Optimization and Coordinated Autonomy in Mobile Fulfillment Systems

John J. Enright and Peter R. Wurman
Kiva Systems
North Reading, MA
{jenright,pwurman}@kivasystems.com

Abstract

The task of coordinating hundreds of mobile robots in one of Kiva System's warehouses presents many challenging multi-agent resource allocation problems. The resources include things like inventory, open orders, small shelving units, and the robots themselves. The types of resources can be classified by whether they are consumable, recycled, or scheduled. Further, the global optimization problem can be broken down into more manageable sub-problems, some of which map to (hard) versions of well known computational problems, but with a dynamic, temporal twist.

Introduction

Kiva's innovative approach to warehouse automation uses hundreds of custom-built mobile robots that carry small shelving units and deliver products to human operators. The humans stand at work stations along the perimeter of a storage area that is filled with thousands of storage shelves. The robots fetch specific inventory shelves from the storage area and bring them to the stations where, guided by software and laser pointers that identify the proper inventory on the shelving unit, the operators pick the inventory and put it into the outgoing shipping cartons. The Kiva system has many advantages (Wurman, D'Andrea, and Mountz 2008), not the least of which is that, by eliminating the walking required in traditional warehouses, the system makes operators 2 to 3 times more productive.

An example Kiva robot and pod are shown in Figure 1. The robots are relatively simple from a mechanical point of view. They have a pair of side-mounted drive wheels and a lifting mechanism capable of raising inventory pods about two inches into the air. The robots are bi-directional, and have sensors mounted on front and back for obstacle detection. The robot's navigation system involves a combination of dead reckoning and cameras that look for fiducial markers placed on the floor during system installation. While the mechanics are straightforward, the onboard control system that allows the robot to operate at an industrial level of reliability is quite sophisticated.

As interesting as individual robots are, they are to a warehouse what taxis are to a city. The complexity of the warehouse is truly expressed in the warehouse control software that runs on the servers.

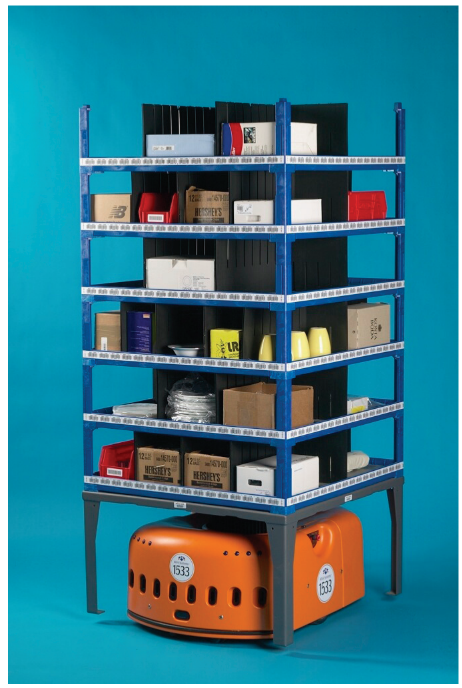

Figure 1: An inventory pod being carried by a Kiva robot.

From a computer science point of view, Kiva's mobile fulfillment system represents an exemplary multi-agent application. Each robot (drive unit, in Kiva jargon) and each work station is an independent agent with some measure of autonomy. There are also several agents that make allocation decisions that are critical to the overall productive behavior of the whole. These agents are distributed across several blade servers and the desktop computers at each work station.

The objective of this paper is not to explain the algorithms and optimizations that Kiva has deployed. Rather, it is to describe the problem domain with enough detail to encourage other researchers to investigate it. We believe that the problem domain embodies many fascinating computational challenges that can be approached with tools from a variety of disciplines.

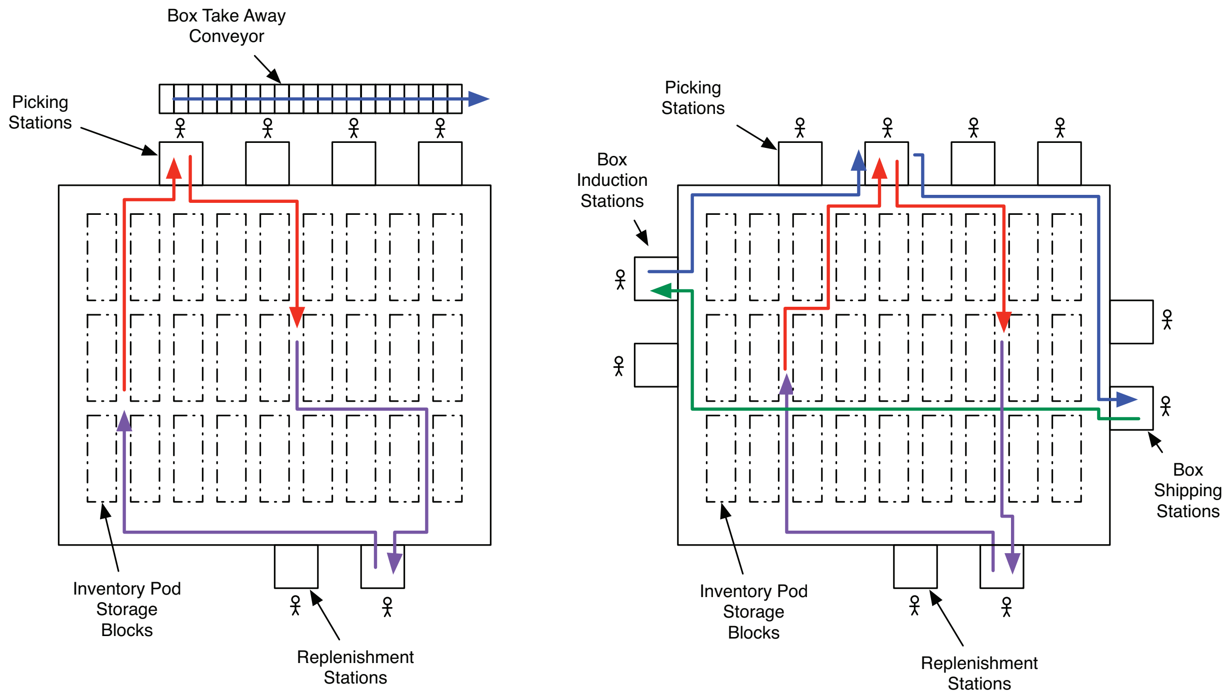

Figure 2: The ItemFetch configuration (left) and the OrderFetch configuration (right).

Resources in a Kiva System

As Kiva's product line has grown, the complexity of the resource allocation problems has increased. In order to understand the range of allocation decisions the system makes, it is necessary to understand the major flows within the system. At the same time, we've simplified the following description to represent a typical implementation. The actual system supports considerable variation on these themes, and includes subsystems not dealt with here, like inventory counting, robot recharging, and trash removal.

Figure 2 shows the two main flows within a Kiva system. The left image shows a warehouse configured with the ItemFetch configuration of a Kiva system. In this configuration Kiva is responsible for transporting the inventory, but not the shipping cartons. The *inventory cycle* (red and purple arrows) captures the lifecycle of the shelving units that hold products in the warehouse. These shelving units, called *pods*, are logically a collection of storage locations, called *bins*. Bins are filled by humans at *replenishment stations*, and are incrementally emptied–again by humans–at picking stations. In between these two activities, they may be stored in the storage blocks, like cars in a large parking lot. In an ItemFetch configuration, the pick workers build the shipping cartons and push them onto a take-away conveyor to transport the finished boxes to the shipping area (blue arrow).

However, a Kiva system can also be configured to transport the orders, as shown in the right image of Figure 2. In such cases, we deploy a second set of pods designed to hold outgoing shipping cartons. These pods travel in the *order cycle* (shown as blue and green arrows). An empty order pod first goes to a *box induction station*, where an operator builds empty boxes and stages them on the order pod. An order pod may carry as many as a dozen orders at a time, though the actual capacity of the pod varies considerably between implementations. The order pods then travel to pick stations where they park alongside the operator. Robots bring inventory pods to the front of the operator, who is directed by the station agent to move the required product from the inventory pod to the appropriate carton on the order pod. Once all of the orders on an order pod have been fully picked, a robot picks up the order pod and returns it to storage. At the appropriate time, the order pod is delivered to a shipping station, where the completed boxes are removed and put on delivery trucks. Note that between any two stations in the cycle, the system may be required to store the pod until the appropriate time to take it to the next station.

When configured with order pods, we refer to the system as an OrderFetch implementation. Note that such a system still includes inventory pods and the inventory cycle. An OrderFetch system enables the customer to have random access to finished orders, and greater flexibility to hit truck delivery deadlines. Often, an OrderFetch system is more cost-effective than an ItemFetch system and a conveyor-based buffering and sorting solution for finished packages.

It is illuminating to view the system as a collection of multiple types of scarce resources with different attributes:

- Inventory: naturally, physical units of inventory are a consumable resource. Units of inventory follow a cycle from replenishment to available on a pod, to allocated to an order, to picked into an order. A typical customer may have 10,000 to 100,000 or more unique product types. Each is described by its dimensions, packaging quantities, and its *velocity* (the frequency with which it is ordered). It is very common in practice for inventory velocities to follow the 80/20 rule: 20% of the products account for 80% of the order volume.

- Inventory pods: pods are a shared, scheduled resource. Only one robot can carry a pod at a time, and the pod can visit only one station at a time, although it can be scheduled to visit more than one station sequentially. Pods may be square or rectangular, and the bins on each shelf may be accessed from one or more of the four faces of the pod. A typical installation may have 5,000 or more storage pods.
- Storage bins: bins on pods are a recyclable resource. Bins are filled at replenishment stations and decremented at pick stations. Once they are completely emptied, they become available again for new products. Unlike traditional warehouse systems, a storage location in Kiva does not have to be filled with the same inventory that it previously held. Because pods can consist of one to one hundred bins, the number of storage bins in a Kiva system can easily exceed 100,000 addressable locations.
- Shelf space for orders: in an ItemFetch configuration, workers put boxes onto a shelf and fill them with inventory from the pods. The shelves are limited in size, and can hold only a certain number of boxes at a time. This shelf space is recyclable in the sense that once the worker pushes a completed order onto the conveyor, that shelf space becomes available for another order. A typical ItemFetch station can have 4 to 20 orders staged at a time.
- Orders: orders are another consumable resource. It may be counter-intuitive to think of them as such, because they represent the output of the system. However, a core allocation problem is how to choose one order from a set of available orders to assign to stations. Once an order is assigned, it is removed from the pool of available orders. Orders themselves are defined by a list of *line items*. Each line represents the demand for a specific quantity of units of a particular product-type. The number of lines in an order varies considerably between customers, but is usually well-modeled as an exponential distribution. Customers that use Kiva for retail restocking often know the orders the night before, and so the available pool of assignable orders at the beginning of the day represents an entire day's work. On the other hand, customers that use Kiva for e-commerce fulfillment receive orders all day–often with a peak in the afternoon–and have to fill the orders the same day. In such cases, there is a smaller rolling window of available orders.
- Order pods: when the system is in an OrderFetch configuration, workers induct empty boxes onto order pods at an induction station. These pods then travel to picking stations and shipping stations. Order pods can carry from two to eight orders, and a typical installation will have 250 such pods. Importantly, while order pods themselves are scheduled resources, they also represent an *aggregation* of orders for downstream allocation steps. The performance of the system can depend heavily on which orders are placed together on an order pod.
- Parking spaces: because inventory pods, and to a lesser extent order pods, spend much of their time parked, waiting until they are needed, the effective allocation of parking spaces is also critical. Parking spaces fill the interior of any Kiva installation in blocks, like a well-planned city. An installation will have as many parking spaces as pods so that when all of the workers go home for the night (if they go home at all), every pod has a place to park. We consider parking spaces recycled resources because once a pod is removed from a space, that space becomes available for any other pod to be stored in it. In the same way that bins are not required to have the same products replenished into them after they've become empty, parking spaces are not reserved for any particular pod.
- Robots: the robots themselves are shared, scheduled resources. Robots can carry only one pod at a time, and they can visit only one station at a time. They are dual purpose, and can carry both inventory and order pods. To date, Kiva has installed several solutions with over 500 robots each, and one installation that has 1,000 robots.
- Stations: the perimeter of a Kiva system is often lined with picking and replenishment stations. In OrderFetch installations, the induction and shipping stations are also mixed in. Stations are a scheduled resource, but they differ from the others in that they have the ability to physically buffer arriving pods in a queue for rapid-fire picking. Depending on the types of products and the tasks involved, operators may interact with a pod for as few as four seconds or as long as two minutes, and it is very important to keep all the operators both busy and load balanced. A large Kiva system can have 150 or more stations.

Because of the tight interaction of these semi-autonomous agents and their resources, we refer to the system behavior as *coordinated autonomy*: the agents operate largely autonomously when working on their individual tasks, but because they compete intensely for resources, the system provides a significant amount of coordination support through the careful allocation of resources.

Interesting Allocation Problems

At a high level, optimizing the system requires a dual objective function: keep the operators as busy as possible while minimizing the amount of equipment necessary, particularly pods and robots. These objectives correspond to minimizing the operational expenses (people) and the capital expenses (equipment) subject to the constraint that all of the work must be completed each day. The two objectives are not always compatible; for example, a solution with a robot per inventory pod would be most likely to keep the workers busy at all times, but would also be unacceptably expensive. In practice, we assume keeping all of the workers fully occupied is a constraint, and therefore the objective is to minimize the equipment.

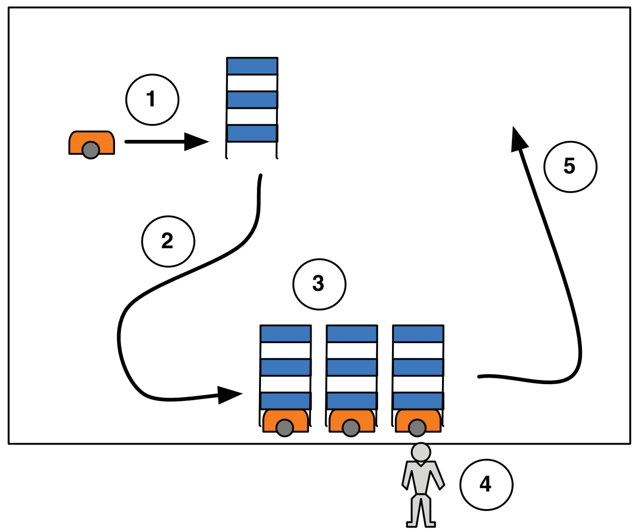

Figure 3: The steps of a robot's task to deliver inventory.

The main mechanism that we have to reduce the equipment costs is to do more work with fewer robots. A robot's task involves the five steps shown in Figure 3:

1. Drive from the robot's current location to the pod's current location.
2. Carry the pod from its current location to the station's queue.
3. Queue at the station until it is the pod's turn.
4. Let the operator pick inventory from the pod.
5. Store the pod back into the storage area.

Improving any or all of Steps 1, 2, 3, or 5 will help reduce the equipment required to sustain the operator rate. However, once the robot and the pod are selected, there is not much you can do to reduce the length of the legs of the mission. All of the choices that reduce the driving distances are made *before* the robot and pod are selected.

Perhaps the biggest lever for reducing the number of robots needed is to increase the average number of items picked from each pod that makes a delivery. For example, if we changed the average number of items picked during a delivery from 1.0 to 2.0, we would need about half of the number of robots.

We refer to this general concept of number of lines picked per pod as *pile-on*. It comes in several varieties, some of which are more amenable to improvement than others. *Bin pile-on* occurs when two or more orders both require product from the same bin, at the same station, at the same time. *Face pile-on* occurs when multiple bins are accessed on the same face of the pod during one face presentation by the robot. The items being picked may be for a single order, or for multiple orders at the station. In contrast to face pile-on, *station pile-on* captures all inventory removed by that station from that pod during that visit, even if the picking requires that the pod be rotated in order to be picked from the other side. Finally, there is *mission pile-on* which captures the number of items picked during the the robot's complete engagement with the pod. A robot journey that takes a pod to three stations, each of which picks one item, would have individual station pile-on of 1.0, but have 3.0 mission pile-on for the entire trip.

One of the things that makes the Kiva system so attractive from an algorithmic point of view is that, while the overall computational problem is an intractable, dynamic, stochastic optimization with incomplete information, it is amenable to decomposition into very approachable sub-problems. It remains to be seen whether approaches that address the global optimization problem perform better than those that decompose the problem. In the following, we highlight some of the sub-problems and discuss their impact on the above steps of a robot mission.

Inventory Pod Selection Problems

Perhaps the most straightforward optimization opportunity occurs when we have to select a pod to deliver inventory to a station. Typically, a station has multiple orders that it is working on with multiple open lines to pick. Thus, when selecting the next pod to deliver we have from just a few to potentially hundreds of pods that have inventory that would satisfy some line at the station. The pods vary by distance, and the set of open lines they satisfy. At a particular instant in time, this problem corresponds to the multi-set multi-cover problem (Dobson 1982; Rajagopalan and Vazirani 1993). However, in reality, the problem has a temporal dimension that is rarely explored; once an order (or order pod) is completed at the station, another order (or order pod) will take its place, bringing even more open lines with it. The timing of the arrival of the new order relative to the arrival of previously selected inventory pods will also affect the efficiency of the system.

The formulation of optimal pod selection should also take into account both the pros and cons of selecting a pod that is already scheduled to visit other stations. When a pod bounces from one station to another–*station hopping* in Kiva jargon–we get to amortize the costs of Steps 1 and 5 across multiple station visits. This benefit is counter-balanced by the fact that station hopping usually introduces a further delay before the pod arrives at a station, and a dependency on the workers at the previous station(s). Moreover, the determination of the optimal sequence in which to visit the stations is an instance of a traveling salesman problem (discussed below).

Pod Storage Allocation

When we are finished with a pod, whether it is an inventory pod or an order pod, we need to decide where to store it. There is an obvious tension between storing it nearby, and reducing Step 5, and storing it near where it is likely to be used next, reducing a future Step 2. At the same time, there is an opportunity cost associated with putting a slow pod near the picking stations. A slow pod might move once every two hours, while a fast pod might move every 15 minutes. We would prefer to use the most convenient parking spots for the faster pods because, over a two hour period, we will save time on eight trips for the fast pod, but only one trip had we stored the slow pod in the good parking spot.

Order Allocation Problems

In the above discussion, we assumed that the orders were present at the station when the pods were being selected. However, the decision about which orders to assign to a station is one of the most critical in the system. First, we consider the problem in an ItemFetch configuration where the outgoing shipping containers are built at the pick station and placed on tables or shelves to be filled. In that configuration, an operator is typically working on 4 to 12 orders. As an order completes, and the operator pushes it onto a shipping conveyor, the shelf location becomes available for a new order. Typically, there are hundreds of new orders waiting to be worked on. When choosing a new order to assign to the station, we can examine the inventory that will be needed to fill it, the open lines of the other orders already at the station, and the types of inventory stored on pods near the particular station. By making wise decisions about order allocation we can both increase pile-on (Step 4) and decrease the time to carry inventory (Step 2).

In an OrderFetch system, order allocation is really the act of selecting a new order pod to replace the order pod that is leaving the station. The same considerations apply to the selection of the order pod as selecting an order to put on a shelf in ItemFetch. However, because order pods also need to be delivered to stations by robots, there is a complex coordination that must take place to smoothly remove one pod and bring in its replacement with as small a time gap as possible.

The OrderFetch scenario adds another layer of complexity to the management of orders. At the induction station, generally well before the time at which we can deliver an order pod to a picking station, we have to decide which orders to put together on the order pod. Clearly, putting together orders that have lines in common is likely to improve bin pile-on. A more difficult and interesting question is whether we can anticipate which station the order pod will be assigned to, what other order pods will be present at the station, and what inventory might be around that station at the time we assign this order pod. Good decisions at this point can increase pile-on later, and reduce the time spent taking pods to stations.

Once the orders are filled, the order pod is eligible for shipping. Shipping stations are typically configured to load particular trucks, like Fedex or UPS, and usually have deadlines. Some customers fill large, 40' trailers, which go to carrier hubs like UPS, while others fill small vans which do local deliveries. Although the allocation of an order pod to the shipping station is relatively straightforward, the priorities and sequencing of these truck deadlines influences many of the decisions upstream. For example, if the Fedex truck is leaving at 6:00pm and the UPS truck is leaving at 7:00pm, then around 5:00 priority should be given to Fedex at both the induction and the picking stations. Then, once the 6:00 deadline has passed, Fedex should be de-prioritized because the remaining orders have to wait until tomorrow for the next pickup.

Replenishment Allocation Problems

A typical warehouse holds three days worth of inventory. This means that each day, one third of the inventory is emptied and must be refilled. Another critical allocation decision is the one that selects a bin in which to store product when the replenishment operator stages inventory for put-away. The replenishment decision must take into account the physical properties of the product and those of the available bins, and should try to maximize the cubic utilization of the pods. However, by also looking at the other products on the candidate pods, there are opportunities to improve things even further. For example, by putting products of similar velocity together, we can create "faster" pods and "slower" pods. Then we can store the fast pods near the pick stations, and the slow pods near the back of the warehouse. By doing so, we should reduce the average amount of time spent on Step 2. A similar idea that instead addresses pile-on, and thus Step 4, is to try to look for product synergies, like storing peanut butter with jelly. In theory, leveraging these synergies would increase the likelihood that we will get pile-on.

Yet another dimension of replenishment is determining the ideal number of bins that a product should be stored in. A slow product will probably occupy only one or two bins in the entire system. But a high velocity product may occupy a hundred bins just because many more units move through the warehouse on any given day. The replenishment decision should account for the product's overall presence in the warehouse, and perhaps drive towards some ideal number of bins for a given product to occupy.

Robot Allocation Problems

Now we turn our attention to the task of assigning robots to delivery tasks. Given that a pod is selected, there are three basic problems.

First, we have the problem of deciding which robot will be allocated to a pod that needs to be delivered. In a busy Kiva system, there are not many robots sitting around idle, but there are robots completing missions frequently. If there are 1000 robots, and each does a 5 minute round trip to deliver inventory, then 3 to 4 robots are setting down their pod every second and become available for another mission. Furthermore, we have some visibility into robots that will be coming free in the near future. Given a large set of pods that need to be delivered and a set of robots that are free or soon to be free, we have a straightforward matching problem to associate the robots to the pods, with the objective of minimizing time spent in Step 1.

However, the problem becomes more interesting when we consider whether we *should* allocate a robot to a pod at the current moment in time. In order to minimize time spent in queues we need to avoid sending robots to a station faster than the operator can process them. To further complicate the problem, different operators perform their tasks at different speeds, and some products are naturally harder to pick than others. Also, some stations are positioned on the floor in more advantageous locations–near the inventory–while others are stuck in corners and the average time to deliver inventory is higher. All of these things need to be taken into account in order to balance the allocation of robots across stations and reduce the amount of time spent in Step 3.

Having allocated a robot to a pod, the second step is to compute a path for the robot to drive from its current location to the pod, and then from the pod's location to the destination. Path planning for a single robot is perhaps the most well understood component of a Kiva system, especially because the entire warehouse is laid out as a simple grid. One can apply well known algorithms like A* or variations of Djikstra's algorithm. However, path planning in an environment with the density of moving robots that we find in a Kiva system is much less well understood. Some research (Roozbehani and D'Andrea 2011; Smith et al. 2010; Treleaven, Pavone, and Frazzoli 2011) has begun to address issues that would benefit a Kiva system. Improvements in this area would help Steps 1, 2, or 5 by making the actual travel between points A and B more efficient.

A final area of research is the TSP problem mentioned above, which occurs when a robot must carry a pod to multiple stations. On the one hand, if the robot were the only concern, we could formulate a TSP with the objective of minimizing the total travel time. This basic problem is made more interesting by the fact that we actually have a dynamic TSP (Pavone et al. 2009), in which occasionally more stations are added to the list of places to visit.

However, the real complexity arises from the way the delivery schedule interacts with the order management at a station. In an ItemFetch environment, an order can sit on the shelf waiting for its last item of inventory and not dramatically affect the other orders, because the pick worker can fill other orders while she is waiting for the station-hopping pod. Even so, there are instances where all of her remaining orders are waiting for items that are on the pod that is scheduled to visit other stations before coming to hers. This basic problem is exacerbated in an OrderFetch environment where an entire order pod may be waiting for the inventory that is station hopping. When an order pod is held up for more than a few seconds, it can have a dramatic negative impact on the the number of open lines that the worker can pick to, which affects all of the other metrics. Finding an appropriate way to apply TSP techniques while still preventing starvation of the pickers is an interesting problem.

Alphabet Soup

In order to facilitate research into the above problems, in 2006 we made available a Java platform that provides an abstraction of the warehouse environment. In this simulation we call Alphabet Soup (Hazard, Wurman, and D'Andrea 2006), the products in the warehouse are colored letter tiles. The objective is to assemble words constructed from the correctly colored tiles. The letters are stored in buckets which can be carried around by flying robots called bucketbots. Letters are replenished into the system as bundles handled at replenishment stations. The order profile is generated by feeding the simulation a dictionary. The natural frequency of letters in the language, along with a configurable variation in color frequency, create a realistic profile of product velocities.

Currently, Alphabet Soup does not support an OrderFetch model, but all of the basic inventory allocation problems are represented, along with the basic issues associated with high-density robot coordination.

Conclusion

Mobile fulfillment systems, like Kiva's, represent a relatively new and understudied class of resource allocation problems. The problem of coordinating robots in warehouses brings together computational problems for a variety of disciplines, including scheduling, decision making under uncertainty, data mining, learning, robot path planning, and classic optimization. Further, the Kiva system represents a natural multi-agent system in which to study issues of coordinated autonomy and decentralized decision making. In short, we think the Kiva system, or an abstraction like Alphabet Soup, is an environment that presents a wide variety of interesting and practical problems, and we encourage research in the area.

Acknowledgments

A working robotic system like Kiva's is the product of a great deal of effort by a large number of talented mechanical, electrical, and software engineers. We thank the world-class Kiva employees who have built such a fascinating product.

References

Dobson, G. 1982. Worst-case analysis of greedy heuristics for integer programming with nonnegative data. *Mathematics of Operations Research* 7(4):515–531.

Hazard, C. J.; Wurman, P. R.; and D'Andrea, R. 2006. Alphabet soup: A testbed for studying resource allocation in multi-vehicle systems. In *Proceedings of the 2006 AAAI Workshop on Auction Mechanisms for Robot Coordination*, 23–30.

Pavone, M.; Bisnik, N.; Frazzoli, E.; and Isler, V. 2009. A stochastic and dynamic vehicle routing problem with time windows and customer impatience. *MONET* 350–364.

Rajagopalan, S., and Vazirani, V. 1993. Primal-dual RNC approximation algorithms for (multi)-set (multi)-cover and covering integer programs. In *Proceedings of the 1993 IEEE 34th Annual Foundations of Computer Science*, 322–331.

Roozbehani, H., and D'Andrea, R. 2011. Adaptive highways on a grid. In Pradalier, C.; Siegwart, R.; and Hirzinger, G., eds., *Robotics Research*, volume 70. Springer Berlin / Heidelberg. 661–680.

Smith, S. L.; Pavone, M.; Bullo, F.; and Frazzoli, E. 2010. Dynamic vehicle routing with priority classes of stochastic demands. *SIAM J. Control and Optimization* 3224–3245.

Treleaven, K.; Pavone, M.; and Frazzoli, E. 2011. An asymptotically optimal algorithm for pick-up and delivery problems with application to large-scale transportation systems. Submitted.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine* 29(1):9–20.