Puzzle-Based Storage Systems

Kevin R. Gue
Byung Soo Kim

Department of Industrial & Systems Engineering
Auburn University
Auburn, AL 36849-5346

February 6, 2007

Abstract

We introduce and develop models for a physical goods storage system based on the 15-puzzle, a classic children's game in which 15 numbered tiles slide within a 4×4 grid. The objective of the game is to arrange the tiles in numerical sequence, starting from a random arrangement. For our purposes, the tiles represent totes, pallets, or even containers that must be stored very densely, and the objective is to maneuver items to an input-output point for retrieval or processing. We develop analytical results for storage configurations having a single empty location (as in the game) and experimental results for configurations with multiple empty locations. Designs with many empty locations can be made to form aisles, allowing us to compare puzzle-based designs with traditional aisle-based designs found in warehousing systems.

1 Puzzle-based storage systems

When an item is requested in a traditional storage system, a vehicle (a forklift driver, or perhaps a shuttle within an automated storage and retrieval system) moves to the location, extracts the item, and transports it to an input-output (I/O) point. Such systems necessarily contain aisles in which the vehicle moves, and these aisles occupy space that could otherwise be used to store items. Thus, aisles reduce the overall storage *density*—the number of items stored per square or cubic foot—of the space. Density is an important consideration in storage systems design because low density means more space is required to store the same number of products, and this leads to a larger space. A larger storage system increases both fixed and variable costs: fixed, because it is more expensive to build, and variable, because workers or vehicles must travel farther to retrieve items.

There are several ways to increase density in a storage system. The first is to reduce the widths of aisles, thus devoting more space to storage and less to aisles; but narrow aisles may not be desireable for two reasons: (1) they usually require specialized (and more expensive) vehicles, and (2) they often oblige one-way travel within aisles, which tends to increase congestion and travel distances. The second is to increase the *lane depth*, or the maximum number of items stored (one behind another) per pick location in an aisle. For rack-based storage in a warehouse, this is commonly achieved with *double-deep* storage rack, in which two pallets (one behind the other) are accessible from each pick location. A storage device called *push-back rack* allows triple-deep or deeper storage—a spring loaded tray stores the deepest pallets, and helps to extract them when required. Lanes deeper than five typically require gravity flow rack or a fully automated system. Gue [5] showed that if items are stored such that k is the maximum lane depth, the density of the storage space can be no greater than $2k/(2k+1)$, leading to theoretical density limits of $2/3$ and $4/5$ for single- and double-deep storage systems, respectively.

At the upper limit of storage density are automated systems based on the 15-slide puzzle (see Figure 1). Here, items are stored in a grid in which only one location is open, and so density is $(n-1)/n$, where n is the number of cells in the grid. The system repeatedly moves

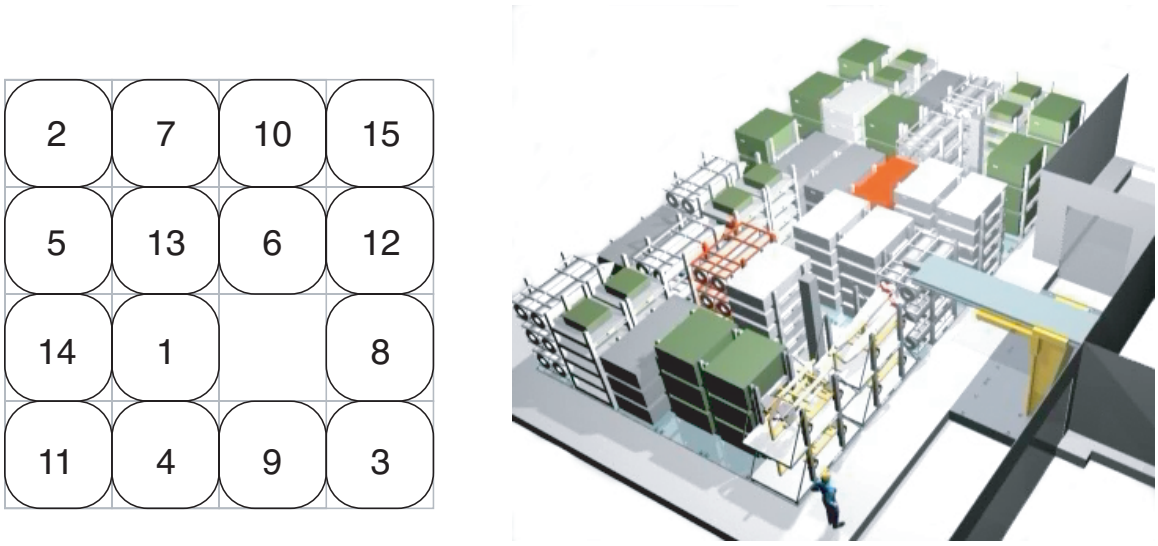

Figure 1: On the left, a representation of the 15-puzzle; on the right, a computer representation of NAVSTORS, a (6×6) puzzle-based storage system under development for the U.S. Navy.

items into the open location to achieve a desired configuration, typically to maneuver one item to the I/O point. Two such systems are under development for the U.S. Navy: the first (called NAVSTORS) will store missiles under a launcher; the second (NAVPAK) will store smaller items in a storeroom. We are also aware of a tote storage system that uses motorized rollers to achieve puzzle-based movement.

Our purpose is to investigate storage systems based on a slide-puzzle architecture, answering questions such as,

- How should one move a chosen item to the I/O point? Is there an optimal way?
- If more than one cell is open, does performance improve? By how much?
- What is the expected retrieval time from a puzzle-based system, and how does it compare with aisle-based systems?

Our intention is not so much to describe the performance of the industrial systems we mention above, although our models could be used to do that, but rather to investigate the

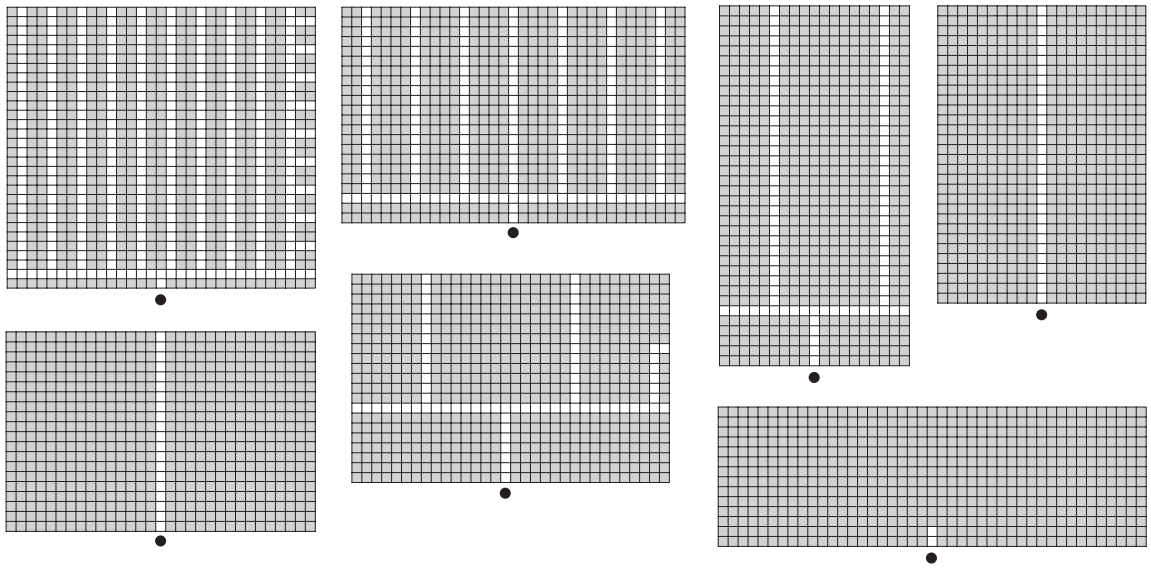

Figure 2: Several ways to store 600 items. Input-output points are indicated by the black dots. Which design has the lowest expected retrieval time?

performance of theoretical storage systems at the upper limits of storage density. In particular, we are interested in the fundamental tradeoff between storage density and expected retrieval item. For systems with "very high density," which Gue [5] defines as those requiring movement of interfering items during some retrievals, higher density means moving more interfering items and increased retrieval times; but it also means the storage space is smaller and items are closer to the I/O point, which tends to reduce retrieval times. For example, consider the six storage configurations in Figure 2, each of which stores 600 items. Which of the six has the lowest expected retrieval time? (Answer in Section 5.)

Our contribution is twofold. First, we develop results for puzzle-based storage systems, including an optimal algorithm for systems having a single open cell. We also consider puzzle-based systems with more than one empty cell, and show that expected retrieval time decreases with the number of empty cells. Second, we describe the relationship between storage density and expected retrieval time for both puzzle- and aisle-based systems. We establish something of a continuum of storage configurations, from single-deep, aisle-based storage, which has minimum density, to puzzle-based systems with one open cell, which have

maximum density. Our results suggest that aisle-based systems are preferred when density is not constrained, and that puzzle-based systems are best if required density exceeds about 90 percent.

We begin in the next section with a discussion of related literature. In Section 3 we give theoretical results for storage systems having a single open cell, as in the game, and then we investigate systems with multiple open cells in Section 4. In Section 5 we compare puzzle-based systems with aisle-based systems having different lane depths. We offer conclusions in Section 6.

2 Related literature

Until late 2006, the 15-puzzle was thought to have been invented in the 1870's by the great puzzlemaster Sam Loyd, who claimed to be its inventor beginning in 1891. In a detailed and fascinating recent book, Slocum and Sonneveld [14] expose Loyd's claim as false, and attribute the invention to Noyes Palmer Chapman, postmaster of Canastota, N.Y., who probably invented the puzzle before 1874. What is not in dispute is that the 15-puzzle quickly became a worldwide craze in 1880, surpassing by far the Rubik's cube craze in the 1980's. In the original 15-puzzle, tiles were removed from the wooden box containing them and reinserted in a random sequence. The goal then was to sort the tiles into their proper sequence, but certain starting arrangements made the puzzle impossible to solve. In jointly published papers, Johnson and Story [9] showed that there exist two groups of sequences, and that sequences in one group cannot be reached from a sequence in the other. Sequences (1, 2, . . . , 13, 14, 15) and (1, 2, . . . , 13, 15, 14) are in different groups, for example. (The result was derived independently by Tait [15] and, according to Slocum and Sonneveld, by Hermann Schubert in the newspaper *Hamburgischer Correspondentt.* All four articles were published within a month of each other.) In an apology to their readers, the Editors of the 1879 issue of the American Journal of Mathematics containing Johnson and Story's papers state that they "thought they would be doing no disservice to their science, but rather

promoting its interests by exhibiting this *à priori* polar law under a concrete form, through the medium of a game which has taken so strong a hold upon the thought of the country that it may almost be said to have risen to the importance of a national institution." They added that the 15-puzzle "may safely be said to have engaged the attention of nine out of ten persons of both sexes and of all ages and conditions of the community" (page 404).

Recent literature on the 15-puzzle or its $n^2 - 1$ generalization can be divided into three groups: (1) recreational mathematics, (2) generic computational search methods, especially enumerative methods, and (3) efficient algorithms to solve the puzzle. For a discussion in the recreational mathematics literature, see Gardner [2]. Research in computational methods has used the 15-puzzle as a "workbench" for enumerative methods. The 15-puzzle is especially amenable to this type of research because the problem is simple to state, and yet there is a large number of starting configurations. (The 15-puzzle has approximately 10^{13} feasible configurations, which can be manipulated to achieve the desired ordering of tiles.) Reinefeld [13] gives a complete solution (an enumeration of all feasible configurations and the optimal solution(s) for each) to the 8-puzzle. Here, an optimal solution is a sequence of moves that sorts the puzzle in the minimum number of moves. He reports that there are 500,880 optimal solutions to the 181,440 configurations, and that an optimal solution requires, on average, 21.97 moves. Karlemo [10] gives a complete solution for the $3 \times 4 - 1$ puzzle. Several authors have considered how to solve the puzzle with the minimum number of moves. Parberry [11] describes an algorithm to solve $n^2 - 1$ puzzles—not necessarily optimally—that uses at most $5n^3$ moves. Ratner and Warmuth [12] prove that solving an $(n^2 - 1)$-puzzle in the minimum number of moves is NP-Hard. Gasser [4] showed that every 15-puzzle may be solved in fewer than 88 moves, and that some configurations require as many as 80 moves. Karlemo [10] showed that every 24-puzzle may be solved in fewer than 210 moves.

The basic problem for all of these papers is to arrange the tiles of the puzzle in a certain sequence, starting from another sequence. When viewed as a physical goods storage system, however, the puzzle presents a different and simpler problem—how to move the tiles to retrieve an item as quickly as possible? In other words, we are not concerned with the

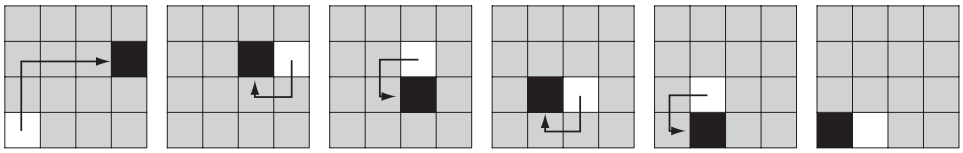

Figure 3: Puzzle movement in a storage system. The empty space (white) effectively "escorts" the requested item (black) to the input-output point in the lower-left corner.

internal arrangement of items in the grid, but only with the location and movement of a particular, requested item.

Our problem is also related to robot motion planning problems [8], which have a large literature. Of special interest to us is the puzzle *Rush Hour*, where the goal is to move a particular piece from one end of the board to the other—which is similar to our objective—and this requires moving interfering items out of the way. "Vehicles" may occupy two grid squares (a "car") or three squares (a "truck"), and may move vertically or horizontally, but not both. Flake and Baum [1] showed that the original Rush Hour (with cars and trucks) is PSPACE-Complete (a stronger condition than NP-Completeness; see Garey and Johnson [3]); Hearn and Demaine [7] show that Rush Hour with cars only is PSPACE-Complete. Tromp and Cilibrasi [16] consider the problem of Unit Rush Hour, in which each vehicle is a single square. They leave the complexity of this problem as an open question. Our problem is different than Rush Hour in that our "vehicles" are of unit-size, and each may move in any direction. This makes our problem considerably easier than even Unit Rush Hour, as we are about to show.

3 Puzzle systems with a single escort

Consider a rectangular storage system with every location occupied by an item, except for the lower left location, which is empty at the beginning of a retrieval and acts as the I/O point. Retrieving an item may be viewed as a series of moves of the empty location, rather than the items (see Figure 3). A more natural choice for the I/O point, of course, would be

the center of the longest edge, but we choose the lower left corner for ease of analysis. Our computational results are based on the better choice of I/O point.

3.1 Retrieving items

Retrieving an item consists of two phases: first, we must get the open location to the item, and second, we must use the open location to move the item to the I/O point. The second operation may be viewed as "escorting" the item to the I/O point because, by the very nature of slide-puzzle movement, the open location must precede the item on every step of its path. Hereafter, then, we refer to an open location as an *escort*.

Our analysis assumes that the escort begins every retrieval at the I/O point. The escort need not be there when a retrieval begins, of course, but our assumption is reasonable given the following observation: When any item is moved to the I/O point, the last move places the escort in one of the locations adjacent to the I/O point, depending on the location of the item after the penultimate move. If the next request is already in the queue, the escort will begin its next retrieval from that location. If the I/O point is on the side of the storage grid rather than on a corner, then the I/O point is a sort of "average location" of its three adjacent locations.

In what follows, we distinguish between *escort-moves* and *item-moves*: an escort-move is a single move of one item that causes an escort to move to an adjacent location; an item-move is comprised of a sequence of escort-moves, resulting in the requested item moving to an adjacent location. Depending on the location of the escort, an item-move may require many escort-moves. In general, we are interested in minimizing the number of escort-moves required to move a desired item from its current location to the I/O point, because this is approximately proportional to the expected time to retrieve the item. We assume throughout that one escort-move takes one time unit.

Define an optimal path for an item to be a series of grid locations the item will occupy as it travels in a minimum number of escort-moves from its current position to the I/O point. It is easy to see that

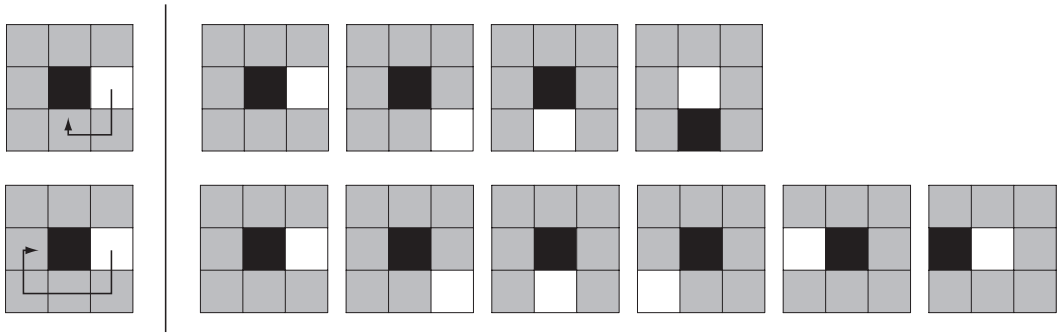

Figure 4: Series of escort-moves to move an item from its current location. In the top series, we seek to move the black item downward; in the bottom series, leftward.

Observation 1 *If the I/O point is located in the bottom left corner of a grid, an optimal path for any item contains only leftward and downward item-moves.*

Therefore, if the I/O point is grid location $(1, 1)$,

Observation 2 *The length of an optimal path for an item in location* (i, j) *is* $i + j - 2$*, which corresponds to the rectilinear distance between the location and the I/O point.*

Once the escort reaches the item, there are only two sequences of escort-moves necessary to get the item to the I/O point. Hayes [6] describes series of escort-moves to move an item from one position to another given that the escort is adjacent to the item. Figure 4 shows two of those series of interest to us. The first moves an item from its current location to a location diagonal to the escort, and it takes 3 moves. In the figure, we wish to move the black item to the location below it. The second moves the item of interest to a location opposite the escort, and it takes 5 moves. It is easy to see that any path to the I/O point can be accomplished with a sequence of 3-moves and 5-moves. Items adjacent to the I/O point when the escort is there can perform a "1-move" in which the escort and the item trade locations. One-moves are precluded from other optimal retrieval sequences because Observation 1 implies that the escort will be above or to the right of the item after every item-move.

Because every optimal path contains the same number of item-moves (corresponding to

the rectilinear distance between the item and the I/O point), an optimal solution minimizes $5x_1 + 3x_2 + x_3$, subject to $x_1 + x_2 + x_3 = i + j - 2$ and other constraints we will get to shortly, where $x_1, x_2, \text{and}, x_3$ are the required numbers of 5-, 3-, and 1-moves, respectively.

The following algorithm describes a retrieval strategy based on this observation. The idea

Algorithm 1 SINGLE-ESCORT

1: **if** $i \geq j$ **then**
2: Move the escort to location $(i-1, j)$, then move the escort into location (i, j).
3: Perform 3-moves downward and leftward until $j' = 1$.
4: Perform 5-moves leftward until the item is at the I/O point.
5: **else**
6: Move the escort to location $(i, j-1)$, then move the escort into location (i, j).
7: Perform 3-moves leftward and downward until $i' = 1$.
8: Perform 5-moves downward until the item is at the I/O point.
9: **end if**

behind SINGLE-ESCORT is to begin the path in a direction that maximizes the number of 3-moves, thereby minimizing the number of necessary 5-moves, and consequently minimizing the total number of moves. We can show that this is an optimal strategy.

Theorem 1 *The minimum number of moves to retrieve an item in location* (i, j) *is*

$$\begin{array}{ll} 6i + 2j - 13 & i > j \\ 6j + 2i - 13 & j > i \\ 8i - 11 & i = j. \end{array}$$

Proof Notice first that an optimal path contains $i + j - 2$ moves, and this is the rectilinear distance between the item and the I/O point. We have shown that every move of the item involves either 3 or 5 moves of the escort, and that an optimal path includes only item-moves down and to the left.

Once the escort is adjacent to the item, moving the item in an improving direction (either left or down) takes at least 3 escort-moves, and 3-moves can continue only until the item is in a location with $i = 1$ or $j = 1$ (directly above or to the right of the I/O point). Remaining item-moves require 5 escort-moves, until the item is in the I/O point.

Because the total number of item-moves is fixed, an optimal strategy simply maximizes the number of 3-moves. A retrieval has three phases: moving the escort to the item, performing 3-moves until either $i = 1$ or $j = 1$, and then performing 5-moves to the I/O point. There are three cases:

If $i > j$, the initial move must be left rather than down, in order to maximize the number of 3-moves. It takes $i + j - 2$ escort-moves to move the item to the left. Subsequent 3-moves occur in pairs, moving the item down and then left, until a 5-move is necessary to continue. There are $j - 1$ pairs of 3-moves, for a total of $6(j - 1) = 6j - 6$ escort-moves. The item is now at location $(i - j, 1)$ and so it requires $i - j - 1$ 5-moves for $5(i - j - 1) = 5i - 5j - 5$ additional escort-moves, for a total of $(i + j - 2) + (6j - 6) + (5i - 5j - 5) = 6i + 2j - 13$.

The remaining two cases ($i < j$ and $i = j$) are analogous. □

From the proof we can see that algorithm SINGLE-ESCORT is optimal.

If each move takes one time unit, then we can easily compute the expected retrieval time for a randomly chosen item. Figure 5 shows the expected retrieval time (number of escort-moves) for storage grids of size 4×4 up to 40×40. Points along the bottom of the plot are a sort of "efficient frontier" of grids, which is comprised of square and nearly-square grids of different sizes. This is expected, because a square grid tends to minimize the number of 5-moves necessary to retrieve the items inside. The plot shows that

Observation 3 *For the best designs, expected retrieval time increases approximately with the square root of capacity.*

Theorem 1 shows that retrieval time is a linear combination of length of the sides of the grid; capacity, of course, increases with the square of the length of the sides, so the observation agrees with our intuition.

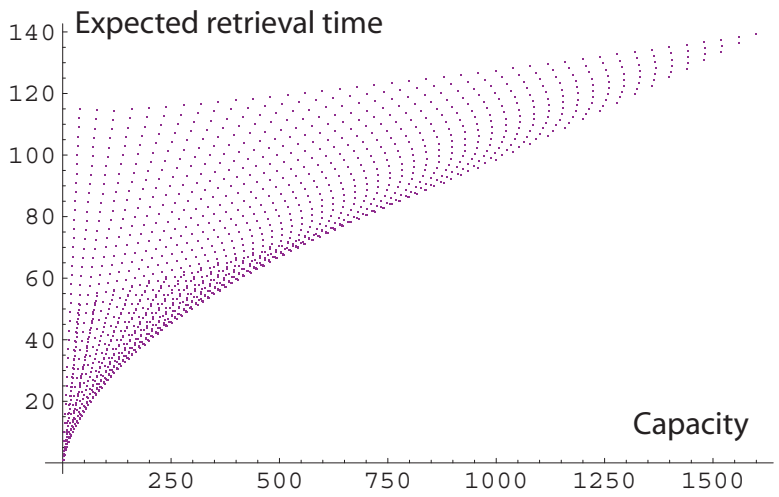

Figure 5: Expected retrieval time for items in a puzzle storage system with one escort.

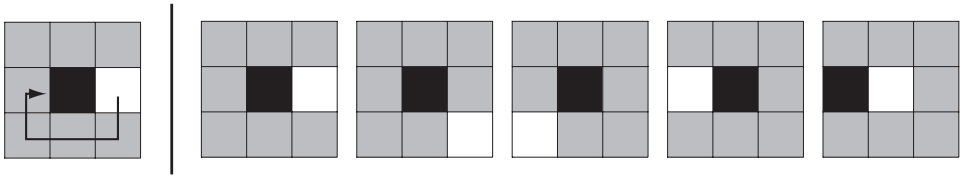

Figure 6: For some systems, it is possible to move the escort more than one location in a time unit, as long as the positions are in a line. Here, Items 3 and 4 move together in a single escort-move.

3.2 Improvements with block movement

For an automated system (and for the 15-puzzle itself) it might be possible to move the escort more than one location in a unit of time if the path is a line. For example, if the escort is in location $(3, 1)$ (see Figure 6), it can be moved to location $(1, 1)$ by *simultaneously moving* items in locations $(1, 1)$ and $(1, 2)$.

Such a system will have slightly improved retrieval times, as follows. Note first that only the escort can "jump" within the grid, not an item being retrieved, and so an optimal path for the item still has the same number of item-moves. The first step is still to move the escort to the item, but now this can be done in one move if the item is in location $(k, 1)$

or $(1, k)$, or in two moves otherwise. The item still must be "puzzled" to the I/O point, as before, but moving the item to a location opposite the escort takes only four escort-moves, rather than five as before (see Figure 6). The 3-move is unchanged.

Theorem 2 *When block movement is possible, the minimum number of moves to retrieve an item in location* (i, j) *is, if* $i \neq 1$ *and* $j \neq 1$,

$$\begin{array}{ll} 4i + 2j - 8 & i > j \\ 4j + 2i - 8 & j > i \\ 6i - 7 & i = j, \end{array}$$

and, if $i = 1$ *or* $j = 1$,

$$\begin{array}{ll} 4j - 3 & i = 1 \\ 4i - 3 & j = 1. \end{array}$$

Proof As before, we seek to maximize the number of 3-moves along the path. Consider first the cases in which $\{i \neq 1, j \neq 1\}$. If $i > j$, the initial item-move is left, and it takes 2 escort-moves. Subsequent 3-moves occur in pairs, moving the item down then left, until a 4-move is necessary to continue. There are $j - 1$ pairs, for a total of $6(j - 1) = 6j - 6$ moves. The item is now at location $(i - j, 1)$ and so it requires $i - j - 1$ 4-moves for $4(i - j - 1) = 4i - 4j - 4$ additional moves, and a total of $2 + (6j - 6) + (4i - 4j - 4) = 4i + 2j - 8$. The case $\{i < j | i \neq 1, j \neq 1\}$ is analogous. For case $i = j$, assume (arbitrarily) we move the item downward first. Getting the escort to the item takes 2 escort-moves, putting the item in location $(i, j - 1)$. There are $j - 2$ pairs of 3-moves, plus an additional 3-move to the I/O point, for $6(j - 2) + 3 = 6j - 9$ additional escort-moves, and a total $2 + 6i - 9 = 6i - 7$.

If $i = 1$ (or $j = 1$), the initial item-move requires one block escort move, and is followed by $j - 1$ (or $i - 1$) 4-moves, for a total $1 + 4(j - 1) = 4j - 3$ (or $4i - 3$). □

4 Puzzle systems with multiple escorts

For a storage system, there is no reason to restrict a design to a single escort. Why not two, or three, or ten? The analysis is more difficult now, because (1) we must decide on an initial configuration for the escorts and (2) at each step we must decide which escort to move. In what follows, we disallow block movement.

An ideal initial configuration achieves two objectives. First, it enables the system to get an escort to the requested item as quickly as possible. This lobbies for having the escorts widely dispersed throughout the grid when a request arrives. Second, an ideal configuration allows the system to use escorts in a coordinated way to reduce retrieval time. This will happen if the escorts lie on, or at least near, the optimal path for the requested item, and so we want the escorts generally near the I/O point.

Our models assume that a retrieval begins with all escorts lined up at the I/O point. We believe this is justified because, in addition to meeting the second objective above, it is reasonable to expect that over a series of retrievals, escorts would tend to migrate toward the I/O point anyway. This suggests that, even if one could show it were the best starting configuration, having escorts widely distributed in the storage space would be difficult or impossible to maintain. This is especially true when there is no time between requests to reposition escorts, which is the very situation in which we would want fast retrievals.

We address the second issue—which escort to move at each step—directly in our algorithm.

4.1 Dynamic program

We can determine the optimal retrieval sequence for a puzzle storage system with multiple escorts by solving a dynamic program. Let,

$c(r, s, e)$ be the minimum cost of moving an item in location r to location s using only an escort in location e,

$m(r, s, E)$ be the minimum number of escort-moves required to move an item in location r to location s, given a *set* of escorts in locations E, and

N_r be the neighbor cells of location r.

Then,

$$m(r, s, E) = \begin{cases} \min_{e \in E}\{c(r, s, e)\} & s \in N_r \\ \min_{k \in N_r, e \in E}\{c(r, k, e) + m(k, s, \{E \setminus e\} \cup \{r\})\} & \text{o/w} \end{cases}$$

is a dynamic program solving our problem. The intuition behind the model is that, at every step, we can use any escort to move the requested item to any of its (at most) four neighbor cells. After each step, the chosen escort is located in the cell formerly occupied by the requested item; and we have another decision to make. Each decision is comprised of (1) which neighbor cell to move into and (2) which escort to use. To compute the cost of a step, we need only know the cost of moving the item to a neighbor cell using a particular escort e, which is easy to calculate.

The state space is very large due to the many possible locations for a set of escorts, and so we are able to solve only problems with up to six escorts. Figure 7 shows the results for configurations with up to six escorts.

4.2 Heuristic

For larger problems we present a heuristic that seeks the minimum number of escort-moves to move an item in location (i, j) to the I/O point $(1, 1)$, given e escorts lined up at the I/O point. The heuristic applies a different rule for items in each of three regions (see Figure 8). Region A lies above the escorts, but extends only to $j = 4$, with a stairstep downward as shown in the figure. Region C is roughly triangular as shown; Region B is comprised of the remaining items.

In Region A, we retrieve items in the intuitive way that interfering items are moved down and to the right, clearing the way for the item to move directly to the I/O point within the

19	21	23	25	29	33	39	45	51
13	15	17	21	25	31	37	43	49
7	9	13	17	23	29	35	41	47
1	5	9	15	21	27	33	39	45
	1	7	13	19	25	31	37	43

16	16	18	22	26	30	34	40	46
10	10	14	18	22	26	32	38	44
4	6	10	14	18	24	30	36	42
1	2	6	10	16	22	28	34	40
		2	8	14	20	26	32	38

15	15	16	19	23	27	31	35	41
9	10	11	15	19	23	27	33	39
4	5	7	11	15	19	25	31	37
1	2	3	7	11	17	23	29	35
			3	9	15	21	27	33

15	15	16	17	20	24	28	32	36
9	10	11	12	16	20	24	28	34
4	5	6	8	12	16	20	26	32
1	2	3	4	8	12	18	24	30
				4	10	16	22	28

15	15	16	17	18	21	25	29	33
9	10	11	12	13	17	21	25	29
4	5	6	7	9	13	17	21	27
1	2	3	4	5	9	13	19	25
					5	11	17	23

15	15	16	17	18	19	22	26	30
9	10	11	12	13	14	18	22	26
4	5	6	7	8	10	14	18	22
1	2	3	4	5	6	10	14	20
						6	12	18

Figure 7: Results from the dynamic program for six configurations. Entries in each cell indicate the number of escort-moves required to bring the item to the I/O point, which is the bottom left cell.

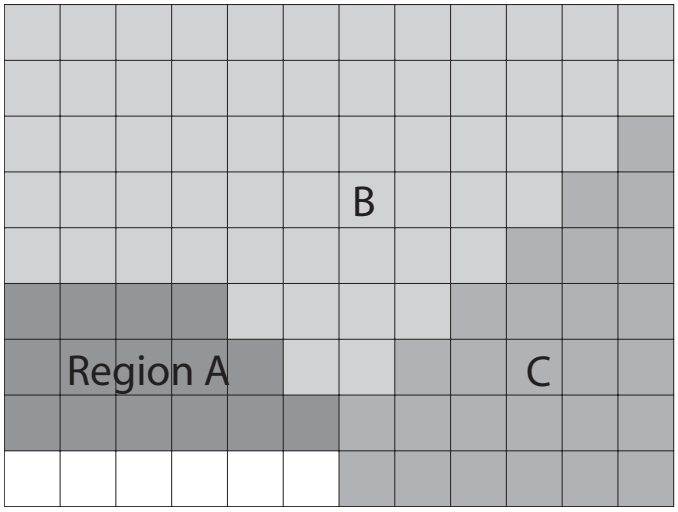

Figure 8: Regions for the multi-escort system heuristic. Region A never extends beyond $j = 4$.

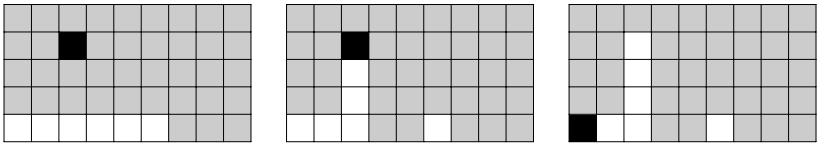

Figure 9: An example retrieval for an item in Region A.

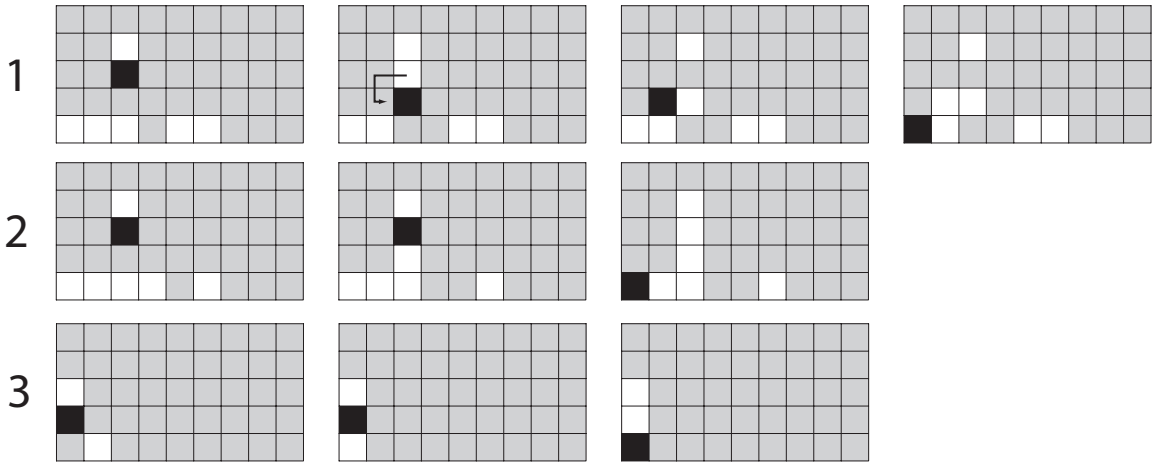

Figure 10: Three "clearing conditions" for the heuristic, with their resulting clearing sequences of moves. Clearing Condition 3 applies only when there are two escorts.

"aisle" created by adjacent escorts (see Figure 9).

Items in Region B are retrieved in a more complicated way. We move the escort closest to the item (which is in location $(\min\{i, e\}, 1)$, for an item in location (i, j)) and "puzzle" it down- and leftward, using 3-moves in the normal way and 5-moves if necessary until the configuration meets one of four "Clearing Conditions" (see Figure 10). Clearing Condition 0 (not shown) is any configuration in which there are no interfering items and the requested item can be moved directly to the I/O point. Clearing Condition 1 is met if the item is in location $(i', 3)$, escorts occupy locations $(i', 4)$ and $(i', 1)$, and $(i' + 1, 1)$ is occupied by an item. Clearing Condition 2 is met if the item is in location $(i', 3)$ and escorts occupy $(i', 4)$, $(i', 1)$ and $(i' + 1, 1)$. Clearing Condition 3, which applies only when $e = 2$, is exactly as shown in the Figure.

Items in Region C move as if they were in a single-escort system, with the rightmost

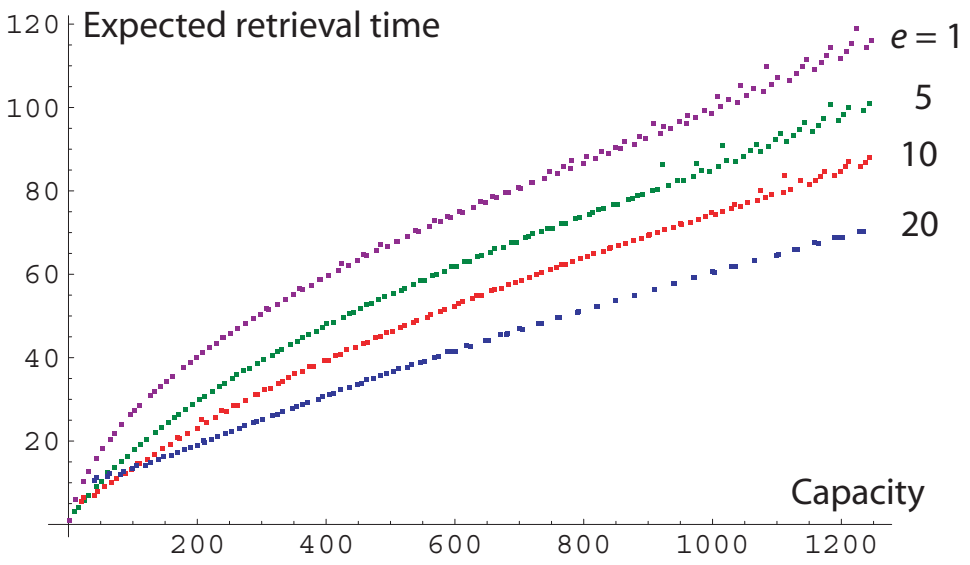

Figure 11: Expected retrieval times for configurations with 1, 5, 10, and 20 escorts.

escort as the only escort, except that the item is puzzled to the aisle and then moved to the I/O point. A formal description of the algorithm for all regions is in Appendix A.

We compared the heuristic to the dynamic program for the 249 cells having items in the 6 configurations in Figure 7, and in every case the heuristic produces an optimal solution. This at least suggests that the heuristic produces good solutions, and we believe its use is justified to develop insights into how puzzle-based systems compare with aisle-based systems.

4.3 Results

Figure 11 shows the expected retrieval time for configurations with $e = 1, 5, 10$, and 20 escorts. The smallest configuration is 4×4, the largest about 35×35. The figure shows only configurations near the "efficient frontier." Each plot shows the same "square root increase" in expected retrieval time as storage capacity increases, and adding escorts reduces expected retrieval time, but at a marginally decreasing rate. For capacity less than about 100 items it is best to have fewer escorts, because a system with, say, 20 escorts is so long that retrieval distances outweigh having fewer interfering items.

Figure 12 shows the same data but with respect to density instead of storage capacity. For

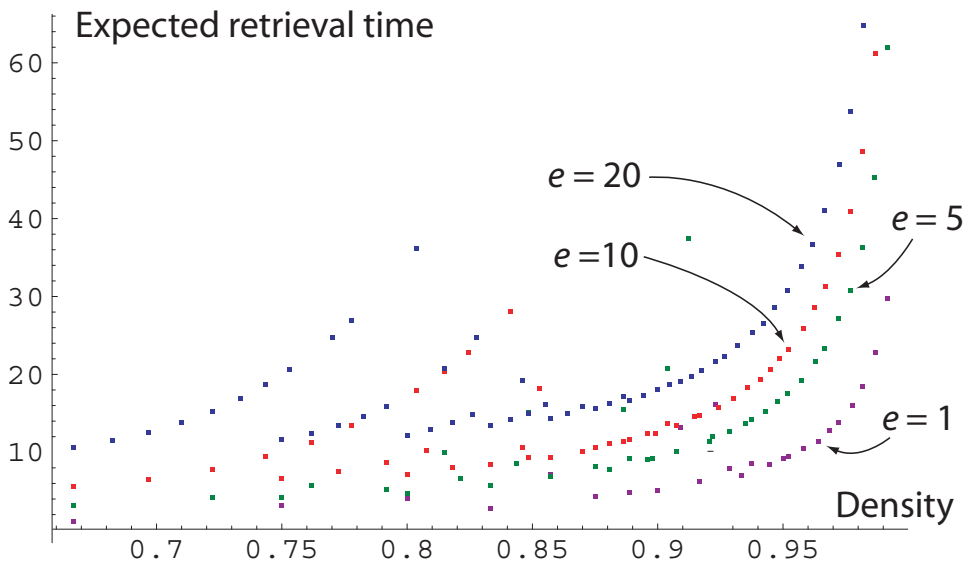

Figure 12: Expected retrieval times as a function of density, for configurations with 1, 5, 10, and 20 escorts.

a given density, configurations with *fewer* escorts tend to have the lowest expected retrieval times, but this is because these configurations have relatively low storage capacities. For example, among four configurations having density of approximately 0.92, the 20-escort configuration has capacity 240, the 10-escort configuration 122, 5-escort 58, and 1-escort 13. The single-escort system has the lowest expected retrieval time, but also the lowest storage capacity by far.

5 Comparison with aisle-based systems

We have established retrieval time results for puzzle-based systems for any number of escorts $e \leq m$, where m is the length of the grid. For larger values of e, we can imagine that configurations take on traditional aisle structures, as described in Gue [5].

Aisle-based systems have the advantage that, once any interfering items are cleared, travel to the I/O point is quick; but they have the disadvantage of lower density and therefore larger distances to travel. Puzzle-based systems have the advantage of very compact design, which

leads to short distances, but there are interfering items at almost every step. Which protocol is better?

We conducted an experiment with many configurations of aisle- and puzzle-based systems to see if one protocol is dominant, or if there are situations in which one protocol might be preferred to the other. Our experiment is based on the following assumptions:

1. All movement must be within the grid; that is, we do not allow an interfering item to be temporarily stored outside the grid while retrieving a desired item.

2. The time to move an item from one location to an adjacent open location is the same for both types of systems. This assumption favors puzzle-based systems, which likely would require more sophisticated mechanical designs and slower travel in the space.

3. There is no pickup-deposit time when moving an item; that is, there is no fixed cost to moving a new item. This assumption also favors puzzle-based designs because in such systems a new item moves at almost every step.

4. For aisle-based designs, there is a single I/O point located in the center of the bottom of the configuration. Given a configuration with vertical picking aisles [5], placing the I/O point below the bottom cross-aisle was by far the best choice in tests of several candidate locations.

5. For aisle-based designs, items are not allowed to "puzzle." For example, interfering items at the end of an aisle must be moved completely out of that aisle to allow a requested item to pass. It turns out that this assumption does not affect expected retrieval times by much.

Figure 13 shows expected retrieval time results for aisle-based systems with lane depths $k = 1, 2, 5$, and 7. Grid sizes vary between 4×4 and 40×40. We observe the same "square root increase" in retrieval times as we did in puzzle-based systems, and configurations with greater lane depths *generally* have higher retrieval times. Note the exception for low levels of storage capacity: designs with $k = 1$ are best up to capacity of about 600, but greater

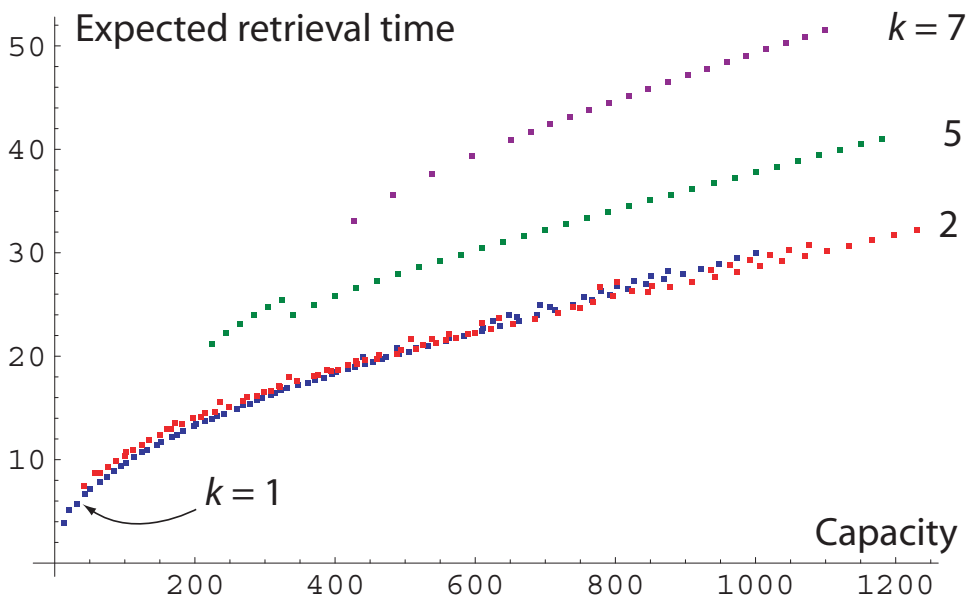

Figure 13: Expected retrieval times for aisle-based systems. The parameter k indicates the maximum lane depth for items in the configuration.

than this capacity, $k = 2$ designs are best because lower expected travel distances outweigh the need to move interfering items in a $k = 2$ design. In fact, it is best to store 600 items in a $k = 2$ aisle-based configuration—the answer to the question we pose in Section 1 (see Figure 2, top row, second design).

Figure 14 shows how aisle-based systems compare with puzzle-based systems. For all capacities, aisle-based systems yield lower expected retrieval times. Considered together, the plots show something of a continuum of designs. Single- and double-deep aisle-based designs have the lowest expected retrieval times, followed by those with increasing lane depths. For lane depths in excess of about 10 (not shown in the figure), puzzle-based designs with many escorts are a better choice. A single-escort puzzle system has the worst expected retrieval time.

If storage density is the main concern, the results are reversed: single-escort puzzle systems provide the lowest expected retrieval times for a given storage density, followed by configurations with increasing numbers of escorts, and then by aisle-based systems with deep

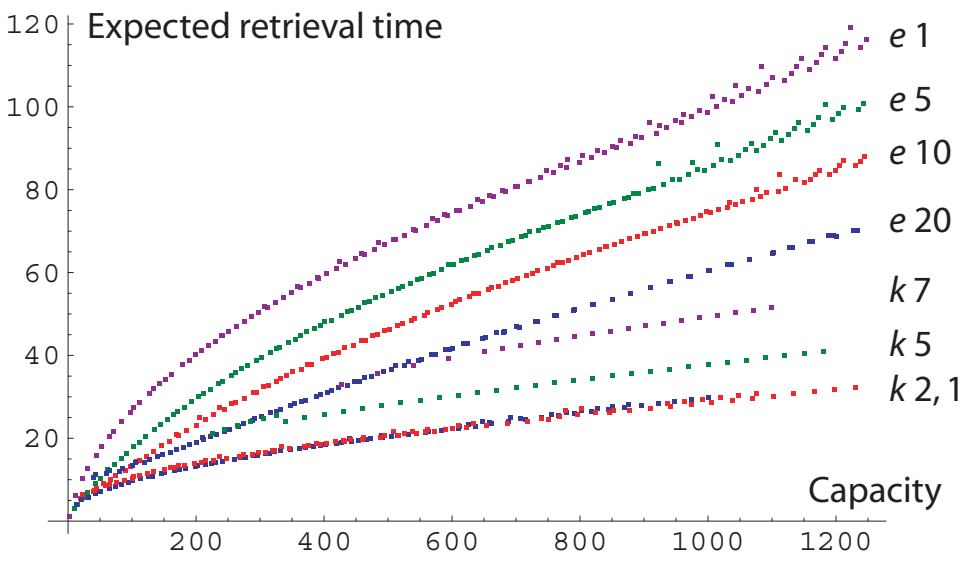

Figure 14: Expected retrieval times for puzzle- and aisle-based systems. Data for single- and double-deep aisle-based systems overlap: single-deep ($k = 1$) is slightly better for capacities up to about 600; double-deep is better for greater capacities.

lanes (see Figure 15). Single-deep aisle-based systems have the highest expected retrieval times for a given storage density.

A perhaps more realistic setting would acknowledge that each movement of an item involves a fixed time to engage the item; that is, there is a fixed charge associated with changing the moving item. To test the implications, we repeated our experiment with a fixed *pick time* associated with the movement of a new item. For simplicity, we chose the pick time equal to the time to travel from one cell to an adjacent cell. Figure 16 illustrates that in the presence of a pick time, puzzle-based systems become even less competitive, because so many items must be engaged to effect a retrieval. If the time to travel between locations were higher for a puzzle-based system—as we might expect for mechanical reasons—the difference would be even greater. As before, if density is the main concern, puzzle-based systems are preferred, although not quite as strongly.

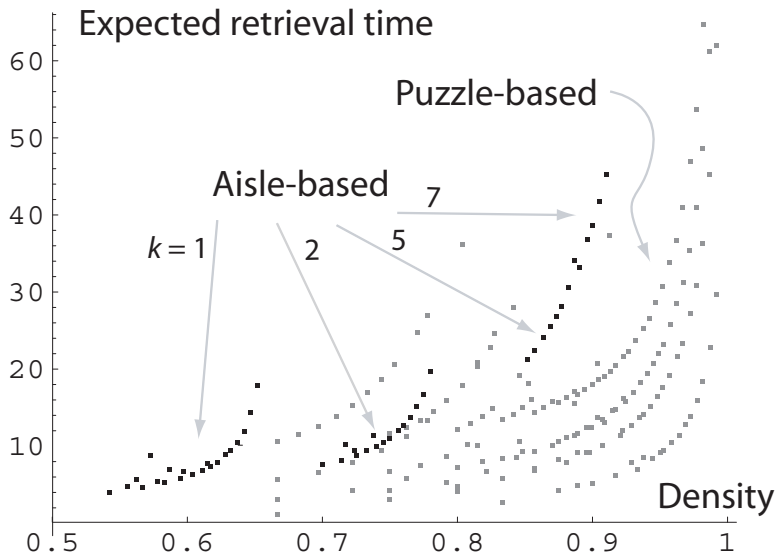

Figure 15: Expected retrieval times with respect to storage density for puzzle-based (gray dots) and aisle-based (black dots) systems.

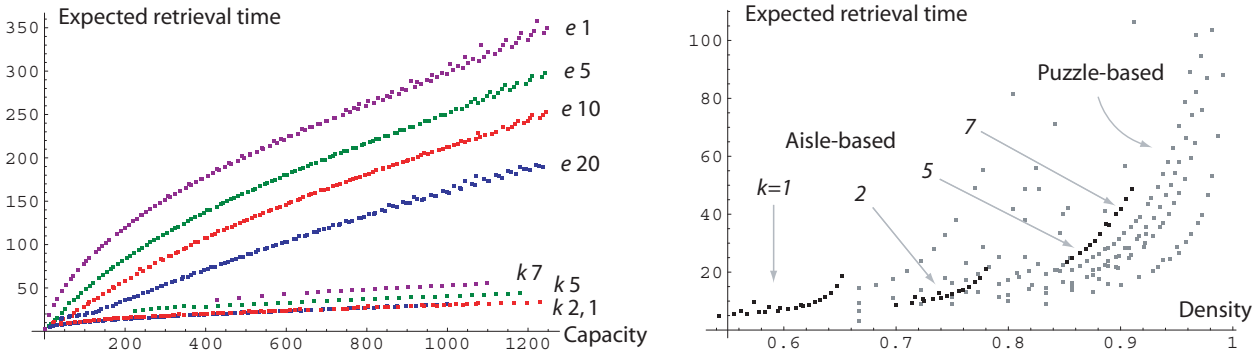

Figure 16: Results for puzzle- and aisle-based systems when there is a fixed time to engage an item for movement. For clarity, the density plot (on right) shows only a subset of the complete data set.

6 Conclusions

The slide-puzzle architecture presents an opportunity to design storage spaces with the highest possible storage density. Until now, nothing has been known about the operational performance of these systems.

We have developed an optimal method of retrieving an item from puzzle-based systems with a single escort. For systems with multiple escorts, we developed a heuristic that produces optimal solutions for a large set of test problems. Our results confirm the intuition that having more escorts in a puzzle system generally reduces expected retrieval time. This may not be the case for smaller systems, where many escorts lined up at the I/O point necessitates a long, thin configuration with long travel distances.

We also find that systems with the highest density have the longest expected retrieval times, and systems with lower density have shorter expected retrieval times, for a given capacity. A notable exception occurs for moderately large aisle-based systems, where double-deep storage can be preferred to single-deep storage. We strongly suspect that for extremely large systems, a triple-deep configuration would be preferred to double-deep, and so on.

If the design goal is to store a certain number of items with the lowest possible expected retrieval time, we find that puzzle-based systems are not competitive. The cost of constantly moving interfering items overwhelms the advantage of having a more compact system. In general, puzzle-based systems should only be considered when densities in excess of 90 percent are required, and then the system should be designed with as many escorts as possible.

To illustrate the potential use of our models, consider the NAVSTORS system illustrated in Figure 1, which is a 6×6 single-escort puzzle system. This system has capacity 35, density $35/36 = 97.2\%$, and expected retrieval time 15.4 moves (35, 97.2%, 15.4). Adding an additional escort by removing one of the storage items results in a two-escort system having capacity 34, density 94.4%, and expected retrieval time 12.1 (34, 94.4%, 12.1). Other alternatives are a three-escort system (33, 91.7%, 9.6) or four-escort system (32, 88.9%, 7.9). If more space could be found, the required 35 items could be stored in a 6×7 grid with

7 escorts; this design has lower density (83.3%), but also a lower expected retrieval time (6.9). This suggests that finding enough space for the equivalent of seven more items could reduce expected retrieval time by more than one half. Which system is best depends on the objectives and constraints faced by the designer.

Acknowledgements

The authors thank the Office of Naval Research for its generous support of this research.

References

[1] Gary W. Flake and Eric B. Baum. *Rush Hour* is PSPACE-complete, or "Why you should generously tip parking lot attendants". *Theoretical Computer Science*, 270(1–2): 895–911, 2002.

[2] Martin Gardner. *The Mathematical Puzzles of Sam Loyd*. Dover, 1959.

[3] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Company, New York, 1979.

[4] Ralph U. Gasser. *Harnessing Computational Resources for Efficient Exhaustive Search*. PhD thesis, Swiss Federal Institute of Technology, Zürich, 1995.

[5] Kevin R. Gue. Very High Density Storage Systems. *IIE Transactions*, 38:93–104, 2006.

[6] Richard Hayes. The Sam Loyd Puzzle and Assorted Problems: An Investigation of Updateable Arrays in Functional Programming. Master's thesis, University of Dublin, 2000.

[7] Robert A. Hearn and Erik D. Demaine. PSPACE-Completeness of Sliding-Block Puzzles and Other Problems through the Nondeterministic Constraint Logic Model of Computation. *Theoretical Computer Science*, 343(1–2):72–96, 2005.

[8] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the Complexity of Motion Planning for Multiple Independent Objects: PSPACE-Hardness of the 'Warehouseman's Problem'. *International Journal of Robotics Research*, 4(3):76–88, 1984.

[9] W. W. Johnson and W. E. Story. Notes on the "15" puzzle. *American Journal of Mathematics*, 2(4):397–404, 1879. Publication of this issue was delayed until April, 1880.

[10] Filip R. W. Karlemo. On sliding block puzzles. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 34:97–107, 2000.

[11] Ian Parberry. A real-time algorithm for the $(n^2 - 1)$-puzzle. *Information Processing Letters*, 56:23–28, 1995.

[12] D. Ratner and M. Warmuth. Finding a shortest solution for the $(n \times n)$-extension of the 15-puzzle is intractable. *Journal of Symbolic Computation*, 10:111–137, 1990.

[13] Alexander Reinefeld. Complete solution of the eight-puzzle and the benefit of node ordering in IDA. In *IJCAI*, pages 248–253, 1993. URL `citeseer.ist.psu.edu/reinefeld93complete.html`.

[14] Jerry Slocum and Dic Sonneveld. *The 15 Puzzle*. The Slocum Puzzle Foundation, 257 South Palm Drive, Beverly Hills, CA, 2006.

[15] P. G. Tait. Note on the theory of the "15 puzzle". *Proceedings of the Royal Society of Edinburgh*, 10:664–665, 1880.

[16] John Tromp and Rudi Cilibrasi. Limits of rush hour logic complexity, 2005. URL `www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:cs/0502068`.

A Multi-escort algorithm

Algorithm 2 MULTI-ESCORT

if the requested item is in Region A **then**
  Move interfering items down and to the right.
  Move the item directly to the I/O point.
end if
if the requested item is in Region B **then**
  Move the item down using the escort in $(\min\{i, e\}, 1)$.
  while the requested item is in Region B **do**
    if $i' \neq 1$ **then**
      Perform a 3-move
    else
      if a 3-move down is possible **then**
        Perform a 3-move
      else
        Perform a 5-move
      end if
    end if
  end while
  while a clearing condition is not met **do**
    if a 3-move left or down is possible **then**
      Perform a 3-move
    else
      Perform a 5-move
    end if
  end while
  Perform the clearing move (see Figure 10)
end if
if the requested item is in Region C **then**
  Move the item left using the escort in $(e, 1)$.
  Repeat 3-moves and then 5-moves until there is no interfering item.
  Move item directly to the I/O point.
end if