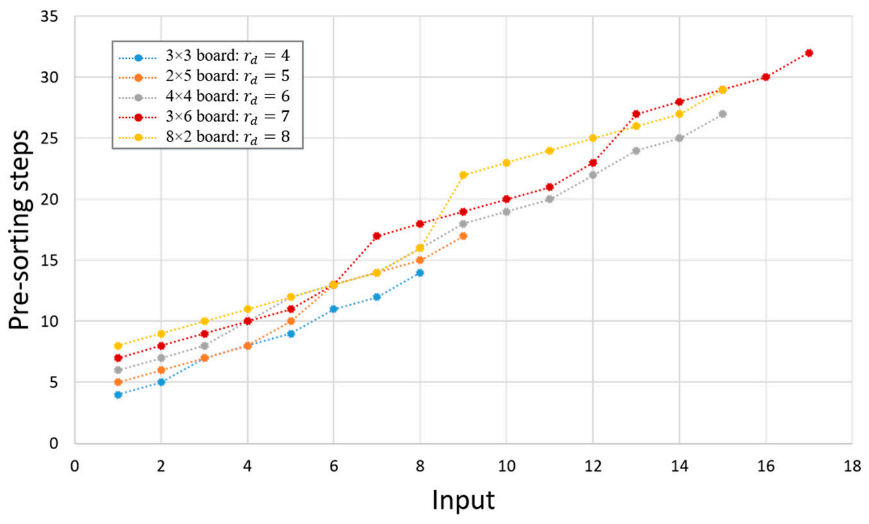


Figure 16. Effect of r_d on the pre-sorting steps.

As is clear from Figure 16, increasing the rectilinear distance of one tile will also increase the pre-sorting steps. However, a reasonable question arises when dealing with different shapes: how does the Aspect Ratio (AR) of the puzzle affect the performance in terms of solution steps? To answer this question, we investigated the relationship between the aspect ratio and rectilinear distance.

The Aspect Ratio is the number of columns divided by the number of rows of the puzzle, and this has a direct effect on the rectilinear distance of one tile, r_d, and further on the pre-sorting steps. Table 6 illustrates the corresponding r_d of the aspect ratio for the different puzzle shapes and sizes outlined previously.

Table 6. Aspect Ratio and rectilinear distance of one tile for different puzzle sizes.

Puzzle Size	Aspect Ratio	Rectilinear Distance
4×4	1	6
2×8	4	8
6×6	1	10
4×9	2.25	11
3×12	4	13
2×18	9	18

According to Table 6, we confirmed the direct relationship between the aspect ratio and rectilinear distance. Thus, a smaller AR reduces the r_d, which also reduces the pre-sorting steps.

From all results illustrated in this study, we concluded that the square puzzle was the most efficient shape in our sequencing system. However, the research findings have a number of possible limitations, namely the puzzle capacity. The maximum capacity of the puzzle is restricted by the size of the board. In fact, a high number of tiles would lead to an unreasonable search time and number of steps. Multiple solutions could solve this limitation such as (a) using multi-puzzles of the same size for one line, and (b) using a buffer-line along with input-line to temporarily store the extra items. These scenarios are still under investigation.

4. Conclusion

Item sequencing has become necessary in order to increase the efficiency of logistics operations. In this study, we focused on the material handling devices that could carry out the sequencing task. We developed a puzzle-based sequencing system with highly efficient floor space utilization. Different searching techniques were discussed, and the A-star algorithm was chosen to find the shortest solution for the puzzle. Furthermore, a pre-sorting process was proposed to overcome unsolvable configurations. In the pre-sorting process, we switched the last two items; therefore, different filling-in processes might affect the overall steps to reach the final goal of the puzzle.

Two shapes of puzzle with the same size were considered to achieve the minimum number of solution steps. The results clarified a different number of states in the same level of the generated tree for both shapes with different sizes. For different puzzles, if we give a random state, there is a high probability that it will be in the tree with the higher number of states in the same level. Based on the results of the numerical calculations, it can be concluded that a square shape can provide a shorter solution than a rectangular shape. The findings suggest that a puzzle-based sequencing system would be preferred for highly efficient floor space utilization.

Author Contributions: Conceptualization, K.I.; methodology, R.A.; software, R.A.; supervision, K.I.; writing–original draft, R.A.; writing–review and editing, R.A.; All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Beyond Co., Ltd.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to acknowledge the valuable comments and suggestions of the reviewers, which have improved the quality of this paper. This work was supported partly by the collaborative research project between the Center for Socio-Robotic Synthesis, Kyushu Institute of Technology and Beyond Co., Ltd.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rushton, A.; Croucher, P.; Baker, P. *The Handbook of Logistics & Distribution Management*; The Chartered Institute of Logistics and Transport: Corby, UK, 2010; p. 636.
2. Fumi, A.; Scarabotti, L.; Schiraldi, M.M. Minimizing warehouse space with a dedicated storage policy. *Int. J. Eng. Bus. Manag.* **2013**, *5*. [CrossRef]
3. Den Berg, J.P.V.; Zijm, W.H.M. Models for warehouse management: Classification and examples. *Int. J. Prod. Econ.* **1999**, *59*, 519–528. [CrossRef]
4. Daraei, M. Warehouse Redesign Process: A Case Study at Enics Sweden AB. Master's Thesis, Mälardalen University, Högskoleplan, Västerås, Sweden, 2014; pp. 1–76.
5. Hsieh, L.F.; Tsai, L. The optimum design of a warehouse system on order picking efficiency. *Int. J. Adv. Manuf. Technol.* **2006**, *28*, 626–637. [CrossRef]
6. Gue, K.R. Very high density storage systems. *IIE Trans.* **2006**, *38*, 79–90. [CrossRef]
7. Gue, K.R.; Kim, B.S. Puzzle-Based Storage Systems. *Nav. Res. Logist.* **2007**, *54*, 556–567. [CrossRef]
8. Shah, B.; Khanzode, V. A comprehensive review and proposed framework to design lean storage and handling systems. *Int. J. Adv. Oper. Manag.* **2015**, *7*, 274–299. [CrossRef]
9. Aleisa, E.E.; Lin, L. For effective facilities planning: Layout optimization then simulation, or vice versa? *Proc.-Winter Simul. Conf.* **2005**, *2005*, 1381–1385. [CrossRef]
10. Inman, R.R. ASRS sizing for recreating automotive assembly sequences. *Int. J. Prod. Res.* **2003**, *41*, 847–863. [CrossRef]
11. Gue, K.R.; Uluda, O.; Furmans, K. A High-Density System for Carton Sequencing. *6th Int. Sci. Symp. Logist.* **2012**. Available online: https://kevingue.wordpress.com/publications/ (accessed on 16 October 2021).
12. Boysen, N.; Stephan, K.; Weidinger, F. Manual order consolidation with put walls: The batched order bin sequencing problem. *EURO J. Transp. Logist.* **2019**, *8*, 169–193. [CrossRef]
13. Boysen, N.; Fedtke, S.; Weidinger, F. Optimizing automated sorting in warehouses: The minimum order spread sequencing problem. *Eur. J. Oper. Res.* **2018**, *270*, 386–400. [CrossRef]
14. Rethmann, J.; Wanke, E. Storage controlled pile-up systems, theoretical foundations. *Eur. J. Oper. Res.* **1997**, *103*, 515–530. [CrossRef]
15. Yalcin, A.; Koberstein, A.; Schocke, K.O. Motion and layout planning in a grid-based early baggage storage system: Heuristic algorithms and a simulation study. *OR Spectr.* **2019**, *41*, 683–725. [CrossRef]
16. Kota, V.R.; Taylor, D.; Gue, K.R. Retrieval time performance in puzzle-based storage systems. *J. Manuf. Technol. Manag.* **2015**, *26*, 582–602. [CrossRef]
17. Gue, K.R.; Furmans, K.; Seibold, Z.; Uludag, O. GridStore: A puzzle-based storage system with decentralized control. *IEEE Trans. Autom. Sci. Eng.* **2014**, *11*, 429–438. [CrossRef]
18. Uludag, O. GridPick: A High Density Puzzle Based Order Picking System with Decentralized Control. Ph.D. Thesis, Auburn University, Auburn, AL, USA, 2014.
19. Gue, K.; Hao, G. A High-Density, Puzzle-Based System for Rail-Rail Container Transfers. In Proceedings of the 14th IMHRC Proceedings, Karlsruhe, Germany, 2016; Available online: https://digitalcommons.georgiasouthern.edu/pmhr_2016/14 (accessed on 16 October 2021).
20. Hao, G. GridHub: A Grid-Based, High-Density Material Handling System. Ph.D. Thesis, University of Louisville, Louisville, KY, USA, 2020.
21. Shekari Ashgzari, M.; Gue, K.R. A puzzle-based material handling system for order picking. *Int. Trans. Oper. Res.* **2021**, *28*, 1821–1846. [CrossRef]
22. Yalcin, A.; Koberstein, A.; Schocke, K.O. An optimal and a heuristic algorithm for the single-item retrieval problem in puzzle-based storage systems with multiple escorts. *Int. J. Prod. Res.* **2019**, *57*, 143–165. [CrossRef]
23. Shirazi, E.; Zolghadr, M. An item retrieval algorithm in flexible high-density puzzle storage systems. *Appl. Syst. Innov.* **2021**, *4*. [CrossRef]
24. Tetouani, S.; Chouar, A.; Lmariouh, J.; Soulhi, A.; Elalami, J. A "Push-Pull" rearrangement while routing for a driverless delivery vehicle. *Cogent Eng.* **2019**, *6*, 1567662. [CrossRef]
25. Iordan, A.-E. A Comparative Study of Three Heuristic Functions Used to Solve the 8-Puzzle. *Br. J. Math. Comput. Sci.* **2016**, *16*, 1–18. [CrossRef]
26. Shaban, R.; Natheer Alkallak, I.; Mohamad Sulaiman, M. Genetic Algorithm to Solve Sliding Tile 8-Puzzle Problem. *J. Educ. Sci.* **2010**, *23*, 145–157. [CrossRef]

27. Piltaver, R.; Luštrek, M.; Gams, M. The pathology of heuristic search in the 8-puzzle. *J. Exp. Theor. Artif. Intell.* **2012**, *24*, 65–94. [CrossRef]
28. Reinefeld, A. Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA*. *Int. Jt. Conf. Artif. Intell.* **1993**, *1*, 248–253.
29. Mishra, A.K.; Siddalingaswamy, P.C. Analysis of tree based search techniques for solving 8-puzzle problem. In Proceedings of the 2017 Innovations in Power and Advanced Computing Technologies, Vellore, India, 21–22 April 2017; pp. 1–5.
30. Pathak, M.J.; Patel, R.L.; Rami, S.P. Comparative Analysis of Search Algorithms. *Int. J. Comput. Appl.* **2018**, *179*, 40–43. [CrossRef]
31. Nilsson, N.J. *Artificial Intelligence: A Modern Approach*; Prentice Hall: Hoboken, NJ, USA, 1996; Volume 82, ISBN 0137903952.
32. Nosrati, M.; Karimi, R.; Hasanvand, H.A. Investigation of the * (Star) Search Algorithms: Characteristics, Methods and Approaches. *World Appl. Program.* **2012**, *2*, 251–256.
33. Zhou, Y.; Cheng, X.; Lou, X.; Fang, Z.; Ren, J. Intelligent Travel Planning System based on A-star Algorithm. In Proceedings of the 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference, Chongqing, China, 12–14 June 2020; pp. 426–430. [CrossRef]
34. Ando, R.; Takefuji, Y. A new perspective of paramodulation complexity by solving massive 8 puzzles. *arXiv* **2020**, arXiv:2012.08231.
35. Korf, R.E.; Reid, M.; Edelkamp, S. Time complexity of iterative-deepening-A*. *Artif. Intell.* **2001**, *129*, 199–218. [CrossRef]