Preprints of the 2012 20th Mediterranean Conference on Control & Automation (MED), Barcelona, Spain, July 3-6, 2012

Task Assignment and Agent Coordination in a Warehouse Environment

Demetris Stavrou and Christos Panayiotou

Abstract— **Automated guided vehicles (AGVs) are widely used in warehouse and manufacturing facilities for point to point transportation of cargo. A typical objective in such facilities is space efficiency (store as many items as possible in a limited space) and throughput maximization. The two objectives are conflicting. Storing more items limits the available space for paths that can be used by the AGVs to transport goods within the facility, forcing the AGVs to take longer, more congested paths limiting the facility's throughput. Therefore, to maintain acceptable throughput, an efficient task allocation and vehicle coordination is required. In this paper we study the automation of such vehicles in a warehouse with a specific topology. In the topology considered, vehicle movement is extremely constrained making the overall system prone to deadlocks. Delays caused by deadlocks are significant and seriously affect operational performance of the warehouse. Our main objective is to derive an efficient task assignment and agent coordination policy such that the overall system throughput is maximized. We present a problem transformation derived by the constraints and we propose a specific policy which is scalable and requires minimal computational power therefore it can be applied in real time.**

I. Introduction

In a warehouse, cargo containers have to be moved between loading areas and storage areas. In order to automate this operation, special automated guided vehicles (AGVs) are used. AGVs are assigned the task of transporting a container from a specific point in the storage area, to the loading area, to be loaded onto a truck or ship (or the other way round). To limit the storage cost, operators have an incentive to increase the number of containers that are stored in the facility, however, this limits the available space remaining for the AGVs to move around in the facility. Limiting the space available for paths can lead to inefficiencies that need to be avoided to the extent possible. For example, a deadlock situation can occur when two AGVs travel towards each other in opposite directions, in a path wide enough to fit only one of them. We point out that this scenario is often possible since establishing one-way directions on each path may not be attractive since it can lead to even longer average path length for the AGVs. To maximize the throughput of a facility, one needs to solve the container allocation problem (which AGV will get which container), the path allocation problem (which path will be used by the AGV) and the coordination problem (how an AGV will interact with other AGV's to avoid deadlocks). The solution to these problems is the topic of this paper.

The problem relates to the *Vehicle Routing Problem* (**VRP**)[1] which considers a fleet of vehicles, all with the same capabilities, used to deliver goods to a set of customers located at different locations and then return back to the depot. Even though several variations of the problem have been proposed, to the best of our knowledge, none has addressed the specific topology we are considering in this paper. Moreover, the formulation of the VRP is at a higher level and is generally not addressing vehicle to vehicle interactions e.g., deadlocks.

The remaining of this paper is organized as follows. Section II provides an overview of the related work. Section III provides a detailed description of the warehouse problem. Section IV presents the more general problem formulation and problem solution. In Section V we present some results showing the performance of our proposed policy. The paper ends with Section VI where we state our conclusions and our plans for future work.

II. Related Work

Vehicle scheduling and routing problem in transportation, distribution and logistics has been studied since 1959 when Dantzig and Ramser described the VRP. The subject has received great attention by many researchers and many solutions have been proposed for the VRP [2] as well as its variants, *Vehicle Routing Problem with Time Windows*, *Capacitated Vehicle Routing Problem* and *Vehicle Routing Problem with Pick-Up and Delivery*.

In the context of automated warehouses [3], vehicle routing has been addressed by several authors while the proposed solutions for automated guided vehicles fall under three categories [4]. In *(i) Static methods* [5], the paths are assigned to a single vehicle and cannot be used by any other vehicle until the destination is reached. *(ii) Time-Window* based methods like [6],[7] and more recently [8],[9], increase utilization of the available paths, as the agents reserve path segments only for a short period of time. Because of the complexity, computations are done a priory. In some approaches, achieve on-line performance by reducing the solution space using priorities. Finally, in *(iii) Dynamic methods* [10], the route planning is done completely on-line as vehicles are moving. Because of the nature of the problem, the numerous methods proposed to solve its general form, introduce the same tradeoff dilemma of performance vs. computational time.

In a less constrained topology the problem of multi-agent routing has also been addressed in [11] and references therein. The problem of deadlock-free routing has also been

This work is partially supported by the Cyprus Research Promotion Foundation's Framework "Programme DESMI 2008", co-funded by the Republic of Cyprus and the European Regional Development Fund under Grant *New Infrastructure Project/Strategic/0308/26*.

D. Stavrou and C.G. Panayiotou are with the KIOS Research Center and the Department of Electrical and Computer Engineering, University of Cyprus, CYPRUS demetris.stavrou@ucy.ac.cy, christosp@ucy.ac.cy

978-1-4673-2529-5/12/$31.00 ©2012 IEEE

considered in several environments using Petri Nets or graph theoretic approaches [12], [13], [14].

To the best of our knowledge the warehouse topology we are considering in this paper has not been studied explicitly. In such warehouse, application of the above methods would most likely result in inefficient policies due to the constraints introduced. In the context of warehouse environments with a specific topology under the assumption of cyclic scheduling, our contribution is the development of a new, computationally efficient, deadlock free policy that solves the task assignment and agent coordination problems.

III. Problem Description

Fig. 1 presents the topology of the warehouse that we consider in this paper with four agents (AGVs) that are marked as A, B, C and D. The area is generally divided into two main sub-areas; *R1* is a "free" moving area (no constrained paths) and *R2* is the storage area where all containers are stored. Containers need to be transported between the storage area *R2* and the loading area of *R1*. In *R1* agents are allowed to move freely, in all directions. The free space in *R1* is an intermediate space between the container stack and the loading area. Agent movement in that area in not in the focus of this work, therefore it is assumed that agents have the hardware and software required to avoid any collisions while traveling there.

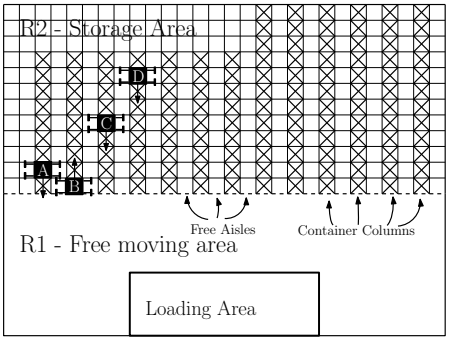


Fig. 1. Warehouse Layout

It is assumed that the warehouse has a set A with $|A| = n$ similar agents used for the container transportation. We define *task* s_j, $j = 1, 2, \cdots$ the location of a container in *R2* that needs to be transported from its current location to the loading area in *R1*. The operator of the warehouse generates a sequence of tasks S

$$S = \{s_1, s_2, ..., s_k\}, \quad k \in \mathbb{N}$$

where it is assumed that $k \gg n$. In general, these tasks need to be completed in the order that they are issued or *almost* with the order they are issued. For example, tasks s_1, s_2, $\cdots$, s_m are issued in this order, then it may be acceptable to complete s_2 ahead of s_1, however, it is not acceptable to complete s_m ahead of s_1 since the loading area may become congested (m depends on the loading area dimensions, the loading cranes etc).

In the proposed approach, tasks will be completed in *cycles* of n i.e. we assume cyclic scheduling. In other words, during the first cycle, the first n tasks will be assigned to the agents. During the second cycle tasks $s_{n+1}, \cdots, s_{2n}$ will be served and so on. For every cycle, the objective of the paper is to assign tasks to agents and then solve the agent coordination problem such that the cycle will complete in minimum time.

A. Warehouse Topology

This research aims towards warehouses with the specific topology shown in Fig. 1. Containers are stacked next to each other, forming long columns leaving narrow free aisles between them where agents can move into. In such environment with limited amount of resources (free aisles), precise coordination and synchronization is needed among the agents, to achieve a good enough coordination plan. It is assumed that the AGVs can move above and lift a container with its wheels on the two sides of the container. The vehicle uses the two free aisles on the two sides of a container column to reach the desired container or to carry the container towards *R1*. Note that the vehicle is tall enough such that when it carries a container there is enough clearance underneath to move without hitting on other containers in the same column.

This topology is efficient, in terms of space, however, it introduces constraints which set it apart from the general topology problems. The container aisles are long, with a single entry/exit point on the edge. The fact that no other alternative access points exist in any other part of the column, means that is not possible to switch columns once the agent enters a column. If an agent enters in the "wrong" column, there are no alternative paths to correct this decision i.e. to switch to another column. To do that, an agent has to exit from the column where it is currently in and then enter the desired one. To do this manoeuvre, no other agent should be in the way. For example, if an agent wants to move down to exit while another agent moves up on the same aisle, then a deadlock occurs.

Another constraint of the problem is the movement of agents in two adjacent aisles. As shown in Fig. 1, when an agent travels in a column it blocks access to the neighbouring columns. The free aisles on the side of the container column are wide enough just for one set of wheels. In Fig. 1, agent *B* is moving up to reach its destination, while agent *A* is trying to move to the exit. A deadlock occurs because each agent blocks the way of the other. To resolve this, one of the two agents has to change its movement direction in order to allow for the other to reach its destination. It is an inefficient way to solve this conflict, thus policies need to be derived that will avoid deadlocks in the first place (before they occur). Note also, that a conflict between two agents can escalate and block more than two agents. For example in the same figure, *A* and *B* will later block agent *C* which will further block agent *D*.

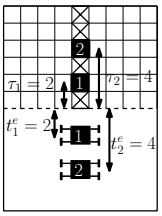


Fig. 2. Two-agent two-task example

B. Two-Agent Two-Task Cycle

The task allocation and AGV coordination is a discrete optimization problem. To gain a better insight on all possible solutions in the solution space, in this section we consider the case of just two AGVs. We assume a warehouse with a single container column and just two agents ($n = 2$, agent 1 and agent 2), therefore the cycle will consist of two tasks, say s_1 and s_2, as shown in Fig. 2. To complete this cycle, one needs to solve the *assignment problem*, i.e., decide if agent 1 will be assigned to s_1 or s_2 while agent 2 will be assigned to s_2 or s_1 respectively. The two possible assignments are denoted as

$$V_1 = \{(1, s_1), (2, s_2)\} \text{ and } V_2 = \{(1, s_2), (2, s_1)\}$$

Once the assignment is chosen, the next problem is how the agents will move to complete their tasks without any conflicts. This is defined as the *coordination problem* which consists of two sub-problems, the *ordering problem* and the *entering mode problem*. The first, defines the order that the two agents will enter the container column. The second problem defines how the agents will enter the column. For the two agent example, assuming the order is agent 1 and then agent 2, then there are two possible entering modes. When agent 1 enters then, either agent 2 enters *before* agent 1 exits, thus both agents will be in the aisle simultaneously, or agent 2 waits for agent 1 to exit and then enters.

Therefore, the solution of the cycle depends on the combination between an *assignment*, an *ordering* and an *entering mode* which we define as the *cycle policy*. Even after the cycle policy is chosen, possible conflicts (deadlocks) have to be resolved. This is done by deciding which agents needs to wait, where and for how long. This is necessary to compute the time needed to complete the cycle. Notice that in this simple example there are two possible assignments, two possible orderings and two possible entering modes, thus there are eight possible solutions that need to be considered. In the general case, each of the sub-problem grows factorially with respect to n, thus, for medium or large problems it is impossible to evaluate all possible solutions. As it turns out, there is no need to evaluate all possible solutions since one can derive an optimal policy as it will be presented next.

IV. Discrete Optimization Problem Formulation

A. Objective Function

There are different methods to measure the performance of an automated warehouse like *maximum throughput*, *minimum travel time* or *even distribution of workload* [15]. In this paper we consider throughput maximization which is equivalent to the minimum time to complete all jobs. Given the cyclic structure of our solution approach, the objective is to determine the assignment and coordination functions that minimize the time to complete each cycle.

Let's consider agent i which is initially placed somewhere in *R1*. Based on a cycle policy of an assignment (V), ordering (R), entering mode (M) the agent is given a task. To complete its objective based on the chosen policy, i will need T_i time units as follows

$$T_i(V, R, M) = t_i^e + w_i^e + \tau_v + t^l + w_i^x + \tau_v + t_i^d \qquad (1)$$

where, t_i^e is the time the agent needs to reach the entrance of the aisle. Before entering, the agent may be required to wait for some time w_i^e for other agents to either enter or exit the aisle (to avoid deadlocks). The time required to travel from the aisle entrance until the location of its task is denoted by τ_v (v being the task assigned according to V). It takes t^l time units for an agent to load the container. Before the agent can exit the aisle, it may need to spend additional time in the aisle (represented by w_i^x) if other agents are obstructing its way. After that, the agent needs τ_v time units to travel from the container location to the exit of the aisle. Finally the time needed to travel to the drop-off location is represented by t^d.

For every agent i, given a cycle policy of an assignment, ordering, entering mode, one can compute $T_i(V, R, M)$ for all i. Cycle k will be completed when the last agent will complete its task, thus, the cycle objective function is

$$\Lambda_k(V, R, M) = \max_i(T_i(V, R, M)), \quad i = \{1, \cdots, n\}$$

For every cycle, the objective is to determine the cycle policy (V, R, M) such that

$$\Lambda_k^* = \min_{(V,R,M)} \Lambda_k(V, R, M) \qquad (2)$$

Hereafter, we will omit writing (V, R, M) when we refer to $T_i(V, R, M)$. Therefore, to derive the best policy, T_i for each agent needs to be computed. To determine T_i, its sub-elements need to be calculated first. In general, t^e can be computed from the starting position of the agent assuming a straight line path to the entrance of the aisle and given a constant agent speed. Also, we assume that t^e will be independent regardless of the tasks assignment or other agents. This is a reasonable assumption since in free space the agents may easily manoeuvre to avoid collisions. t^d is also independent of the assignments to other agents for the same reason. This also can be computed from the distance from the aisle to the loading area. The loading time t^l is fixed for all agents and all tasks. For simplicity in the sequel, it will be ignored and it will be assumed that it is included in τ_v. Now, times τ_v, w_i^e and w_i^x, depend on the task assigned to the agent. Moreover, they are subject to the movement of the other agents.

Note that (1) ignores the case where an agent travels to its assigned task location, then moves further up in the aisle (to allow a following agent pick its task) and then exits. The reason this possibility is left out is because given a set of

tasks it is never optimal (with respect to (2)) for an agent to travel further than its task location. This is shown in the following Lemma the proof of which is omitted due to space limitations.

Lemma 1: Given a set of tasks in a cycle, assume that agent i is assigned task j and thus it will move a distance d_j in the aisle, there is always a better (V, R, M) (w.r.t. (2)) than having agent i travel a distance $d'_j > d_j$.

Sketch of proof: One needs to consider the complete set of (V, R, M) in an environment of two agents and two tasks. Then, derive T_i and T_k for the specific policy (denoted by $(V, R, M)_1$) that causes i to travel $d'_j > d_j$ and $\hat{T}_i$ and $\hat{T}_k$ for a $(V, R, M)_2$ that does not (there is always such a (V, R, M)). One can show that the $(V, R, M)_2$ is always better than the $(V, R, M)_1$.

In order to solve (2) we assume that the two main problems, *assignment problem* and *agent coordination problem*, are decoupled and solve them sequentially. Before we proceed with the solution of the problems, let us present the concept of the *Abstract Time Windows* which is a tool used in visualizing and deriving the optimal solution.

B. Abstract Time Windows

In this paragraph we show how the agent movement can be transformed, inspired by the concept of time-window graphs. Given an agent-task assignment, we know that the earliest time that the agent will enter the aisle is t_i^e. In addition, the shortest period that the agent will occupy the aisle (ignoring t^l) is $2\tau_v$, thus the earliest that an agent can exit from the aisle is $t_i^x = t_i^e + 2\tau_v$. Using t^e and t^x, we can describe the movement of the agent in the aisle and therefore define the *Abstract Time Window* (**ATW**) which is schematically presented in Fig. 3.

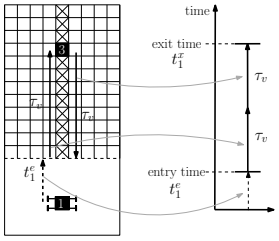


Fig. 3. Abstract Time Window for agent $i = 1$ and task $v = 3$.

In a warehouse where $n > 1$, ATWs are used to detect and resolve possible deadlocks. If the vehicle on-board system fails to detect the other vehicle, it is possible that a deadlock could lead to collision. Therefore ATWs are also a way to avoid collisions. To do that, we define the following condition for conflict detection.

Deadlock-Free Condition (DFC): If $t_i^e < t_k^e$ then either (a) $t_i^x < t_k^e$ or (b) $t_i^x > t_k^x$ for all $i, k = 1, \cdots, n$.

One can use DFC to detect possible deadlocks using the following Lemma.

Lemma 2: A solution is deadlock-free if and only if DFC holds.

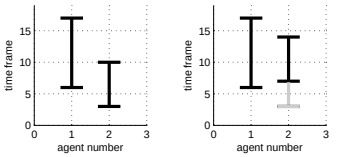
The proof is omitted due to space limitations.

If DFC is violated, then one can use the following two operations in order to resolve the conflict

i Extend t_i^e to $\hat{t}_i^e = t_i^e + w_i^e$, $w_i^e \geq 0$.

ii Extend t_i^x to $\hat{t}_i^x = \hat{t}_i^e + 2\tau_v + w_i^x$, $w_i^x \geq 0$.

In other words, the ATW can be shifted upwards or it can be stretched but it cannot be compressed. A shift upwards implies that the agent needs to wait at the entrance of the aisle for w_i^e time units, while a stretch implies that the agent needs to wait at the task location for w_i^x time units. Fig. 4 shows a simple example with the use of DFC and the allowable operations. Fig. 4(a) shows a scenario where DFC is violated while Fig. 4(b) show the deadlock resolution by shifting the ATW of the second agent upwards (i.e., agent 2 will need to wait at the entrance of the aisle until agent 1 enters).



(a) Pre-adjust (b) After adjusting

Fig. 4. ATW Modification

The ATW representation is used to generate deadlock-free solutions for all possible assignment and coordination decisions. Moreover, w_i^e and w_i^x of (1) are derived from the final ATW representation which are used to calculate the objective function. Next, we formulate and solve the task assignment and coordination problems.

C. Assignment and Coordination Problem

Let us now proceed with the assignment problem. Let $x_{i,j}$ be a binary variable that takes the value 1 if agent i is assigned to task j, $i, j = 1, \cdots, n$ and 0 otherwise, then the assignment problem can be formulated as follows:

$$\min_{x_{i,j}} \max_i T_i$$

s.t.

$$\sum_{i=1}^{n} x_{i,j} = 1 \text{ and } \sum_{j=1}^{n} x_{i,j} = 1$$

Due to the non-linear objective function, this problem is not the typical assignment problem which can be solved using linear programming. Assuming $w_i^e = 0$ and $w_i^x = 0$ the objective function omits possible delays added by DFC. The assignment problem now falls under the *Bottleneck Assignment Problem* [16][17]. This problem can also be solved very efficiently using the following lemma.

Lemma 3: Assuming $\langle t^e \rangle$ is sorted such that $t_i^e < t_{i+1}^e$ and $\langle \tau_v \rangle$ is sorted such that $\tau_j > \tau_{j+1}$, then the solution to the assignment problem is $x_{i,i} = 1$ for all $i = 1, \cdots, n$ and $x_{i,j} = 0$ for all $i \neq j$, $i, j = 1, \cdots, n$.

In other words, the agent that is located closest to the entrance of the container aisle is assigned to the task that is located farther away from the entrance (deeper into *R2*).

For the proof which is omitted due to space limitations we assume that all tasks are located in a small set of aisles all next to each other. Also, note that the proof is based on an induction argument on the number of agents n.

Next, we need to solve the coordination problem, and decide the waiting times of each agent w_i^e (waiting at the entrance of the aisle) or w_i^x (waiting at the task location) necessary to prevent any deadlocks. Given the solution to the assignment problem, one can conjecture that the solution to the ordering problem would be $1, 2, \cdots, n$, i.e., the agents closest to the aisle will enter first, the second closest second and so on. To determine the agent coordination and the deadlock resolution policy one can use the ATWs, i.e., ATWs can be used to determine the waiting times w_i^e and w_i^x such that a deadlock-free solution is obtained.

Using the results of Lemmas 3 and 2 together with ATW allowable operations (i) and (ii), one can generate an efficient algorithm for deriving the optimal assignment and coordination solution. Given the n tasks and the initial positions of the n agents in a cycle, the algorithm consists of the following steps.

Algorithm 1:

1) Tasks are assigned to agents using Lemma 3.
2) Construct the initial ATWs for all agents.
3) Check the DFC. If it is not violated, then the obtained solution is also deadlock free and thus the optimal solution has been found.
4) If a deadlock occurs, then one can start with the agent that is involved in a conflict and has the smallest τ_v and apply the ATW allowable rules iteratively until all conflicts are resolved.

The above algorithm, though more efficient than a brute force enumeration of all possible solutions in the solution space, in its worse case, it is still of exponential complexity. To reduce the complexity even further we used a heuristic, referred to as **Algorithm 2**. Algorithm 2 is similar to Algorithm 1 except in step 4, rather than considering both ATW allowable operations we consider only operation (ii). In other words, again we start from the agent with the smallest τ_v that is involved in a conflict and simply stretch the ATW of its immediate predecessor just enough as to eliminate the conflict. This heuristic reduces the complexity to $O(n)$ and performs very well as indicated by our simulation results.

V. Preliminary Performance Evaluation

A. Exhaustive Simulation

In this paragraph, we use exhaustive simulation to derive the complete set of solution. From that set, the policies that generated the best performance are identified. We compare the exhaustively derived policies and solutions, against what we derived in the previous section. Simulations were done using MATLAB. Multiple random scenarios were generated, using $n = 2, 3, 4, 5$ agents. Each scenario was solved using exhaustive search. The same scenarios were also solved using our Algorithm 2 for post-processing comparison.

We analyzed the results obtained and classified the results. The analysis showed that the best *assignment* happens when the agent closest to the entrance of the aisle is assigned to the container located farther in the aisle. The same applies for the rest of the agents, until finally the farthest from the entrance agent is assigned the closest container. The best *ordering* is the one where agents enter the aisle according to how close they are to the entrance. That is, the closest agent will enter first, the second closest will enter second and so on. Finally, in the overwhelming majority of cases investigated, the best *entering mode* was the one where the agents enter the aisle in series, that is, no agent waits at the entrance for any other agent to exit and therefore all agents are in the aisle simultaneously as much as possible. We define this best performing combination as $(V, R, M)^*$. This is precisely our algorithm 2 presented in the previous section. In other words, given a problem with a set of agents A and a set of tasks S, it can be solved in real-time by applying the $(V, R, M)^*$ and then running algorithm 2 to resolve the conflicts.

B. Comparison with Greedy Approach

Having derived $(V, R, M)^*$, we further test our method's performance against a greedy algorithm. To do that, a set of scenarios with random container positions were generated. Each scenario was solved with a greedy algorithm and then with our suggested algorithm. The greedy algorithm places ATWs incrementally starting from $i = 1$. First it tries to place its ATW as soon as possible. If a conflict is detected it tries the next available time slot. When no conflicts are detected, it moves to the next i. This procedure is repeated until all ATWs are placed.

To display the scalability of our algorithm, we repeated the experiment for up to $n = 40$. Fig. 5 shows the results. As displayed, the time required to finish a scenario is increased at a much higher rate using the greedy algorithm. It is important to note that both algorithms are of the same complexity $O(n)$ therefore take approximately the same time to finish.

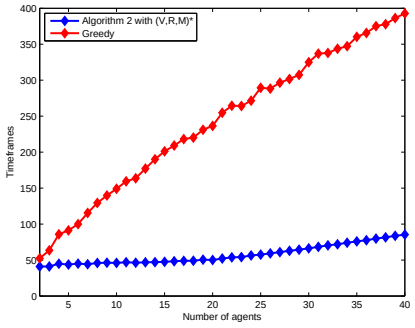


Fig. 5. Performance comparison with a greedy algorithm

C. Comparison with other methods

In this section we evaluate the performance of other methods operating on the same topology. Methods have been proposed that solve the problem on-line using *Time Windows* methodology. Because the *Time Windows* method

is computationally intensive, the solution set had to be reduced to allow real-time performance. This was achieved by applying priorities, either to agents or containers. This however, makes the method myopic. Unfortunately such methods cannot be used in a warehouse that lacks of multiple paths, like the one considered in this paper, because the myopic characteristic can cause significant inefficiencies. In many cases, the routing is the best possible combination of time windows that **remained available** from previous vehicles and this does not guarantee global performance of the solution.

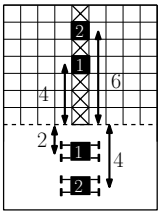


Fig. 6.

To demonstrate this we consider a simple, $n = 2$ example shown in Fig. 6. Task s_1 has higher priority than s_2 and therefore it will be routed first. It is trivial to derive that agent 1 will finish this task quicker than agent 2 and therefore it is assigned to that task. The time-window routing of agent 1 will finish at time $t = 10$. Then task s_2 should be routed. The remaining agent 2 is assigned to this task. When routing takes place, the time-window method will detect and resolve any head-on conflicts and will force agent 2 to wait for agent 1 to exit the aisle before entering, since only then, an available time-slot exists (that will not cause any conflicts). Using a dynamic time-window method task s_1 will finish at time $t = 10$ and task s_2 will finish at $t = 21$ (this is referred to solution A). An alternative, solution B, which is given by our approach, is s_1 assigned to agent 2 and s_2 to agent 1. Using this assignment and routing, s_1 will finish at time $t = 12$ and s_2 will finish at $t = 14$. This solution was generated using $(V, R, M)^*$ and our algorithm 2. Comparing the two solutions we can see that solution A delivers the first task as soon as possible but extends the delivery of the second task (and ultimately the global finish time) significantly. On the other hand our approach chooses a solution that slightly increases the time of the first task by 2 but decreases the global time by 7. This shows the myopic characteristic of some methods, described earlier. For problems with larger n the myopic approach can cause a significant increase in the completion time of the global solution.

VI. Conclusion and Future Work

In this paper we studied a hyperopic strategy for cargo transportation in special topology environment. Firstly the problem is broken down into two coupled subproblems of *assignment* and *coordination*. Global completion time is used as performance evaluation. To study this problem we introduced the *ATW* transformation. The feasibility of a path is evaluated with some simple condition rules and if any conflicts are detected then ATWs are modified. Following an exhaustive procedure, a specific combination of *assignment* and *coordination* was derived which always produces the optimal solution. This is also confirmed analytically. Based on this, we conclude that in a warehouse using the special topology under study, the agents can decide the assignment and routing without the need of an analytical approach. Because the algorithm is $O(n)$, it can run in real time making it a feasible solution for implementation on real AGVs.

We are planning on implementing this method on a robot testbed in our laboratory. This will guide us to adjust our method on real life conditions. Furthermore, we will develop an embedded algorithm for the operational control of the robots in real time, based on the work of this paper. Finally, we are interested in expanding this into a fully distributed method.

References

[1] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. pp. 80–91, 1959.

[2] G. Laporte, "Fifty years of vehicle routing," *Transportation Science*, vol. 43, no. 4, pp. 408–416, 2009.

[3] P. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Magazine*, vol. 29, no. 1, p. 9, 2008.

[4] L. Qiu, W. Hsu, S. Huang, and H. Wang, "Scheduling and routing algorithms for AGVs: a survey," *International Journal of Production Research*, vol. 40, no. 3, pp. 745–760, 2002.

[5] S. Daniels, *Real-time conflict resolution in automated guided vehicle scheduling*. PhD thesis, PhD Dissertation: Dept. of Industrial Eng., Penn. State University, USA, 1988.

[6] C. Kim and J. Tanchoco, "Conflict-free shortest-time bidirectional AGV routeing," *International Journal of Production Research*, vol. 29, no. 12, pp. 2377–2391, 1991.

[7] C. Kim and J. Tanchoco, "Operational control of a bidirectional automated guided vehicle system," *International Journal of Production Research*, vol. 31, no. 9, pp. 2123–2138, 1993.

[8] R. Mohring, E. Kohler, E. Gawrilow, and B. Stenzel, "Conflict-free real-time AGV routing," in *Operations Research Proceedings 2004*, pp. 18–24, Springer, 2005.

[9] N. Smolic-Rocak, S. Bogdan, Z. Kovacic, and T. Petrovic, "Time Windows Based Dynamic Routing in Multi-AGV Systems," *Automation Science and Engineering, IEEE Transactions on*, vol. 7, no. 1, pp. 151–155, 2009.

[10] F. Taghaboni-Dutta and J. Tanchoco, "Comparison of dynamic routeing techniques for automated guided vehicle system," *International Journal of Production Research*, vol. 33, no. 10, pp. 2653–2669, 1995.

[11] M. Pavone, S. Smith, F. Bullo, and E. Frazzoli, "Dynamic multi-vehicle routing with multiple classes of demands," in *American Control Conference, ACC 2009*, (St. Louis, MO, USA), pp. 604–609, Jun. 2009.

[12] Z. A. Banaszak and B. H. Krogh, "Deadlock avoidance in flexible manufacturing systems with concurrently competing process flow," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 6, pp. 724–734, 1990.

[13] N. Q. Wu and M. C. Zhou, "Modeling and deadlock avoidance of automated manufacturing systems with multiple automated guided vehicles," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 35, no. 6, pp. 1193–1202, 2005.

[14] C. O. Kim and S. S. Kim, "An efficient real-time deadlock-free control algorithm for automated manufacturing systems," *International Journal of Production Research*, vol. 35, no. 6, pp. 1545–1560, 1997.

[15] I. Vis, "Survey of research in the design and control of automated guided vehicle systems," *European Journal of Operational Research*, vol. 170, no. 3, pp. 677–709, 2006.

[16] U. Derigs and U. Zimmermann, "An augmenting path method for solving linear bottleneck transportation problems," *Computing*, vol. 22, no. 1, pp. 1–15, 1979.

[17] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment problems*. Society for Industrial Mathematics, 2009.