applied system innovation

MDPI

Article

An Item Retrieval Algorithm in Flexible High-Density Puzzle Storage Systems

Ehsan Shirazi [1] and Mohammad Zolghadr [2,*]

[1] Department of Industrial and Management Systems Engineering, West Virginia University, Morgantown, WV 26506, USA; ehshirazi@mix.wvu.edu
[2] School of Business Administration, University of California Riverside, Riverside, CA 92521, USA
* Correspondence: mzolg001@ucr.edu

Abstract: This paper studies a design of a puzzle-based storage system. We developed an item retrieval algorithm for our system which has three advantages over the previous counterparts in the literature: (i) we can retrieve items from all sides of our storage system; (ii) the existence of only one empty cell in our system is sufficient to retrieve an item; and (iii) our algorithm never ends in deadlocks. The main feature of our algorithm is to prefer three moves to five moves in the process of moving the seized empty cell toward the optimal side of the requested item. The conventional view in the literature assumes that increasing the number of empty cells always reduces the number of movements required for retrieving items; however, our simulation results show that depending on the size of the puzzle and the number of the requested items, increasing empty cells might make the retrieval process more complicated.

Keywords: puzzle-based storage systems; warehousing system design; novel algorithms; material handling; agent-based modeling; simulation

Citation: Shirazi, E.; Zolghadr, M. An Item Retrieval Algorithm in Flexible High-Density Puzzle Storage Systems. *Appl. Syst. Innov.* **2021**, *4*, 38. https://doi.org/10.3390/asi4020038

Academic Editor: Friedhelm Schwenker

Received: 16 March 2021
Accepted: 1 June 2021
Published: 11 June 2021

1. Introduction

Warehousing design is one of the main competitive areas in the retail industry. All giant retail corporations endeavor to improve their warehousing systems to shorten the delivery time of their goods and services. For instance, Amazon has recently started its Prime Now plan to deliver daily essentials and groceries in less than two hours in a few metropolitan areas; indeed, the backbone of this plan is Amazon's innovative warehousing and material handling system [1]. Therefore, implementing novel designs and algorithms plays a key role in enhancing the performance of warehousing systems. In this paper, we study modern autonomous agent-based conveyor systems to support high-density, high-throughput storage/retrieval. More specifically, we designed a new puzzle-based algorithm for these systems and implemented it into an object-oriented software program.

We mainly focus on high-density storage systems (HDSS). High-density storage systems are mainly used in industries with high demand and temporary storage [2]. These systems benefit from high flexibility while minimizing the number of replenishment/retrieval movements of the items. Compared to the traditional conveyor systems, HDSS are decentralized and highly adaptable to exogenous shocks such as demand fluctuations [2]. Due to the presence of independent agents for building up a decentralized network in HDSS, different storage layouts can be built to best conform to requirements. In HDSS, we can store items on the top of other ones or deeply on the shelves; therefore, the main research question arising here is to find the most efficient way to replenish and retrieve items.

Gue [3] was the first person who applied decentralized puzzles to HDSS. The puzzle network can manipulate the location of the empty spots to move the requested item toward the exit point. Gue's puzzle network includes one input location, one output location and one single or multiple empty spots to replenish and retrieve items. Gue et al. [4] developed

this initial puzzle by considering replenishment from the top, storing in the middle and retrieving from the bottom.

The main contribution of our puzzle-based algorithm to the literature is considering the possibility of replenishment from all sides of the puzzle. In addition, since our algorithm needs only one empty cell to replenish the requested item, the storage system benefits from a higher density. Finally, we also included a "deadlock prevention" algorithm such that our puzzle network never ends in deadlocks.

2. Related Literature

Puzzle networks have been developed using two kids' single player games, Rush Hour, and the 15-puzzle game. According to Hearn and Demaine [5], puzzle networks can be considered as PSPACE-Hard problems. In fact, the 15-Puzzle game was the main inspiration for Gue and Kim [6] to build their high-density storage system based on using one empty spot to move other items. While they focused on the special case of one replenishment spot, Gue and Furmans [7] developed their system by considering the possibility of replenishment from both the top and the bottom of the puzzle. Rohit et al. [8] evolved the previous puzzle networks by randomly placing empty cells in the puzzle.

Regarding the puzzle configurations, Gue et al. [9] explained how the number of empty cells affects the performance of the system. They showed that a puzzle with fewer columns has a better performance, and that increasing the ratio of the rows to the columns above ten can significantly improve the performance of the system.

To solve the deadlock issue, Furmans et al. [10] used a puzzle-based network to prevent deadlocks of one or more grids' failure. Gue et al. [4] further introduced GridPick as a deadlock-free system, and Uludag [11] developed the GridPick concept to build a system with two retrieval sides. In addition, Krühn et al. [12] built a conveyor matrix, using a small scale and multi-directional modules, to introduce their potential deadlock-free algorithm.

All of the above studies emphasized the retrieval of one single load, whereas Mirzaei et al. [13] investigated the minimum retrieval time of multiple loads from each grid. They observed that double-loaded puzzles can reduce the retrieval time by 17%.

For the latest progress in the literature of puzzle-based storage systems, we refer the reader to two recent studies: Hao [14] developed an automatic puzzle-based storage system under decentralized control called the GridHub system. This system results in a high throughput with parallel order processing. Shekari Ashgzari and Gue [15] introduced an algorithm which accomplishes a "puzzle-like" movement of items on the grid. The authors also studied the effects of several design parameters on the system performance.

For a deeper overview of the recent literature on automated warehousing and puzzle-based storage systems, we refer the reader to Azadeh et al. [16], Boysen et al. [17], Kota et al. [18], Manzini et al. [19], Seibold [20], Yalcin et al. [21] and Zaerpour et al. [22].

3. Methodology

3.1. Network Elements

Our puzzle is a rectangle which consists of several rectangular cells with the same dimensions. In puzzles, we represent each agent by one cell. Each cell plays the role of an independent machine which can store an item or can be empty. The cell decides whether to keep the stored tote or pass it either forward, backward, to the left or to the right. Each network consists of some empty cells referred to as white cells or escorts. White cells direct the requested item and move along with it to the exit point (Figure 1). The number of white cells in each network affects the efficiency of the whole network. Stored items—which we represent in gray color—can be considered as goods, totes (items), pallets or unit loads; the network can call each of these items for retrieval. We represent the requested item in black color.

Figure 1. One movement in the puzzle using a white cell.

3.2. Retrieval Scenarios

In this study, we considered four different scenarios to retrieve an item. These scenarios are different in the likelihood of reaching deadlocks and implementation difficulty.

3.2.1. Assigning White Cells to Specific Areas of the Puzzle

This scenario first divides the puzzle into subareas and then assigns one or more white cells to each subarea (Figure 2). We can move each white cell only in its own subarea. Obviously, this restriction on the movements of white cells is detrimental to the performance of the puzzle. Moreover, if we want to retrieve a black cell outside its current subarea, we require two white cells: one from the item's subarea and one from the retrieval subarea; this overuse of white cells is detrimental to the efficiency of the network.

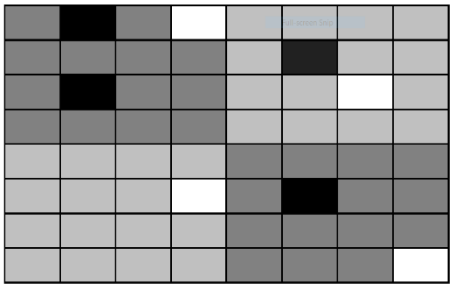

Figure 2. A puzzle including white cells assigned to different subareas in the board.

3.2.2. At Least One Empty Cell in Each Column and Row

In this scenario—introduced by Gue et al. [4]—empty cells open a temporary aisle in the board. The main advantage of this scenario is that it never ends in deadlocks; however, since this scenario requires a large number of white cells, its implementation negatively affects the performance of the network.

3.2.3. Assigning a White Cell Specifically to Each Requested Item

A white cell only serves the requested item through its entire retrieval operations. Therefore, we can assign the same white cell to a different item. Indeed, we can implement this scenario on a first come, first served (FCFS) basis. Despite the simplicity of this scenario, it leads to a higher number of movements. For instance, in some cases, it might be more efficient to switch from one called item to another one before the complete retrieval of the former; however, this scenario prevents from switching the called item we serve before complete retrieval of the previous item.

3.2.4. Assigning a White Cell to a Called Item Based on the Nearest Distance

In this scenario, we can switch from one called item to another one before the complete retrieval of the former. In contrast to the third scenario, this scenario is more complicated to implement because at each step, we must solve a two-dimensional optimization problem: first, finding the nearest called cell/item, and then finding the shortest distance to that cell/item.

3.3. Puzzle Setup

We considered square puzzles to implement our algorithm. We assumed we can retrieve an item from any point on the boarder of the square puzzles. Puzzle-based networks are considered as PSPACE-Hard problems, and finding the optimal solution becomes extremely hard as the puzzle dimensions grow. Therefore, we considered simulation to find an estimation of the optimal solution.

We considered 6 different sizes for our simulated puzzles: 3×3, 6×6, 12×12, 20×20, 25×25 and 50×50. We assumed that three items are always requested from the board, i.e., we always have three black cells, but the number of empty cells—white cells—is a variable. Indeed, we considered only three black cells because of two reasons: (i) we can generate a sufficient number of deadlock situations for our analysis, and (ii) the algorithm obtains the solution in a reasonable time.

3.4. Algorithm

Our algorithm consists of two phases: search and movement. In the "search" phase, empty cells are assigned to the requested cells. If the number of empty cells is smaller than the number of requested cells, the requested cells compete to seize an empty cell. In the "movement" phase, we move the empty cells. We repeat this cycle until all requested items are retrieved.

We define the optimal side as the nearest side of a requested cell to the retrieval point. The selection of this optimal side is a critical step in our algorithm. Consider the puzzle shown in Figure 3. Since our algorithm is capable of retrieving the requested item from all sides of the puzzle, we arbitrarily assume that the network has called the black cell from the left side of the puzzle. Therefore, cell 1 is next to the optimal side of the called cell, and white cells must compete to seize cell 1. In this step, we select the white cell at the bottom of the puzzle due to its proximity to cell 1.

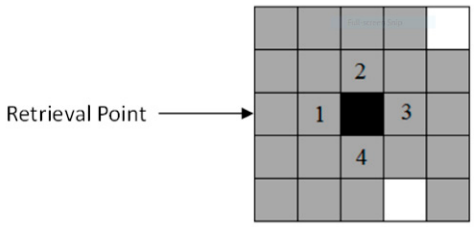

Figure 3. Importance of the optimal side of requested items.

A black cell must seize a white cell to move. To do so, first, the seized white cell must reach the optimal side of the black cell. Therefore, when we have more than one white cell in the puzzle, we must select the white cell which has the shortest distance to the optimal side of the black cell. More specifically, in our algorithm, the black cell sends a message to all of its four directions to seize a white cell. If there is no white cell next to the requested item, each cell which has received the message sends it to three different directions to find an empty spot. Figure 4 shows how our message-passing algorithm finds an empty cell. The white cell at the bottom is found to be the closest empty cell to the requested item in this figure.

The main feature of our algorithm is to prefer 3 moves to 5 moves in moving the seized white cell toward the optimal side. Figures 5 and 6 illustrate 3-move and 5-move scenarios, respectively. In Figure 5, since the black cell is called from the left side, it can be seized by the white cell in 3 moves. However, in Figure 6, the white cell is in the opposite direction of the optimal side, and therefore it seizes the black cell in 5 moves. Gue and Kim [6] showed that to minimize the number of movements in a puzzle, there must be as many 3 moves as possible until retrieval. However, in some cases—similar to Figure 6—the only way we can move the black cell toward the retrieval point is with 5 moves.

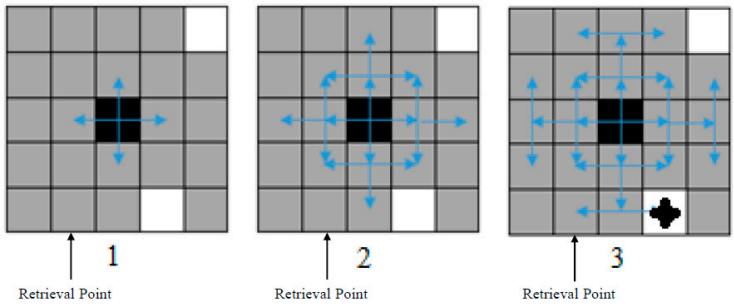

Figure 4. An example of message-passing algorithm.

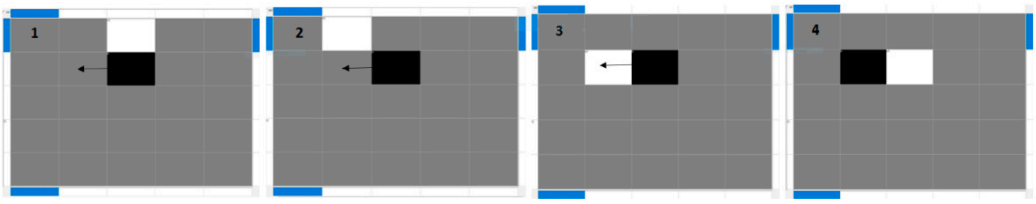

Figure 5. An example of a 3-move scenario.

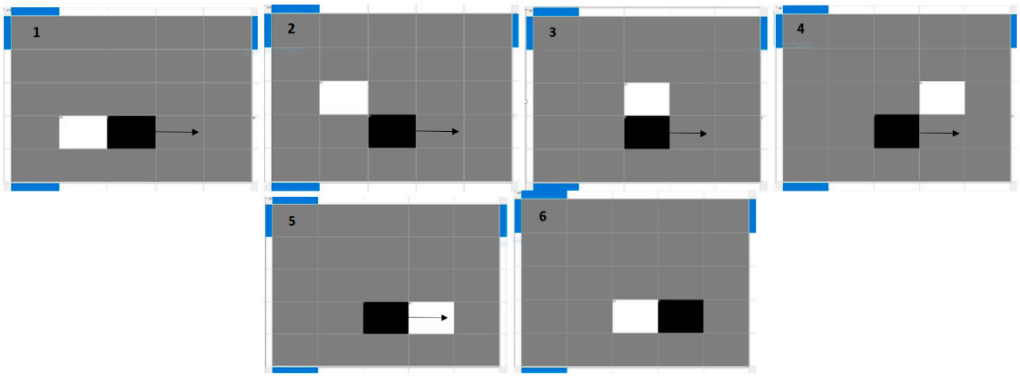

Figure 6. An example of a 5-move scenario.

3.4.1. Deadlock Prevention

A puzzle ends in a deadlock when a black cell seizes a white cell, but they cannot move anymore. There are three deadlock situations: (i) more than one black cell competes to seize one white cell at equal distance (Figure 7a); (ii) more than one white cell has the same distance to the optimal side of a black cell (Figure 7b); and (iii) colliding black cells must be retrieved from opposing sides while seizing different white cells (Figure 7c). All other potential deadlocks can be considered as a permutation of the above three situations.

One novelty aspect of our work is including a "deadlock prevention" algorithm into our main algorithm to break deadlocks. In situations 1 and 3, the algorithm temporarily considers both black cells as gray cells. Then, it randomly draws a number from {0; 1; 2; 3} for each requested item; this number indicates after how many movements the requested cell turns back to black. If the drawn numbers are the same for both requested cells, the algorithm again draws two new numbers until the puzzle exits the deadlock situation. In situation 2, our deadlock prevention algorithm randomly selects one of the white cells. As a result, our algorithm can exit deadlocks without any external interference.

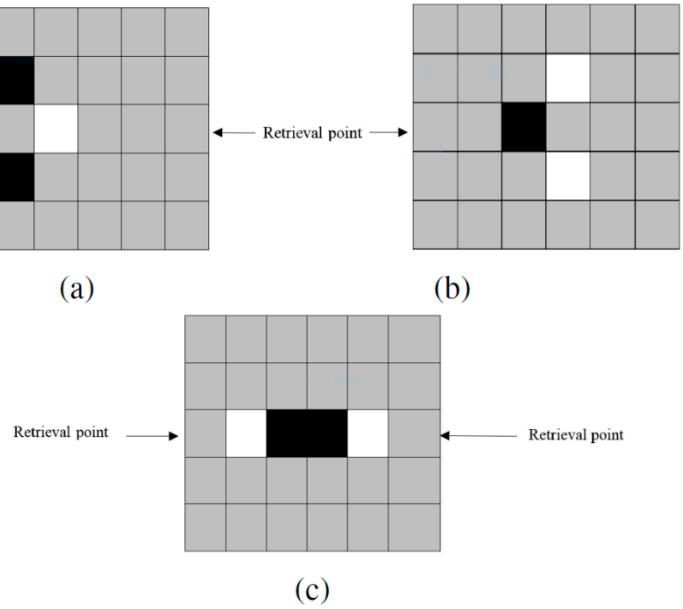

Figure 7. Deadlock prevention situations. (**a**) Deadlock Type 1; (**b**) Deadlock Type 2; (**c**) Deadlock Type 3.

3.4.2. Methodology Flowcharts

As Figure 8 shows, the algorithm first randomly selects one of the black cells and identifies its location and optimal retrieval side in the puzzle. Then, the nearest white cell is selected, and potential deadlocks are resolved. At this step, the search phase finishes, and the algorithm enters the moving phase. In the moving phase, either a white cell moves toward the black cell, or vice versa. Finally, when the black cell reaches the retrieval point, the item is retrieved, and the black cell becomes an empty cell. We present the flowchart for the deadlock prevention part of the algorithm in Figure 9.

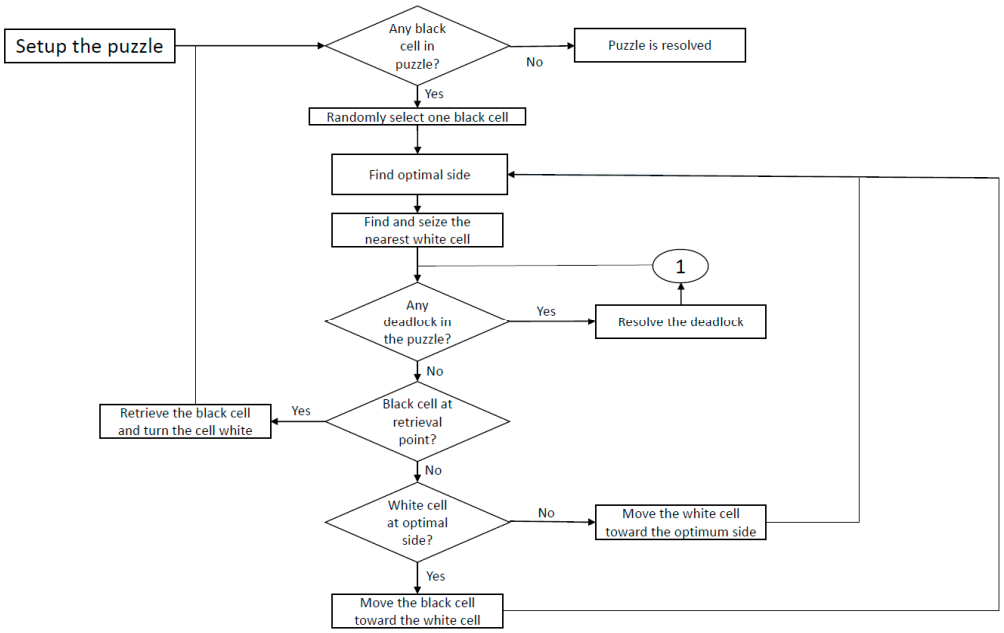

Figure 8. Main algorithm.

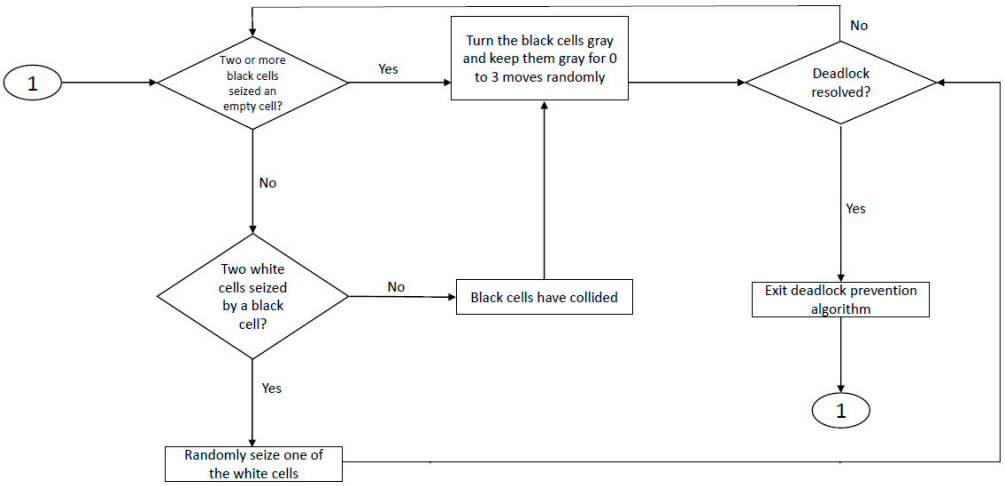

Figure 9. Overview of deadlock prevention algorithm.

3.4.3. The Novelty Aspects of the Present Algorithm

To summarize, the major new aspects of our algorithm are listed below:

1. We considered a deadlock prevention algorithm in our main algorithm. This prevents the algorithm to stick in deadlock situations. Therefore, our algorithm always results in a solution.
2. Our high-density puzzle algorithm is capable of retrieving items from all sides of the puzzle. This increases the flexibility of our storage system and improves the average retrieval movements (ARM) needed to retrieve the items.
3. We introduced a novel search algorithm which works based on sending a message to all four directions of the black cell to seize a white cell. If the algorithm cannot find a white cell next to the black cell, each cell which has received the message sends it to three different directions to find an empty spot. The main advantage that our search algorithm provides is decreasing ARM by finding the nearest empty spot to the requested item.

3.5. Examples

3.5.1. Example 1

Our puzzle in this example includes one black cell and two empty/white cells (Figure 10). We assume that the item should be retrieved from the top right-hand corner of the puzzle. Therefore, the optimal side of the black cell (our requested item) is its upper side—marked in red. Then, the black cell must find the closest white cell. Since the distance from the optimal side of the black cell to each white cell is the same, we are in a deadlock situation. Our deadlock prevention algorithm leads the black cell to randomly select one of the white cells. The black cell chooses the white cell at the bottom and moves it closer to the optimal side—as we observe in the second frame. This continues until the white cell reaches the optimal side shown in the fifth frame. When the black cell moves to the position of the empty cell, the algorithm must select the next white cell considering its distance to the optimal side. At this step, since the white cell at the top of the board is closer to the optimal side, it is selected and moves toward the optimal side of the black cell—as frames 7 through 9 show. This cycle of searching–moving continues until the item exits the board.

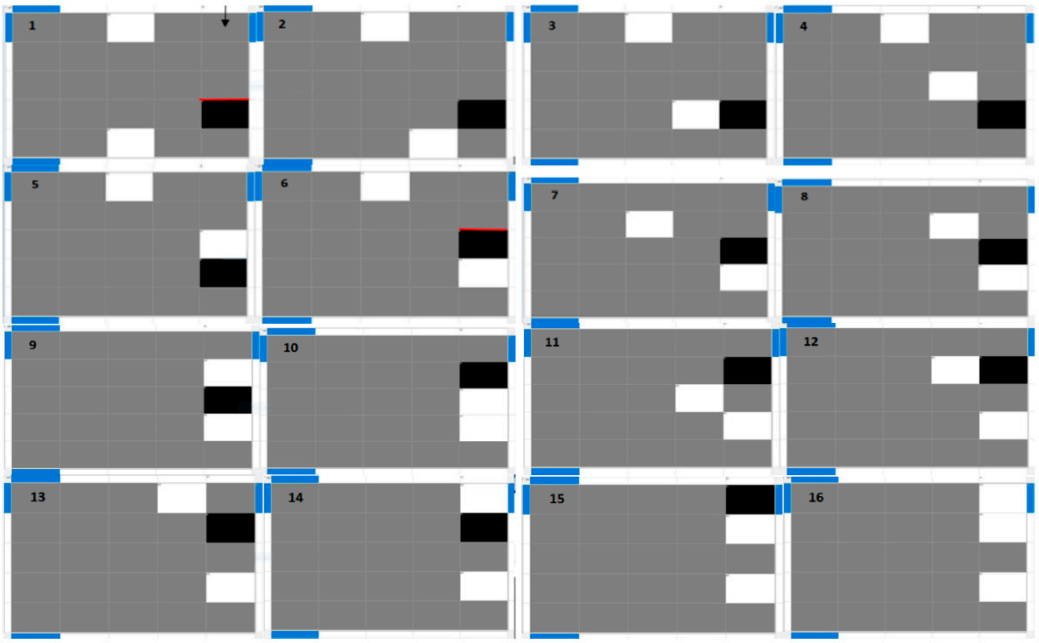

Figure 10. Example 1.

3.5.2. Example 2

In this example (Figure 11), we have one black cell and five white cells. The black cell should be retrieved from gray cell number 3. Based on the retrieval point, the black cell has two optimum sides, and each optimal side has a different closest white cell. Using our searching–moving phases, we obtain five possible paths to retrieve the black cell from the puzzle (Figure 12). Considering the required number of movements associated with each path and each white cell, shown in Table 1, our algorithm selects path (d), meaning that the black cell must seize white cell number 1 in the searching phase. Indeed, the main reason for the optimality of path (d) is using the greatest number of 3 moves to retrieve the requested item.

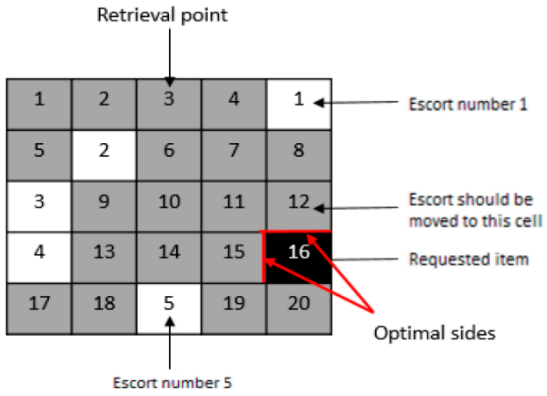

Figure 11. Example 2.

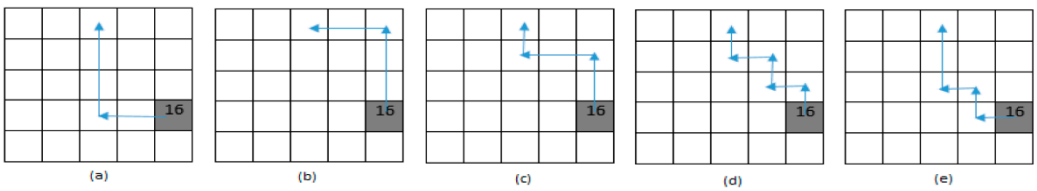

Figure 12. Possible path to retrieve the requested item in example 2. (**a**) Path a; (**b**) Path b; (**c**) Path c; (**d**) Path d; (**e**) Path e.

Table 1. The required number of movements based on selected empty cell and path in example 2.

Path	(a)	(b)	(c)	(d)	(e)
Number of movements with white cell number 1	23	21	19	15	19
Number of movements with white cell number 5	21	19	21	17	17

4. Analysis and Results

4.1. Simulation Results

In this section, we investigate how the number of empty/white cells and the puzzle size affect the number of movements needed to retrieve black cells. We developed an object-oriented program to simulate more than 50,000 iterations using different puzzle configurations. As Figure 13 shows, we first generate a puzzle with a specific size and number of empty cells. Then, we randomly place the white cells, three black cells and a retrieval point into the puzzle. Finally, we take the average of the number of movements we need to retrieve each of the black cells. In each iteration, we randomly change the locations of the white and black cells. After 100 iterations, we find the average retrieval movement (ARM) for each one of our puzzle configurations.

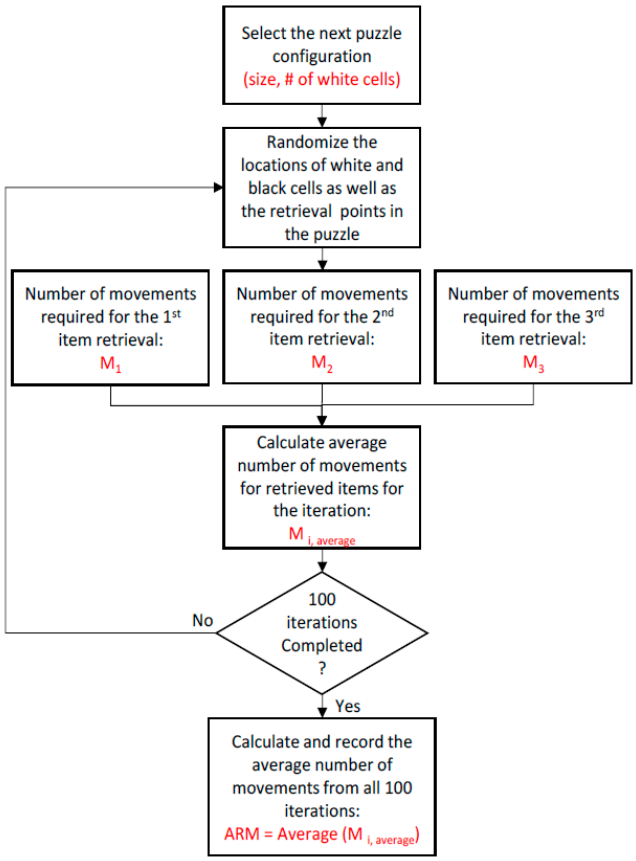

Figure 13. Steps of simulation.

To illustrate the simulation process, consider a 6 × 6 puzzle which includes 18 white cells. Figure 14 shows the average number of movements needed for each iteration. For instance, in iteration 34, we have the lowest number of movements needed because the white cells and black cells are randomly located close to each other. The ARM for this puzzle configuration (6 × 6 with 18 white cells) is 10.4.

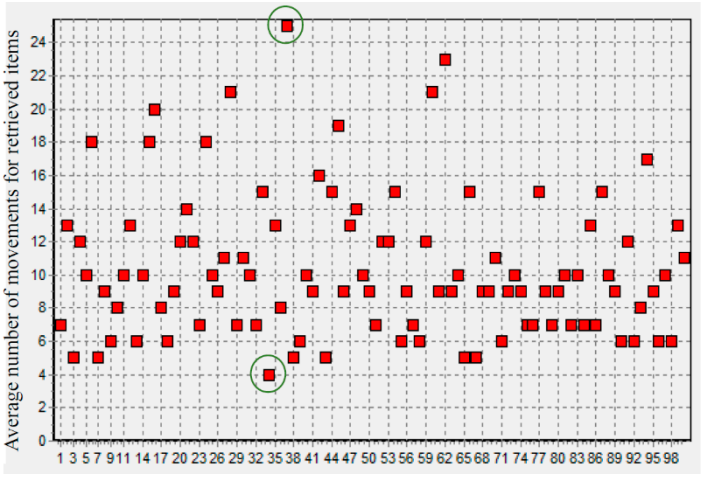

Figure 14. Average number of movements needed to retrieve items in every iteration of a 6 × 6 puzzle with 18 white cells.

Then, we consider how changing the number of white cells affects the number of movements needed in the 6 × 6 puzzle with three black cells. In general, increasing the number of white cells in our puzzles has two opposite effects:

1. Options Effect: increasing the number of white cells increases the options that the black cell has for seizing the nearest white cell;
2. Deadlock Effect: increasing the number of white cells increases the possibility of reaching a deadlock.

According to Figure 15, as the number of white cells grows from one to three, the ARM slightly increases because the number of deadlock situations increases, and the Deadlock Effect outweighs the Options Effect. However, when the number of white cells grows further from 3 to 25, the Options Effect defeats the Deadlock Effect, and the ARM sharply decreases. However, by increasing the number of white cells above 25, the Options Effect becomes less influential, and the ARM reduces slightly.

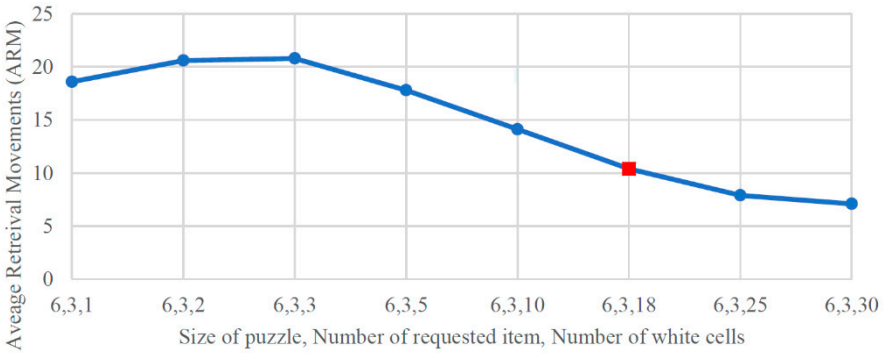

Figure 15. The effect of the number of white cells on the ARM in a 6 × 6 puzzle.

Next, consider a 12 × 12 puzzle. As Figure 16 shows, the effect of the number of white cells on the ARM is similar to Figure 15; first, a slight increase in the ARM is followed by a sharp decrease (from 3 white cells to 90 white cells), and after a specific point (90 white cells), increasing the number of white cells decreases the ARM slightly.

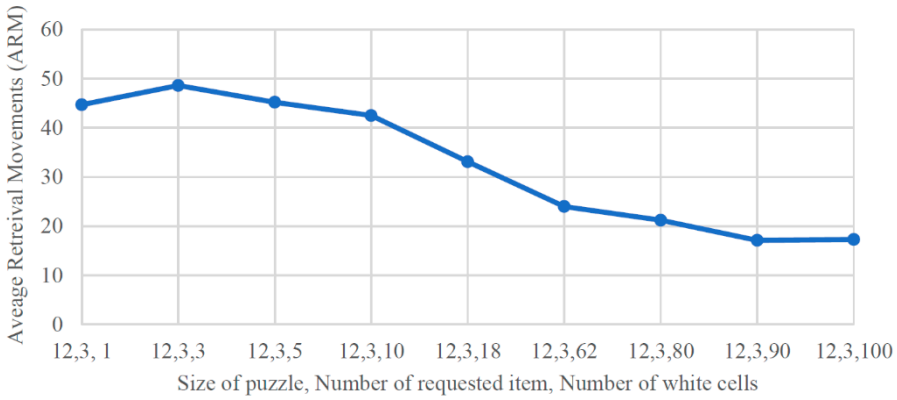

Figure 16. The effect of the number of white cells on the ARM in a 12 × 12 puzzle.

However, the effect of the number of white cells on the ARM for a 20 × 20 puzzle is more interesting. According to Figure 17, as the number of white cells grows from 3 to 10, the Options Effect becomes more influential, and an initial increase in the ARM is followed by a sharp decrease. Then, the Options Effect loses its mastery, and the decreasing trend becomes slight (from 10 to 40 white cells). The trend thus far is what we observed in 6 × 6 and 12 × 12 puzzles. However, in a 20 × 20 puzzle, if we increase the number of white cells above 40, again, a new sharp decrease in the ARM occurs, and this sharp decrease is again followed by another slight decrease from 200 white cells to 300 white cells. Indeed, above 40 white cells, the Options Effect outweighs the Deadlock Effect, and the sharp decrease continues with the increasing white cells until there are 200 of them; at this point, increasing the number of white cells becomes detrimental to the system due to a powerful Deadlock Effect. Therefore, we can conclude that although the Options Effect (Deadlock Effect) decreases (increases) the ARM in general, its strength does not monotonically increase (decrease) in the number of white cells. We observe the same trend as Figure 17 for our 25 × 25 puzzle in Figure 18.

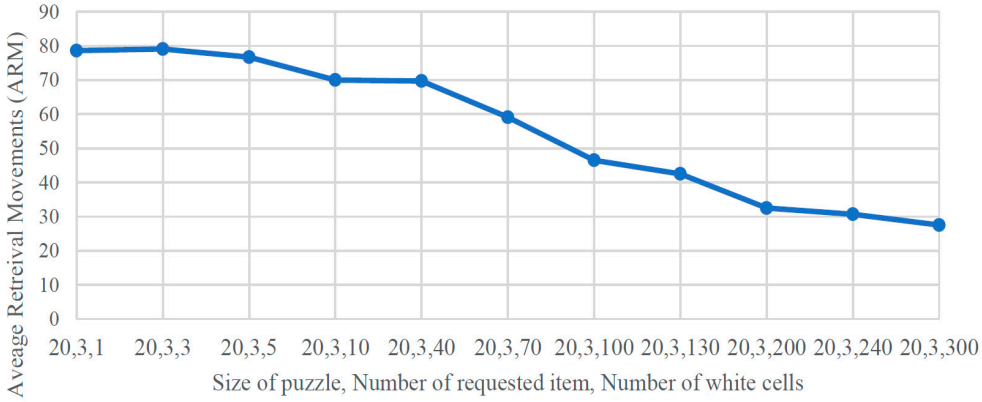

Figure 17. The effect of the number of white cells on the ARM in a 20 × 20 puzzle.

Therefore, when the puzzle size is relatively small, "the number of white cells–ARM" curves present three different trends (slight increase → sharp decrease → slight decrease). However, as the puzzle size grows, the decreasing part of the previous pattern repeats itself (slight increase → sharp decrease → slight decrease → sharp decrease → slight decrease).

In HDSS, we always face a crucial trade-off; on the one hand, we aim to reduce the ARM, and on the other hand, we desire a higher storage density. Using the above-mentioned pattern, we can balance this trade-off and find the profit-maximizing number of empty cells in our puzzle.

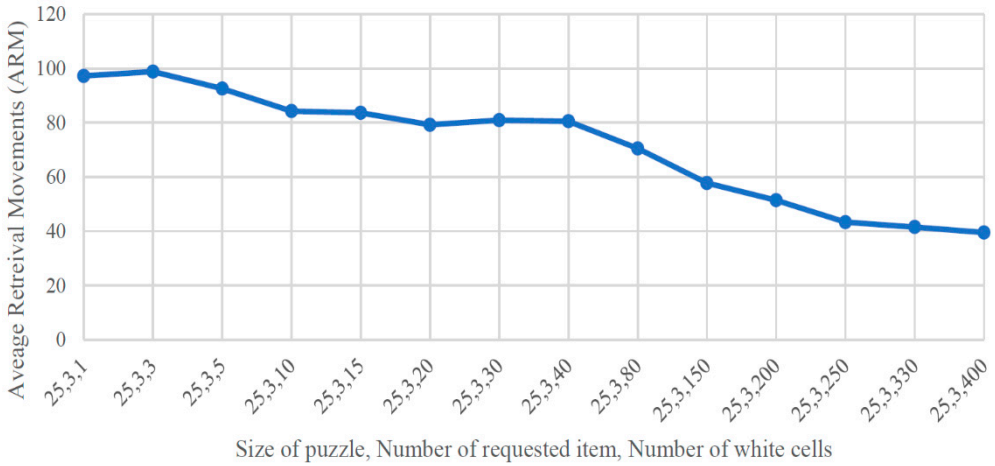

Figure 18. The effect of the number of white cells on the ARM in a 25 × 25 puzzle.

4.2. Comparison of Our Algorithm with Previous Studies

In this subsection, we aim to compare the result of our proposed approach with the results of the contemporary efficient algorithm introduced by Gue and Kim [6]; they found the optimal ARM for a puzzle network including one retrieval point at the lower left corner of the puzzle, one white cell at the retrieval point and one black cell. This optimal ARM, N^*, can be calculated as

$$N^* = \begin{cases} 6i + 2j - 13 & \text{if } i > j \\ 8i - 11 & \text{if } i = j \\ 6j + 2i - 13 & \text{if } i < j \end{cases}$$

where i and j are the row and the column of the black cell counted from the retrieval point. In Table 2, we compare the results of our algorithm to the above optimal formula.

Table 2. Comparison of our results with the optimum mathematical solution obtained by [6].

Case Number	Black Cell Location (*i*,*j*)	Mathematical Optimum Number of Movements	Software Program Simulation Results
1	5,5	29	29
2	15,15	109	109
3	25,25	189	189
4	50,50	389	389
5	5,15	87	87
6	15,25	167	167
7	25,35	247	247
8	35,50	357	357
9	15,5	87	87
10	25,15	167	167
11	35,25	247	247
12	35,50	297	297

As Table 2 shows, our algorithm obtains the same results as the formulation of Gue and Kim [6]. The main reason is that in our algorithm, we followed their main conclusion stating that the optimal puzzle must make the maximum number of 3 moves possible to retrieve the called items.

Taylor and Gue [23] studied how the location of the white cells impacts the average retrieval time for two configurations: (i) random distribution of empty cells and (ii) input/output distribution where white cells are at the retrieval point. We compared the results of our algorithm with their results for these two configurations. As Figure 19 shows, our algorithm outperforms their results in the input/output distribution configuration. However, the results are similar in the random distribution configuration (Figure 20).

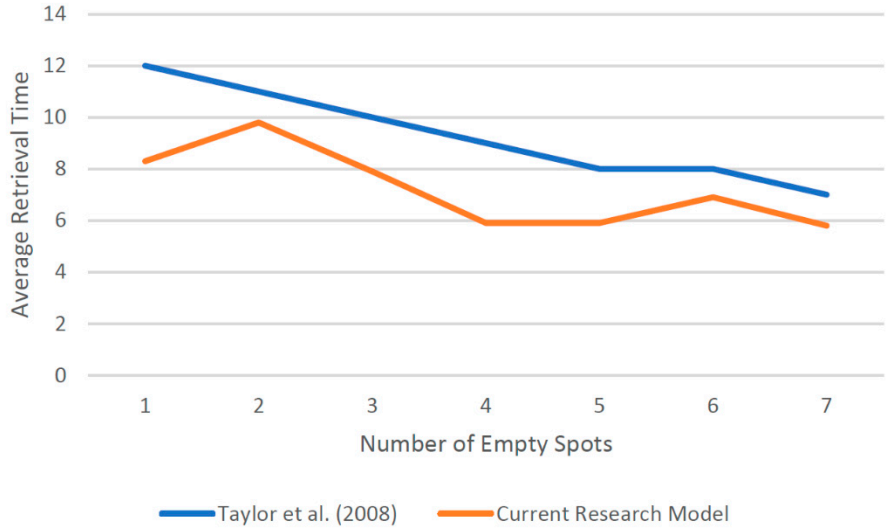

Figure 19. Comparison of our algorithm's and Taylor and Gue's (2008) results in the I/O distribution configuration.

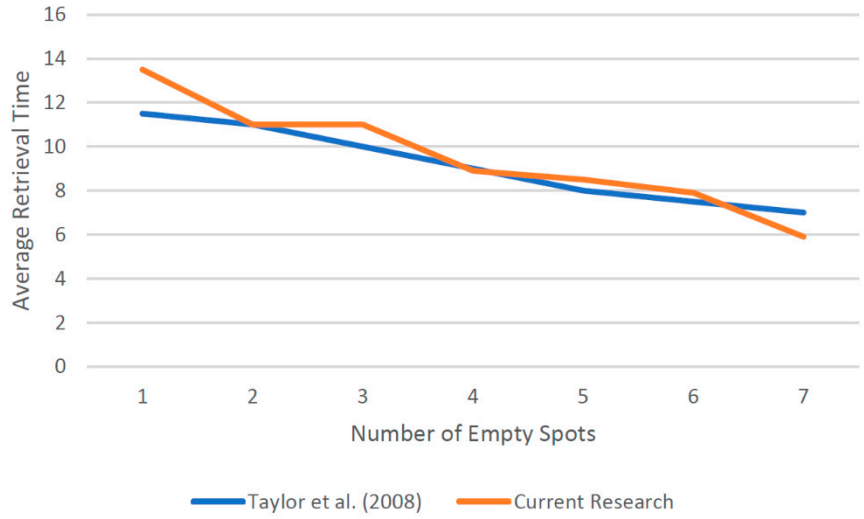

Figure 20. Comparison of our algorithm's and Taylor and Gue's (2008) results in the random distribution configuration.

5. Conclusions

This paper developed an item retrieval algorithm in high-density storage systems (HDSS) by preferring three moves to five moves in directing the seized white cell to the optimal side of the required cell. The main contributions that our algorithm adds to the previous ones are listed below:

1. Including a deadlock prevention algorithm into our main algorithm;
2. Retrieving items from all sides of the puzzle;
3. Introducing a new search algorithm to find the nearest empty cell to the requested item.

Using simulation, we emphasized the effect of empty cells on the retrieval time. We observed that in the presence of less than three white cells, increasing the number of white cells increases the ARM. As the number of white cells grows above three, increasing the number of white cells decreases the ARM sharply. However, above a specific number of white cells, this decreasing trend becomes less significant. We explained this pattern by two effects of increasing the white cells in our algorithm: Options Effect and Deadlock Effect. By growing the puzzle size, the above-mentioned pattern repeats again—we observe another

"sharp decrease in the ARM followed by a slight decrease". This finding can play a key role in balancing the "puzzle density–ARM" trade-off in storage systems.

In this paper, we assumed the number of requested items is constant. One can extend our algorithm by considering the number black cells as a variable. Another assumption behind our results is that we can move all cells in the puzzle unlimited times; however, due to safety instructions in real-world cases, some specific items might not move more than a few times.

Author Contributions: Conceptualization, E.S. and M.Z.; methodology, E.S.; software, E.S.; validation, E.S.; formal analysis, E.S. and M.Z.; writing—original draft preparation, E.S. and M.Z.; writing—review and editing, M.Z.; visualization, E.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Srinivas, S.S.; Marathe, R.R. Moving towards "mobile warehouse": Last-mile logistics during COVID-19 and beyond. *Transp. Res. Interdiscip. Perspect.* **2021**, *10*, 100339.
2. Seibold, Z.; Stoll, T.; Furmans, K. Layout-optimized sorting of goods with decentralized controlled conveying modules. In Proceedings of the 2013 IEEE International Systems Conference (SysCon), Orlando, FL, USA, 15–18 April 2013.
3. Gue, K.R. Very high density storage systems. *IIE Trans.* **2006**, *38*, 79–90. [CrossRef]
4. Gue, K.R.; Furmans, K.; Seibold, Z.; Uludağ, O. GridStore: A puzzle-based storage system with decentralized control. *IEEE Trans. Autom. Sci. Eng.* **2013**, *11*, 429–438. [CrossRef]
5. Hearn, R.A.; Demaine, E.D. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.* **2005**, *343*, 72–96. [CrossRef]
6. Gue, K.R.; Kim, B.S. Puzzle-based storage systems. *Nav. Res. Logist.* **2007**, *54*, 556–567. [CrossRef]
7. Gue, K.R.; Furmans, K. Decentralized control in a grid-based storage system. In *IIE Annual Conference Proceedings*; Institute of Industrial and Systems Engineers (IISE): Peachtree Corners, GA, USA, 2011; p. 1.
8. Rohit, K.V.; Taylor, G.D.; Gue, K.R. Retrieval Time Performance in Puzzle-Based Storage Systems. In *IIE Annual Conference Proceedings*; Institute of Industrial and Systems Engineers (IISE): Peachtree Corners, GA, USA, 2010; p. 1.
9. Gue, K.R.; Uludag, O.; Furmans, K. A high-density system for carton sequencing. In Proceedings of the International Material Handling Research Colloquium 2012, Gardanne, France, 25–28 June 2012.
10. Furmans, K.; Gue, K.R.; Seibold, Z. Optimization of failure behavior of a decentralized high-density 2D storage system. In *Dynamics in Logistics*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 415–425.
11. Uludag, O. GridPick: A High Density Puzzle Based Order Picking System with Decentralized Control. Ph.D. Thesis, Auburn University, Auburn, AL, USA, 9 January 2014.
12. Krühn, T.; Sohrt, S.; Overmeyer, L. Mechanical feasibility and decentralized control algorithms of small-scale, multi-directional transport modules. *Logist. Res.* **2016**, *9*, 1–4. [CrossRef]
13. Mirzaei, M.; De Koster, R.B.; Zaerpour, N. Modelling load retrievals in puzzle-based storage systems. *Int. J. Prod. Res.* **2017**, *55*, 6423–6435. [CrossRef]
14. Hao, G. GridHub: A Grid-Based, High-Density Material Handling System. Ph.D. Thesis, University of Louisville, Louisville, KY, USA, May 2020.
15. Shekari Ashgzari, M.; Gue, K.R. A puzzle-based material handling system for order picking. *Int. Trans. Oper. Res.* **2021**, *28*, 1821–1846. [CrossRef]
16. Azadeh, K.; De Koster, R.; Roy, D. Robotized and automated warehouse systems: Review and recent developments. *Transp. Sci.* **2019**, *53*, 917–945. [CrossRef]
17. Boysen, N.; De Koster, R.; Weidinger, F. Warehousing in the e-commerce era: A survey. *Eur. J. Oper. Res.* **2019**, *277*, 396–411. [CrossRef]
18. 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]
19. Manzini, R.; Accorsi, R.; Gamberi, M.; Penazzi, S. Modeling class-based storage assignment over life cycle picking patterns. *Int. J. Prod. Econ.* **2015**, *170*, 790–800. [CrossRef]

20. Seibold, Z. *Logical Time for Decentralized Control of Material Handling Systems*; KIT Scientific Publishing: Karlsruher, Germany, 2016.
21. 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]
22. Zaerpour, N.; Yu, Y.; de Koster, R.B. Optimal two-class-based storage in a live-cube compact storage system. *IISE Trans.* **2017**, *49*, 653–668. [CrossRef]
23. Taylor, G.D.; Gue, K.R. The effects of empty storage locations in puzzle-based storage systems. In *IIE Annual Conference Proceedings*; Institute of Industrial and Systems Engineers (IISE): Peachtree Corners, GA, USA, 2008; p. 519.