GridStore: A Puzzle-Based Storage System with Decentralized Control

Kevin R. Gue, Kai Furmans, Zäzilia Seibold, Onur Uludağ

***Abstract*—We describe a high-density storage system for physical goods in which identical conveyor modules can be plugged together to store and retrieve unit-loads or small containers. Material movement conforms to the "puzzle architecture" found in popular board games such as the 15-puzzle and Rush Hour. Control of the system is decentralized, meaning that each module contains identical operating logic that directs its behavior based on local conditions and message passing. We prove the system deadlock-free and show its performance under a wide variety of operating configurations.**

***Note to Practitioners*—One of the obstacles to widespread adoption of automated material handling systems is inflexibility: once a system is designed and installed, making changes to its configuration is typically very difficult. The GridStore system overcomes this obstacle for high-density storage systems by implementing decentralized control, meaning that each conveyor module in a GridStore system has its own, independent controller. One implication of decentralized control is that the system can be expanded or reconfigured rapidly, and without rewiring or reprogramming. Applications of this system could range from small, desktop systems to large, container-handling systems.**

I. Introduction

STORAGE systems for physical goods are a necessary and important part of most manufacturing and distribution enterprises. In manufacturing, an automated storage and retrieval system (AS/RS) can be a space-effective way to store components, sub-assemblies, or work-in-process inventory. In distribution centers, AS/RSs can be the centerpiece of fully-automated unit-load delivery systems, or they can be used to replenish case or broken-case order picking areas. Tote-based systems are often used to bring small items to an order picking workstation, where workers assemble orders. Such systems have become essential parts of modern logistics systems, where replenishment quantities are typically small and product velocity is high.

Despite their potential to reduce the costs of material handling labor, automated storage systems face significant obstacles to adoption. One of the most significant is the perception that such systems are inflexible; that is, that they are incapable of accommodating changes in storage capacity, throughput capacity, or types of material stored and handled. For example, retail distribution is highly seasonal, so companies are reluctant to buy sufficient automation to handle products during the busy season because utilization of the system would be so low during off seasons. Third-party logistics providers (3PLs) also struggle to justify automation, because their contracts are rarely long enough to ensure payback for such a large capital expenditure.

For our purposes, flexibility in material handling systems embodies two traits, *scalability* and *reconfigurability*. For example, a traditional, aisle-based AS/RS is somewhat scalable because one can add or remove racks, aisles, and cranes, but it is difficult and expensive to reconfigure because adding an additional crane or aisle involves significant changes to mechanical and electrical systems. Another contributor to inflexibility is centralized control, in which making a simple change to the system requires changes to control logic and software. For changes to most automated material handling systems, there is a need to employ the services of technicians from multiple supporting firms.

Furmans et al. [1] proffer an alternative approach called *plug-and-work* material handling, which is based on conveyor modules that can be plugged together to form networks or, as we describe here, high density storage systems. Changing the configuration requires only that modules be unplugged, rearranged, and plugged together again. Modules self-discover the new network through a message passing scheme [2]. A primary feature of plug-and-work systems is decentralized control. Each module in a plug-and-work system contains the same control logic, and conveyance by a module at each time step is based solely on local conditions and on the results of message passing between modules.

In this paper, we describe a high-density storage system for physical goods called GridStore, which features a scalable, modular structure and decentralized control. We show that the underlying control rules for GridStore guarantee deadlock-free operation, and we describe the performance of the system for several sizes and configurations.

II. Related literature

Research in automated storage and retrieval systems has a long history. The seminal paper is [3], which describes throughput models for classic crane-in-aisle systems. Dozens of papers have been written on design questions for crane-in-aisle AS/RSs. See [4] for a survey of this work. A number of studies have considered high-density, "compact" AS/RSs [5]–[7]. These systems provide flow rack versions of crane-based

Kevin Gue is the Tim Cook Associate Professor of Industrial & Systems Engineering at Auburn University, Auburn AL, 36849 USA, e-mail: kevin.gue@auburn.edu.

Kai Furmans is Professor of the Institute for Material Handling and Logistics, Karlsruhe Institute of Technology, Germany, email: kai.furmans@kit.edu.

Onur Uludağ is pursuing his Ph.D. in Industrial Engineering at Auburn University, email: ozu0001@auburn.edu.

Zäzilia Seibold is pursuing her Ph.D. in Mechanical Engineering at Karlsruhe Institute of Technology, Germany, email: zaezilia.seibold@kit.edu.

Manuscript submitted 18 December, 2012.

systems with replenishment from the same side. De Koster et al. [5] address the sizing problem of a compact AS/RS. Yu and de Koster [6] studied storage assignment with a class-based policy.

Some recent work has addressed the design and operation of a new class of systems called Autonomous Vehicle Storage and Retrieval Systems (AVS/RSs) [8]–[12]. Instead of aisle-captive cranes, AVS/RSs employ a number of autonomous vehicles for the storage and retrieval of items. The authors have pursued both simulation and queueing-theoretic models of these systems.

Gue [13] investigated the question of maximum storage density in a grid, given a constraint on lane depth for every item. The author addressed only the arrangement of items in a grid, and not their movement. Gue and Kim [14] considered a storage system based on the 15-puzzle. They assume that one or more open cells "escort" a requested item to a single pickup and deposit point, and that only one item moves at a time. The sole exception to this policy showed the optimal algorithm for retrieving a single item using a single open cell when multiple items can move at the same time. Alfieri et al. [15] addressed a puzzle-based storage system, but with a limited number of automated vehicles. Taylor and Gue [16] investigated several design parameters in puzzle-based systems with multiple open cells, including the best placement for open cells, demand patterns, and storage profiles. It is important to note that the methods in [14] and in [16] assumed "puzzle movement," in which open cells act independently and only one at a time. Moreover, all systems in these papers assumed centralized control and a single pickup and deposit point. The decentralized control scheme we describe below features any number of open cells moving *simultaneously* to achieve coordinated retrieval of requested items and storage of replenishing items.

The computer science literature has addressed problems related to ours. The most closely related is the "warehouseman's problem," which was first investigated in [17]. The classic warehouseman's problem considers a rectangle containing smaller rectangles of different sizes. The objective is to achieve a final configuration from a specified initial one. The authors showed that the general problem is PSPACE-hard, a stronger condition than NP-hardness. Sharma and Aloimonos [18] showed that the warehouseman's problem is tractable under some restrictions. A closely related problem is the board game Rush Hour, in which 2×1 cars and 3×1 trucks are placed at different orientations on a grid to simulate traffic gridlock. The objective is to move vehicles forward and backward to allow the passage of a single car from a blocked position to an exit on the perimeter of the board. Flake and Baum [19] showed that the feasibility question (can the car escape at all?) is PSPACE-complete. Hearn and Demaine [20] showed that Rush Hour with cars only is PSPACE-complete. Demaine and Hearn [21] describe a method for investigating the complexity of puzzles such as Rush Hour. Depuy and Taylor [22] present an integer program to solve the Rush Hour puzzle in the minimum number of moves. For computational and other reasons, integer programming is inappropriate for our very large-scale, real-time control problem.

An important system related to GRIDSTORE is the Flexconveyor [2], a modular conveyor system that features decentralized control to move totes among unit-sized conveyor modules. Mayer and Furmans show that their control algorithm is deadlock-free for any number of totes in the system, as long as at least one module is unoccupied.

III. THE GRIDSTORE SYSTEM

Before presenting the details of decentralized control and system behavior, we give a high-level view of the GRIDSTORE system.

Consider a rectangular grid of square conveyor modules, each of which is capable of conveying in the four cardinal directions: North, South, East, and West. This capability is standard in the conveyor industry, and is made possible by, for example, roller conveyors in one dimension (N, S) and pop-up belts in the other (E, W). Modules communicate only with neighbor modules to which they are connected in the grid. They are also capable of communicating with the items they contain (via RFID, for example). An alternative way of knowing about the items is via messages passed from neighbor modules along with the items.

The southernmost side of the grid is a retrieval conveyor, which we assume conveys away from the grid at infinite speed. That is, once a requested item reaches the retrieval conveyor, it is removed from the system and will not interfere with the movement of other items. Although this assumption sounds unrealistic, all that is needed in practice is a retrieval system that conveys away from the grid at a speed greater than the speed of movements within the grid. The northernmost boundary is a replenishment conveyor onto which items appear, to be stored in their *home rows*. The system entertains requests, via RFID for example, for particular items in the grid. Once an item has been requested, its host conveyor module attempts to move it southward.

Figure 1 illustrates three items moving to the retrieval conveyor. At each time step, interfering items move out of the way of requested items such that, if possible, the requested items move uninterrupted to the retrieval conveyor in a sort of "virtual aisle." As long as the module immediately in front of a moving item is clear, the item is free to continue its movement. The GRIDSTORE control algorithm we describe below both controls movement of requested items to the retrieval conveyor and movement of replenishing items to their home rows.

IV. DECENTRALIZED CONTROL

It is tempting to imagine the retrieval of requested items and storing of replenishing items as a large combinatorial optimization problem. We have chosen another path for several reasons. First, any reasonably sized system would have an enormous number of integer variables almost certain to lead to intractability. Second, the system must respond to random requests, and therefore any optimization model would have to be solved thousands of times per day in real-time, a feat hard to imagine. Third, reliability of the system would be inextricably tied to a large-scale optimization model, for which such models are ill-suited. Fourth, centralized control makes

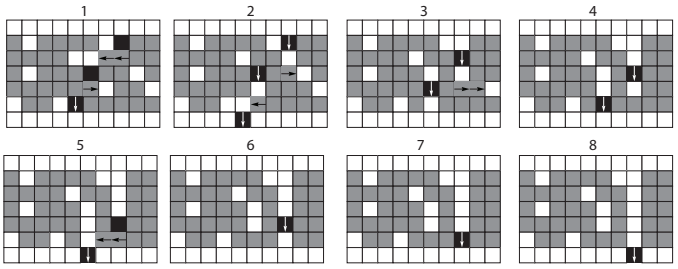

Fig. 1. Three items (in black) moving southward to a retrieval conveyor. Items that have not been requested (in gray) move out of the way to clear a path.

the system difficult to reconfigure, and ease of reconfiguration is one of the motivations behind the system.

Instead, coordinated movement within the GRIDSTORE system is accomplished by decentralized control; at each time step, each module assesses its current state and the states of its neighbors, and then executes the same set of instructions.

The system operates according to an Assess-Negotiate-Convey cycle. In the Assess phase, each module takes a *state* according to the presence or absence of an item. The three possible states are:

- SEEKING: The module is occupied with a "moving" (requested or replenishing) item.
- OCCUPIED: The module is occupied with a stored (not moving) item.
- EMPTY: The module has no item.

To determine whether or not to convey, and in which direction to convey, modules engage in two electronic *negotiations*, which are accomplished by means of message passing. Modules take on *negotiating positions*, which change during the negotiation. In the convey phase, the modules move the items according to the results of the negotiation.

Negotiation is done with message passing: Once a module receives a message, it changes its negotiating position and/or sends a message to its neighbor according to the protocol. Negotiating positions simply reflect having received or sent a message from a particular side. Thus, there is an equivalence between negotiating positions and message types.

A. North-South Negotiation

The North-South negotiation has two distinct phases. In the first, replenishing items look for opportunities to accomplish a "tandem replenishment." In the second, requested items look for opportunities for tandem movement.

Replenishment is the process of moving items from the replenishment row to their home rows. Once in its home row, a replenishing item becomes an ordinary stored item. Tandem replenishment is based on the insight that it does not matter which stored items are in a row, only that the number of items with that assigned home row is constant (more on this requirement below). Items exchange home rows to accomplish more efficient replenishment—instead of one item moving through several rows in several cycles, several items move in tandem in one cycle, after exchanging home rows. Figure 2 shows two examples in which the replenishment is accelerated by tandem movement. Both examples show a storage column before the exchange of home rows, after the exchange of home rows, and after the southward movement of the items by one row. In the left example, the number of replenishing items is the same at the beginning and end of the cycle; whereas, in the right example, the number of replenishing items increases by one. Even though an increased number of replenishing items hinders East-West movement slightly, the exchange of home rows also enables the movement of the requested item in this case. Details of the message passing scheme for tandem replenishment are in the Appendix.

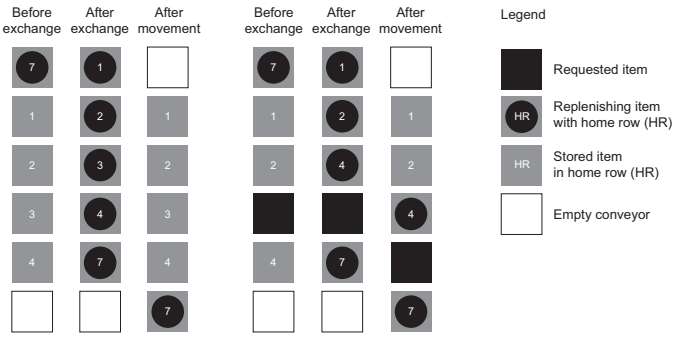

Fig. 2. Examples of tandem replenishment by exchange of home rows.

The goal of the second phase of North-South negotiation is to identify *convoys* of moving items, which will move together in the conveyance phase. For example, if the item south of a requested item is moving southward, then the requested item should follow it in the same time step. Without this step, would-be convoys separate themselves with empty modules between each moving item, and it would take more cycles to accomplish retrievals.

At the beginning of North-South negotiation, the modules are in a default negotiation position (no position). Message types are:

- ↓R SEEKING modules initiate a **Request** message to the south in order to check for an EMPTY module.
- ↑C An EMPTY module responds with a **Commit** message that it will receive the moving item.

The North-South negotiation is defined by a communication protocol with 6 rules (Figure 3). Each section in the table represents a rule for a certain situation. The left side of each rule shows the module state and its negotiating position upon receiving a certain message type. The right side shows the new negotiating position and the messages sent by the module. The left most rule, for example, shows an EMPTY module receiving a **Request** message. It responds with a **Commit** message to the North and takes the corresponding negotiating position. It also forwards the **Request** message to the South.

The table omits infeasible combinations of states and received messages, as well as conditions that do not result in a message being passed or a negotiating position being changed. The first row of the table shows the initiation of the **Request**. The second and third rows define the rules for the receipt

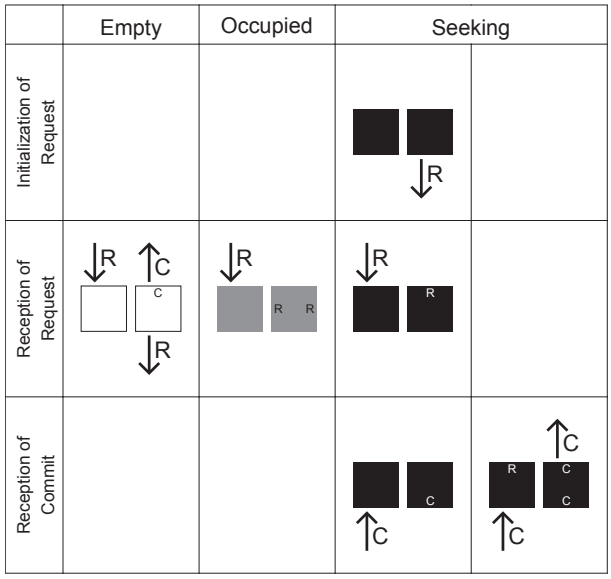

Fig. 3. Communication protocol for North-South negotiation. Negotiating positions are represented by small letters on the side of the conveyor state symbol.

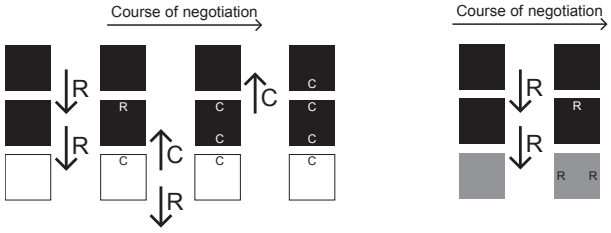

Fig. 4. Two examples for North-South negotiation: identification of convoy (left side) and necessity to clear the path (right side)

of a **Request** from the North and a **Commit** from the South respectively.

Figure 4 shows two examples of North-South negotiation. In the left example, two SEEKING conveyors send **Request** messages to their southern neighbors. The EMPTY module responds to the **Request** with a **Commit** message. Because the middle module has received a **Request** from its northern neighbor, it forwards the **Commit** message to the North. All three modules turn their negotiating position to **Commit** so that both items move in tandem.

The example on the right shows an OCCUPIED module blocking the way of two requested items. To move its item out of the way by finding an EMPTY module, the OCCUPIED must engage in East-West negotiation, to which we now turn.

B. East-West Negotiation

The outcome of the North-South negotiation serves as input to the East-West negotiation. The goal of this phase is to move stored items out of the way of moving items. The message passing protocol follows a Request-Willing-Commit logic. Modules that need to create a "virtual aisle" look East and West for EMPTY modules to accommodate.

The negotiation protocol aims to move stored items in convoys east- or westward. Message types for East-West negotiation are:

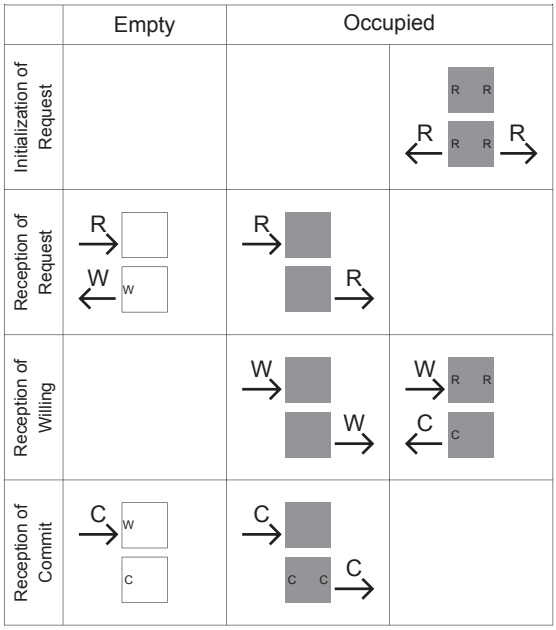

Fig. 5. Communication protocol for East-West negotiation (only shown for messages arriving from West)

R →	Modules trying to become empty send **Request** messages to the East and West in order to find an EMPTY module.
W →	An EMPTY module responds to the first arriving **Request** that it is **Willing** to receive movement from that side.
C →	The requesting module responds to the first arriving **Willing** that it will **Commit** in that direction.

The East-West negotiation is also defined by a communication protocol (Figure 5). Again, each section in the table represents a rule for a certain situation. The upper graphic of each rule shows the module state and its negotiating position while receiving a certain message type. The lower graphic shows the new negotiating position and the messages sent by the module. As before, infeasible combinations of states and received messages are omitted. Rules for messages from the East are analogous.

East-West negotiation is based on the following insight: A module that tries to become empty sends a message to its two neighbors "asking" if it can convey in that direction. The neighbor modules may themselves be OCCUPIED, so they may have to ask *their* neighbors if they can convey, and so on. Eventually, and inevitably, one of two things happens: an EMPTY module receives the message and replies with a **Willing** message, or a module that cannot convey (because it is in state SEEKING, for example) receives and discards the message. The **Willing** message is sent back to the original requestor which replies with a **Commit** message.

In order to reduce movement in the grid, an EMPTY module should, in the presence of multiple requests, reply to the **Request** generated by its nearest requestor. Similarly, a module attempting to become empty should reply to the **Willing** message generated by the nearest EMPTY module. Even if messages arrive from both sides at almost the same

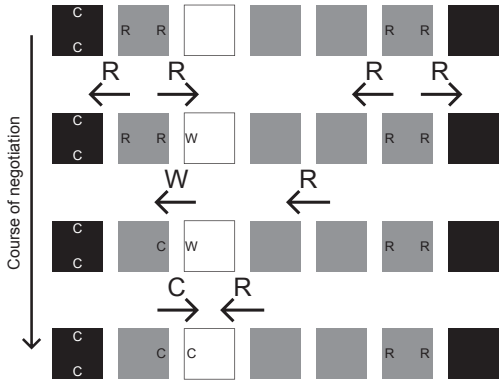

Fig. 6. Example for East-West negotiation

time (because separated by the same number of modules), in real hardware there would always be a small time separation between the messages. In the simulation, as in real hardware, messages cannot be generated at exactly the same time, and in the case that a module sends "simultaneous" messages to the east and west, we randomly choose which direction is sent first in order to avoid artificially-induced imbalances.

Figure 6 shows an example with two modules trying to become empty and, thus, initiating **Request** messages to the East and West. The EMPTY module replies to the first arriving **Request** (from the West) with a **Willing**. The requesting module responds with a **Commit** and both modules take negotiating position **Commit**. The **Request** from the East is ignored because the EMPTY module has already accepted the **Request** from the West.

After East-West negotiation, those modules that have committed will convey. After conveying (or staying idle), modules reset their negotiation positions to default, and the Assess-Negotiate-Convey cycle is repeated.

V. BEHAVIOR

The Assess-Negotiate-Convey cycle controls internal movement of items in the grid, but the storage system as a whole also requires a control scheme to interact with surrounding material handling systems. Without an external control scheme, it is possible to deadlock the system: simply introduce items as replenishments on the top row without requesting any items to depart. Eventually the system fills entirely and is deadlocked.

To prevent such pathological behavior, we require the GRIDSTORE system to operate in a constant work-in-process (CONWIP) mode, where "work-in-process" refers to the number of active requests in the system. The departure of a requested item in a GRIDSTORE system initiates two actions: (1) a replenishing item enters from a randomly selected column in the top row, whose home row is the home row of the item just departed, and (2) a new request is initiated. These two conditions ensure a balanced system in the long run, and as we show below, they ensure the system is deadlock-free. To initialize the simulation, we release the required level of work-in-process at once; therefore, there is no warm-up period during which WIP must be accumulated. In the analysis that follows, we assume that the new request is equally likely to be for any unrequested item in the system.

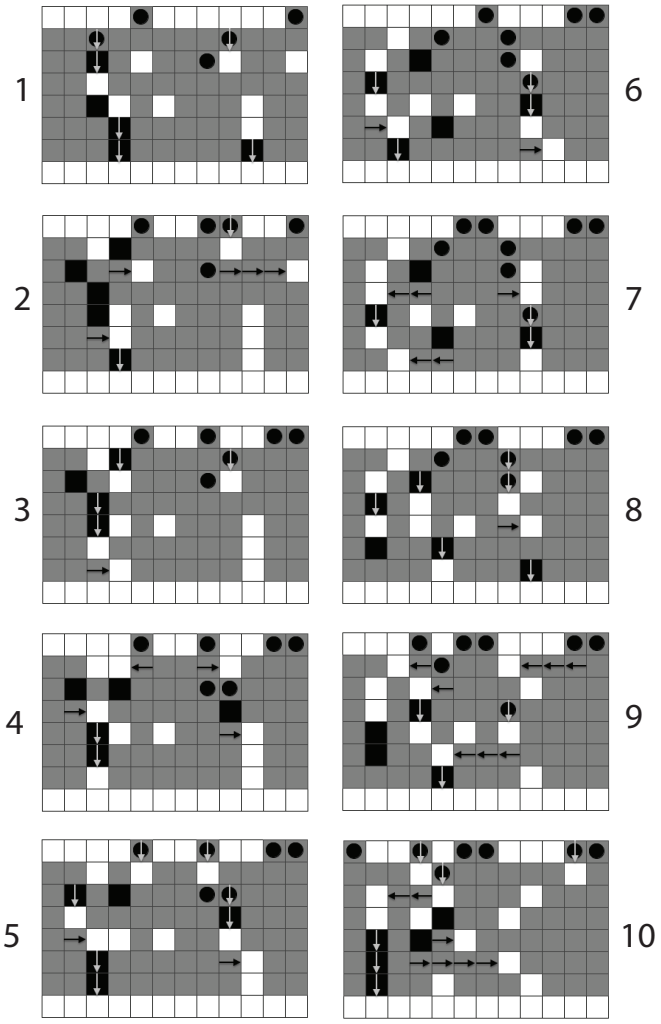

Fig. 7. Several iterations of the GRIDSTORE algorithm. Squares with a round circle represent replenishing items, which turn gray when they reach their home rows. Black squares represent requested items. Animation is available at http://ieeexplore.ieee.org.

Figure 7 shows several iterations in a small example system. Notice in graphic #1 near the top that a replenishing item and a requested item move together in the same iteration. This is an outcome of the North-South negotiation, which establishes convoy-like behavior before the East-West negotiation begins. In graphics #5 and following, as the requested item in column 9 moves downward, each potentially interfering item ahead moves out of the way in the same iteration, thereby allowing the requested item to depart the system unimpeded in a "virtual aisle." Not every requested item experiences uninterrupted travel, of course, but the control rules are designed to effect this sort of behavior. After each departing item reaches the retrieval row, a replenishing item is introduced in a random column on the top row. (Introducing replenishing items into a random column is necessary to avoid a system deadlock, as we show below.) A downloadable animation of GRIDSTORE is available at http://ieeexplore.ieee.org.

An important practical and theoretical question is whether GRIDSTORE is deadlock-free, where a deadlock means that at least one requested or replenishing item *will not* reach its

destination row.

Proposition 1: The GRIDSTORE system is deadlock-free.

Proof: Because GRIDSTORE moves requested items only southward, it is sufficient to show that a requested or replenishing item will always move southward to its destination eventually. Assume to the contrary that there are requested or replenishing items in the grid unable to move southward. Consider the southernmost deadlocked item. The module south of the deadlocked module must be occupied, either by a moving item or by a stored item. Call this the blocking module. If the blocking module contains a moving item, then that item will eventually move southward; that is, its module cannot be deadlocked, because it is south of the southernmost deadlocked module. Therefore, the deadlocked module will eventually convey downward, a contradiction.

If the blocking module contains a stored item, it will compete for empty modules when they appear by sending request messages to the East and West. The initial setup of the GRIDSTORE system requires $k > 0$ empty modules in each row, which ensures that at all times at least k modules in each row are either EMPTY or SEEKING to move their items southward. Because replenishing items enter a random column, it is impossible for a module always to be occupied with a moving item; at some time it will become empty. Therefore, the blocking module will have an infinite number of chances to secure the services of an empty module. When it does, its item will move out of the way and the deadlocked module will convey its item, a contradiction.

Applying this argument to all remaining deadlocked items completes the proof. ■

VI. PERFORMANCE

Performance of the GRIDSTORE system depends on a number of design and operational parameters. Here, we investigate the effects of the number of active requests (work-in-process, WIP) in the system and the number and distribution of empty modules per row (k). We also ran experiments with respect to the question of aspect ratio of the grid. The results were as expected: Systems with more columns have higher throughput; systems with fewer columns have lower throughput for approximately the same number of items stored.

The experiments below are based on a number of assumptions:

1) The grid contains 144 unique items, stored among 12 rows and 13, 14 or 15 columns depending on the initial number k of empty modules per row.
2) Once requested, an item must make its way to the retrieval row south of the grid, where it immediately disappears. We make this assumption because we are interested in the performance of GRIDSTORE without respect to other material handling systems to which it might be connected.
3) North of the grid is an unlimited number of replenishment rows. Replenishing items introduced upon retrieval of a requested item are randomly assigned to an open module in a replenishment row closest to the grid.

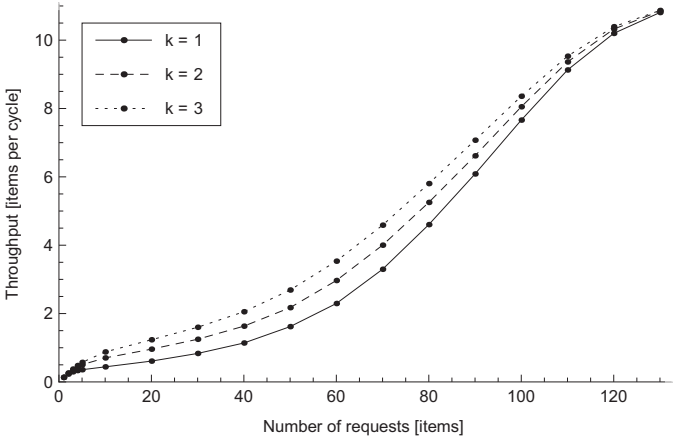

Fig. 8. Throughput diagram for systems with different density.

4) When initiating a new request, the system selects randomly from among items not currently requested. They can already be in their home rows or still be in transit.

The decentralized control rules were imbedded into the necessary number of (software) conveyor objects, which communicate with their neighbors as described above. Each simulation run executed for 5,760 time steps (corresponding to an 8-hour-shift and a 5 second cycle time), with a warm up period of 200 steps. Results in the plots reflect 30 replications of each run. Experiments were run in AnyLogic.

A. Average Throughput and Retrieval Time

For a constant work-in-process system, throughput rate and average retrieval time are related by Little's Law, so we discuss them together. Figure 8 shows the average throughput in items per cycle for systems with different storage densities (a higher value of k implies a lower density). Throughput increases with increasing WIP. The density of the system has the expected effect: Because more empty modules can form more virtual aisles, systems with lower density have higher throughput. As WIP approaches the storage capacity of the grid, almost all items are moving toward the retrieval conveyor and throughput approaches the number of columns, less the number of empty modules per row ($n - k$).

Figure 9 shows how long it takes to retrieve a requested item for different levels of WIP. Average retrieval time is shortest for low WIP, increasing to its maximum at about WIP $= 30$. For very high levels of WIP, items are requested as soon as they enter the system, and retrieval time equals the number of rows because items move toward the retrieval row in tandem without obstruction.

Three main phenomena (see Figure 10) can explain the rise and fall of retrieval time with increasing WIP:

1) *Competition for empty modules:* For very low levels of WIP, empty modules align themselves in a way that items proceed unimpeded toward the retrieval row. Retrieval times are low. As WIP increases, requested items can be blocked by stored items below, which are forced to compete for empty modules making way for other requested or replenishing items.

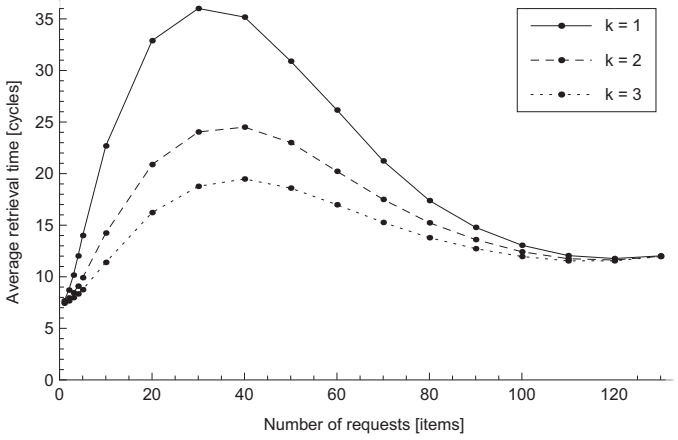

Fig. 9. Average retrieval time diagram for systems with different density.

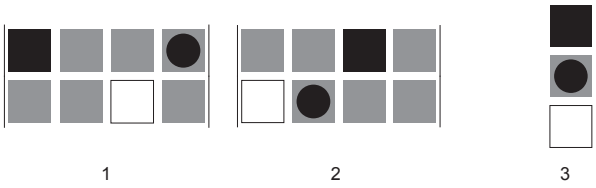

Fig. 10. Three phenomena influencing the performance of GRIDSTORE.

2) *Interference by requested items:* As WIP increases, requested items can be blocked by stored items below because there is another moving item between the stored item and an empty module, which blocks East-West movement and leads to an increase in retrieval time.
3) *Tandem movement southwards:* When WIP is high, items are more likely to move in tandem with other requested or replenishing items. Tandem movement makes more efficient use of empty modules because they need no reposition to facilitate passage of moving items. Tandem movement is largely responsible for the decrease in retrieval time at high levels of WIP.

Figures 8 and 9 also illustrate the expected result that systems with more empty modules have higher throughput and lower expected retrieval time, but with diminishing benefit as k increases. Cases with $k > 3$ (not shown) confirm this observation.

B. Distribution of Retrieval Time

Figure 11 shows the distribution of retrieval time depending on the row where the item was located when requested. Retrieval time increases and is also more distributed for upper rows, as expected. The point of this plot is to show just how much delay can be experienced by requests in upper rows—in some cases more than 150 conveying cycles are required to retrieve an item.

C. Occupancy

Figure 12 shows the average system state for WIP $= 30$ just before the convey phase (therefore, the retrieval row is always empty). Lengths of the bars correspond to the average number of requested, replenishing, and stored items in a specific row; white space indicates the number of empty modules. The vertical line shows the average throughput, which is the same for every row. The average number of

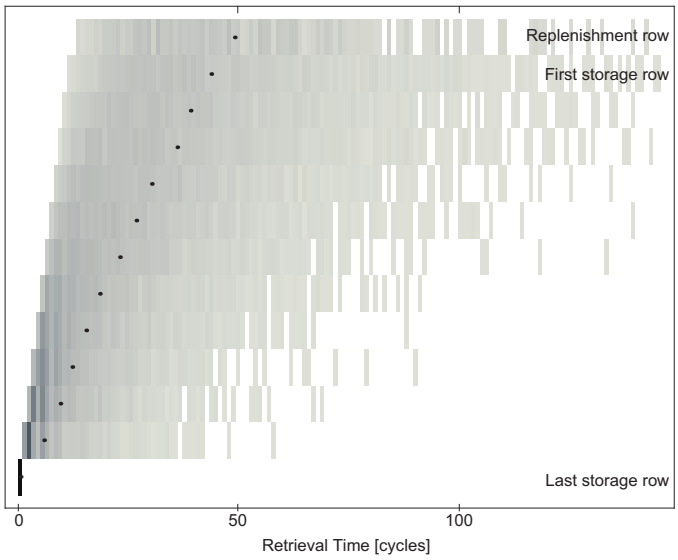

Fig. 11. Distribution of retrieval time depending on row with WIP $= 10$ and $k = 1$ empty module per row. Grayscale values correspond to the likelihood that retrieval time takes a certain number of cycles when retrieving an item from that row. Black dots indicate mean retrieval times.

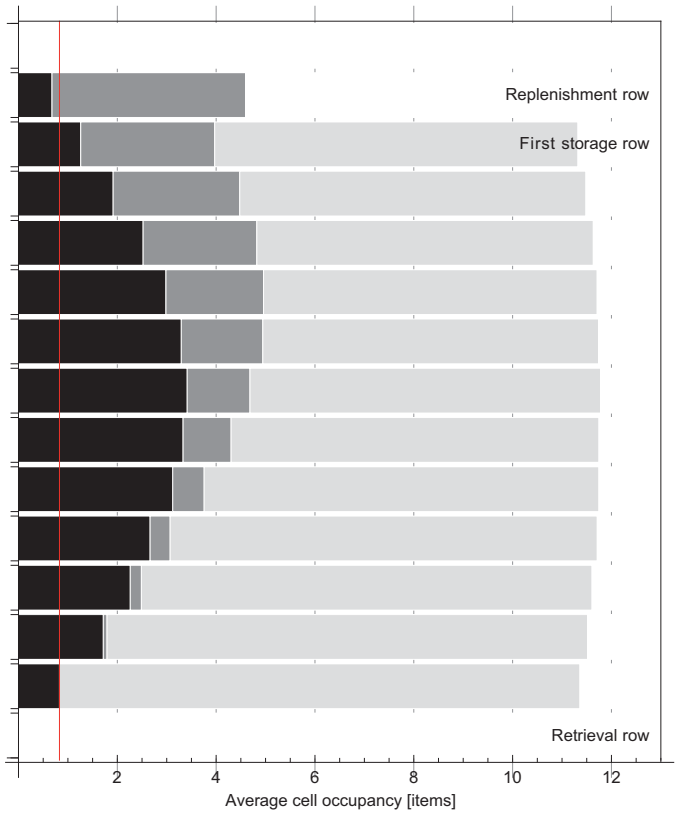

Fig. 12. Module occupancy with WIP $= 30$ and $k = 1$ empty module per row. Black bars represent the average number of requested items per row; dark gray the replenishing items, light gray the stored items, and white space the empty modules)

TABLE I
DISTRIBUTION OF EMPTY MODULES FOR 3 CONFIGURATIONS

Row	Uniform	Decreasing k	Increasing k
1	2	3	1
2	2	3	1
3	2	3	1
4	2	3	1
5	2	2	2
6	2	2	2
7	2	2	2
8	2	2	2
9	2	1	3
10	2	1	3
11	2	1	3
12	2	1	3

requested items in the last row equals the throughput because items are retrieved in one cycle and never have to wait. The additional requested and replenishing items in the second to last row are due to blocking caused by phenomena 1 and 2 described above. The average number of waiting items (sum of black and gray bars) continues to increase until about the middle of the grid, at which point it decreases. This can be explained as follows: Although items requested from northern rows are more likely to be blocked (for the reasons above), there are fewer of them because fewer requested items must pass through northern rows. By contrast, *every* requested item passes through the southernmost row. In addition, tandem replenishment effectively moves replenishing items several rows in one cycle, which decreases the number of waiting replenishing and requested items in the northern part of the grid.

D. Distribution of Empty Modules

So far, we have assumed a uniform distribution of the empty modules among the rows of GRIDSTORE. It would seem that a greater number of empty modules in a row would facilitate higher throughput, but having more in one row means having fewer in another. Also, more empty modules in a row means a lower probability that an item will be requested from that row. Is there a preferred allocation of empty modules?

Here we compare three approaches: a uniform distribution of empty modules, distributions with more empty modules in southern rows ("increasing k"), and distributions with fewer empty modules in southern rows ("decreasing k"). The test grid has 12 rows and 14 columns. All three configurations have same total number empty modules ($k = 2$ on average). The grid includes 144 stored items among the $12 \times 14 = 168$ modules. Table I shows the distribution of empty modules for each configuration.

Figure 14 shows the percent difference between the variable k configurations and the uniform distribution of empty modules. For very low levels of WIP (fewer than 5), there is no competition for empty positions. Therefore, the distribution of empty modules has no effect on performance due to low traffic flow and the absence of blocking (see Figure 14). For low to moderate levels of WIP, the increasing k configuration is best. All configurations have about the same performance for high WIP levels. The relatively low levels of WIP are approximately those in which blocking is most problematic (see Figure 9), which suggests that having additional empty modules in southern rows tends to relieve congestion. At higher levels of WIP, there is not much blocking by stored items, so the configurations perform similarly. The retrieval time plot (Figure 15) supports this interpretation.

We suspect that any implementation of GRIDSTORE in

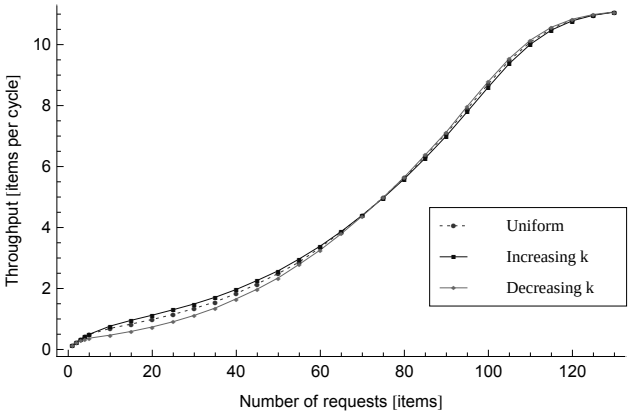

Fig. 13. Throughput for three configurations with different distributions of empty modules.

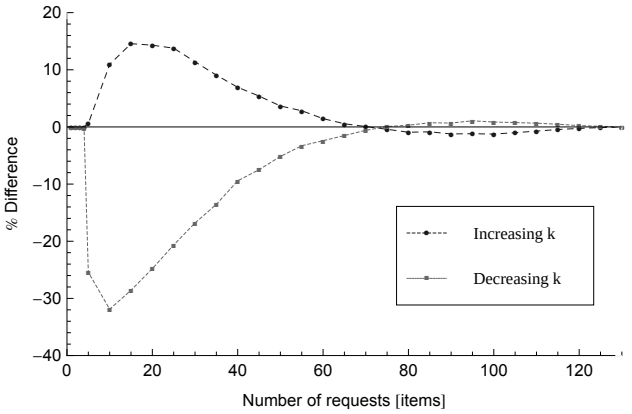

Fig. 14. When the uniform distribution is the base case, percent difference in throughput between the uniform, "decreasing k", and "increasing k" distributions of empty modules.

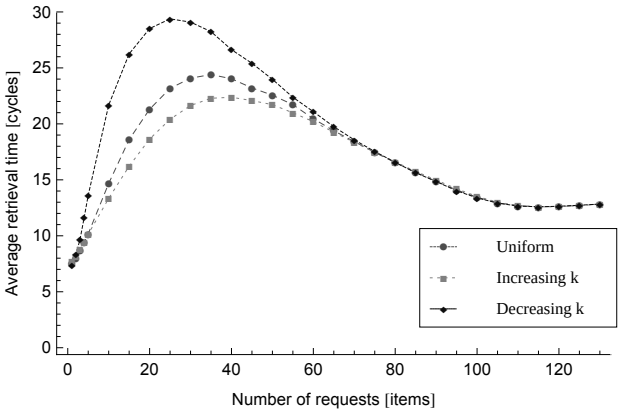

Fig. 15. Average retrieval time for three configurations with different distributions of empty modules.

practice would likely operate at low to medium levels of WIP, and therefore that an increasing k configuration would be preferred.

VII. Conclusions

The GridStore system offers high storage density and high throughput, which previously have been considered conflicting objectives in material handling systems design. Because storage locations are themselves conveyors capable of transport, the system can deliver items at almost any required rate. In order to achieve high density in the third dimension, multiple levels of GridStore could work simultaneously in cooperation with vertical lifts, for example.

We have modeled throughput in GridStore with a constant work-in-process policy in which the number of active requests in the system is constant. For low to medium levels of WIP, throughput increases at an *increasing* rate, in contrast with many vehicle-based material handling systems in which additional requests lead to congestion and increases in throughput at a decreasing rate (or even to decreasing throughput). This feature of GridStore makes it a potentially attractive alternative to traditional automated storage and retrieval systems when the required throughput can fluctuate to high levels—in support of high-speed order picking workstations at peak times, for example. (We are ignoring the cost of such systems, which obviously would be a major criterion for equipment selection.)

The major contribution of our work is to introduce to the material handling community a new way of thinking about material movement on a grid. The GridStore system extends existing research on puzzle-based storage systems [14] by allowing simultaneous retrieval of an arbitrary number of requests. Future research might develop systems that can deliver items to any exterior boundary, rather than to a single side, as GridStore requires.

The complex behavior of GridStore is made possible by a relatively simple decentralized control scheme, which is a second contribution of our work. Decentralized control allows for complex system operations and conflict resolution through electronic negotiation. Effective and, we believe, interesting behavior of the system emerges as a result of these rules. Although decentralized control is not new to material handling, it has been confined almost exclusively to the control of vehicles such as AGVs or shuttles (an exception is [2]). In the context of storage systems design, decentralized control also makes possible a high degree of reconfigurability and scalability. Modules in an implementation of GridStore could be unplugged and replugged together to form systems of any size and (rectangular) shape.

Appendix

Tandem replenishment is accomplished by exchange of home rows in a Willing-Commit cycle: An empty module indicates that it is willing to receive by sending a message to the North. Conveyor modules occupied with replenishing items check if they can use the empty module for tandem replenishment. If so, they send a message to the South initiating the exchange of home rows. There are two message types in this negotiation:

Empty cells initiate a message to the North indicating they are **Willing to Exchange**. The message has two attributes: the origin row (oR) of the message and the forwarded home row (mHR). Initially, the message does not include any home row (-), because the empty conveyor has no item.

Once a **Willing to Exchange** message reaches a module with a replenishing item able to use the empty cell for tandem movement, the module initiates the exchange of home rows. The **Commit Exchange** message has the home row of the initiating conveyor as an attribute (oHR).

The communication protocol for exchange of home rows is shown in Figure 16. A **Willing to Exchange** message is forwarded to the north until it reaches either an empty cell or a conveyor that initiates an exchange of home rows. A module is only allowed to initiate the exchange if the home row of the carried replenishing item is farther away than the empty cell. Modules occupied with stored or replenishing items and unable to initiate an exchange save the mHR attribute and replace it with the home rows of their items. In the communication protocol, the home row of stored and replenishing items is indicated in the middle of the conveyor symbol, and the saved mHR information is indicated in the upper portion.

If the **Willing to Exchange** message reaches a replenishing item that can exchange home rows, the associated conveyor initiates a new **Willing to Exchange** message to the North. Additionally, it takes the mHR attribute of the received **Willing to Exchange** message as its new home row and initiates a **Commit Exchange** message to the South with its old home row as the oHR attribute.

The **Commit Exchange** message is forwarded until it reaches the item north of the empty module that initiated the message passing. Stored and replenishing items receiving a **Commit Exchange** message take the saved home row as their new home row. The last item takes as home row the oHR attribute. In this way, the home row of the replenishing item that initiated the **Commit Exchange** message becomes the home row of the item north of the empty cell.

Acknowledgment

This research was supported by the National Science Foundation (CMMI 0926346) and the Institute for Material Handling and Logistics at Karlsruhe Institute of Technology.

References

[1] K. Furmans, F. Schönung, and K. R. Gue, "Plug-and-Work Material Handling Systems," in *Progress in Material Handling Research: 2010*, K. Ellis, K. Gue, R. de Koster, R. Meller, B. Montreuil, and M. Ogle, Eds. Material Handling Institute, 2010, pp. 132–142.

[2] S. Mayer and K. Furmans, "Deadlock prevention in a completely decentralized controlled materials flow systems," *Logistics Research*, 2010.

[3] Y. A. Bozer and J. A. White, "Travel-time models for automated storage-retrieval systems," *IIE Transactions*, vol. 16, no. 4, pp. 329–338, 1984.

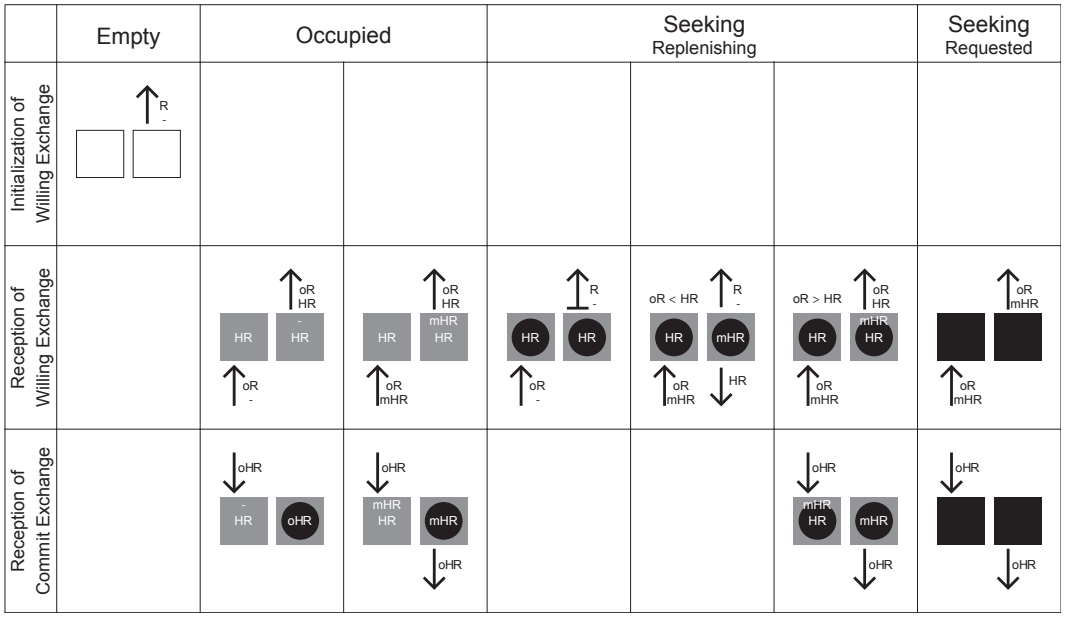

Fig. 16. Communication Rules for Tandem Replenishment

[4] B. R. Sarker and P. S. Babu, "Travel time models in automated storage/retrieval systems: a critical review," *International Journal of Production Economics*, vol. 40, pp. 173–184, 1995.

[5] R. B. M. De Koster, T. Le-duc, and Y. Yu, "Optimal storage rack design for a 3-dimensional compact AS/RS," *International Journal of Production Research*, vol. 46, no. 6, pp. 1495–1514, 2008b.

[6] Y. Yu and R. B. M. De Koster, "Optimal zone boundaries for two-class-based compact three-dimensional automated storage and retrieval systems," *IIE Transactions*, vol. 41, pp. 194–208, 2009.

[7] Y. Yu and R. De Koster, "Sequencing heuristics for storing and retrieving unit loads in 3D compact automated warehousing systems," *IIE Transactions*, vol. 44, pp. 69–87, 2012.

[8] L. Zhang, A. Krishnamurthy, C. J. Malmborg, and S. S. Heragu, "Variance-Based Approximations of Transaction Waiting Times in Autonomous Vehicle Storage and Retrieval Systems," *European Journal of Industrial Engineering*, vol. 3, no. 2, pp. 146–169, 2009.

[9] B. Ekren, S. S. Heragu, A. Krishnamurthy, and C. J. Malmborg, "Simulation Based Experimental Design to Identify Factors Affecting Performance of AVS/RS," *Computers & Industrial Engineering*, vol. 58, no. 1, pp. 175–185, 2010.

[10] B. Ekren and S. S. Heragu, "Simulation Based Regression Analysis for Rack Configuration of Autonomous Vehicle Storage/Retrieval Systems," *International Journal of Production Research*, vol. 48, no. 21, pp. 6257–6274, 2010.

[11] B. Ekren and S. Heragu, "Performance Comparison of Two Material Handling Systems: AVS/RS and AS/RS," *International Journal of Production Research*, 2011.

[12] B. Ekren, S. S. Heragu, A. Krishnamurthy, and C. J. Malmborg, "Semi-Open Queuing Network Approach for Modeling Autonomous Vehicle Storage and Retrieval System," *IEEE Transactions on Computer Automation and Systems Engineering*, 2011.

[13] K. R. Gue, "Very High Density Storage Systems," *IIE Transactions*, vol. 38, pp. 93–104, 2006.

[14] K. R. Gue and B. S. Kim, "Puzzle-based storage systems," *Naval Research Logistics*, vol. 54, no. 5, pp. 556–567, 2007.

[15] A. Alfieri, M. Cantamessa, A. Monchiero, and F. Montagna, "Heuristics for puzzle-based storage systems driven by a limited set of automated guided vehicles," *Journal of Intelligent Manufacturing*, 2010.

[16] G. D. Taylor and K. R. Gue, "The Effects of Empty Storage Locations on Puzzle-Based Storage Systems," in *Proceedings of the Industrial Engineering Research Conference*, 2008, pp. 519–523.

[17] 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*, vol. 4, no. 3, pp. 76–88, 1984.

[18] R. Sharma and J. Aloimonos, "Coordinated motion planning: The warehouseman's problem with constraints on free space," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, pp. 130–141, 1992.

[19] G. W. Flake and E. B. Baum, "*Rush Hour* is PSPACE-complete, or 'Why you should generously tip parking lot attendants'," *Theoretical Computer Science*, vol. 270, no. 1–2, pp. 895–911, 2002.

[20] R. A. Hearn and E. D. Demaine, "PSPACE-Completeness of Sliding-Block Puzzles and Other Problems through the Nondeterministic Constraint Logic Model of Computation," *Theoretical Computer Science*, vol. 343, no. 1–2, pp. 72–96, 2005.

[21] E. D. Demaine and R. A. Hearn, "Constraint Logic: A Uniform Framework for Modeling Computation as Games," in *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity*, 2008.

[22] G. W. DePuy and G. D. Taylor, "Using Board Puzzles to Teach Operations Research," *INFORMS Transactions on Education*, vol. 7, no. 2, 2007.

Kevin Gue is the Tim Cook Associate Professor of Industrial Engineering at Auburn University. He graduated from the U.S. Naval Academy in 1985 with a Bachelor's Degree in Mathematics. He received his Ph.D. from the School of Industrial & Systems Engineering at Georgia Tech in 1995. From 1995 to 2004, he served on the faculty of the Naval Postgraduate School. Dr. Gue's research addresses the design and control of logistics systems. He is a past president of the College-Industry Council on Material Handling Education.

Kai Furmans is Full Professor and Head of the Institute for Material Handling and Logistics at the Karlsruhe Institute of Technology, Germany. He graduated with a Diploma in Industrial Engineering (Wirtschaftsingenieur) from Karlsruhe University in 1988, and received his Ph.D. in Mechanical Engineering also from Karlsruhe University in 1992. As a post-doc he was a visiting researcher at Thomas J. Watson Labs IBM in Yorktown Heights, and received his venia legendi for Logistics in 2000 from Karlsruhe University. He worked from 1996–2003 for Robert Bosch GmbH in several positions in logistics, his final position being head of divisional logistics for the thermotechnology division. His research interests are design of material handling systems and models for supply chain material handling systems.

Zäzilia Seibold is a Ph.D. candidate at the Institute for Material Handling and Logistics at the Karlsruhe Institute of Technology, Germany. In 2011, she graduated with a French-German double diploma in Mechanical Engineering from Karlsruhe Institute of Technology, Germany, and from the National Institute of Applied Science in Lyon, France. Her research addresses design and evaluation of decentralized control algorithms for material handling systems.

Onur Uludağ is a Ph.D. candidate in Industrial & Systems Engineering at Auburn University. He graduated from the Istanbul Technical University in 2008 with a Bachelors Degree in Management Engineering. He received his MISE Degree from the Department of Industrial & Systems Engineering at Auburn University in 2011. His research interests are modeling and design of concurrent distributed systems and grid-based logistics systems with decentralized control.