2018 Second IEEE International Conference on Robotic Computing

Optimal Trajectory and Schedule Planning for Autonomous Guided Vehicles in Flexible Manufacturing System

Atefeh Mahdavi
School of Computer Science
Florida Institute of Technology
Melbourne, Florida 32901
Email: amahdavi2015@my.fit.edu

Marco Carvalho
School of Computer Science
Florida Institute of Technology
Melbourne, Florida 32901
Email: mcarvalho@cs.fit.edu

***Abstract*—This paper proposed an algorithm to solve major challenges in the domain of autonomous transportation systems. These challenges are trajectory planning and collision avoidance. Generally, in a shared infrastructure where several agents aim to use limited capacity resources, finding a set of optimal and conflict-free paths for each single agent is the most critical part. In such a dynamic environment where continual planning and scheduling are required, we have adopted a coupled (centralized) technique and face the problem by breaking it into two distinct phases: path planning and collision avoidance. In the first phase, the lowest cost path is planned for each agent by a modified version of Dijkstra algorithm. Following that, in the second phase, the head-on collisions can be foreseen and avoided by detecting the agents' common segments of the path choices. Finally, based on the capacity of each resource and the feasibility of each path, the task assignment and scheduling algorithm allocates different tasks between agents.**

I. Introduction

Nowadays, autonomous guided vehicles (AGVs) are playing an ever-increasing role in different aspects from indoor industry applications to outdoor autonomous transit systems. Examples include autonomous aerial vehicles assisting victims in many search and rescue scenarios or robots capable of lifting and carrying loads and preparing orders for costumers in automated warehouse delivery systems [1–4]. Multi-robot systems (MRS) have laid the foundation for a broad research in robotics application domains [5]. In all these challenging frameworks a set of cooperative or self-interested robots or a mixture of them increase the global performance of the system. Robots in our framework are self-interested, in the sense that they accomplish their tasks without any dependency on the other robots, in contrast to cooperative ones, which have to coordinate activities toward gaining a system goal [6]. Time and financial efficacy are the major benefits of these applications compared to conventional methods using human resources; however, implementation of an algorithm for the corresponding system which minimizes the time and cost of this process and copes with any unexpected incident is the most crucial task.

According to the proposed taxonomy by Parker, multi-robot path planning can be classified into coupled and decoupled categories [7]. Some examples of multi-robot motion planners based on this classification include [8–10]. A coupled approach uses global information, and plans the paths for all robots treated as a single entity. Alternatively, decoupled planning will compute individual paths for each robot, followed by methods for handling any potential conflicts between the paths as they arise. Notwithstanding some advantages of being complete and optimal, the computational time of coupled methods increases exponentially as the number of robots increases. On the other hand, decoupled approaches are fast, but at the cost of losing completeness and optimality [11], [12]. In this research, since the size of problem is small, consisting of three agents, our method of choice is coupled.

From the priority point of view, all agents in the proposed framework have the same priority on the resources, unless the collision exists. In this situation, the priority of moving is taken into account and each agent moves based on its assigned priority in a sequential order. In comparison with prioritized planning [13] in which each robot is assigned with a unique priority. Then for each selected robot in a decreasing order of priority a path is planned, avoiding conflicts with the plans of higher priority robots. Context-aware routing introduced by Ter Mors is an example of prioritized planning approach based on a search through the graph of free time windows on the resources [14]. A free time window is an interval that a new agent can plan around existing reservations of other agents on a resource without making any conflict.

This research is inspired by conditions in an industrial setting in which fleets of vehicles are fixed to tracks and have to interact with each other in a self-interested way to accomplish different tasks as fast as possible. The reported work is able to find the shortest path in multi-agent systems using model switches. Switches are able to direct trains toward straight or diverging paths. Because of this property of switches, running regular version of Dijkstra algorithm on the graph of such systems generates some infeasible paths that can not be taken by trains. The proposed version of Dijkstra algorithm addresses this problem by defining virtual nodes and adding an additional characteristic to each node of the graph

0-7695-6370-8/18/31.00 ©2018 IEEE
DOI 10.1109/IRC.2018.00034

(see Section IV). Once each agent is assigned with a desired path, an online collision detection and collision avoidance algorithm which is described in Section V, avoids all collisions and deadlocks. The salient feature of this algorithm is finding common nodes of all generated paths continuously to detect the collision points. Once the location of collision is detected, the algorithm assigns the best actions of waiting and moving to each train before the place of collision. The criteria for this action selection is based on the minimum travel time.

The proposed algorithm has been evaluated for different scenarios both in simulated experiments and on model trains. As the outcome of this paper, a set of tasks will be assigned to the agents. Based on the origin and the destination of each task, the shortest path will be selected for conducting each of the tasks while head-on collision between the agents are avoided. The rest of this paper organized as follows. First, we discuss related work in Section II. After a brief overview of experimental setup details in Section III, we outline the modified version of Dijkstra algorithm in Section IV. The planner algorithm is then described in Section V. Finally, in Section VI we draw our conclusion and possible future work.

II. Related Work

Kiva warehouse management system introduced in [15] is the first large-scale autonomous robot system in which small, autonomous robots transport movable storage shelves from storage locations to stations. This system uses a standard implementation of A* algorithm to plan paths and prioritizes the goals by maintaining a list of high level goals. Another relevant work [16] presents a decoupled algorithm for coordinating robotic agents in large-scale industrial environments based on a sharing resource protocol and replanning strategy. In this approach, each agent determines its most convenient path as a primary path towards the desired destination on a graph. Following that, in order to reduce waiting time to access a shared node, a new path planning is run for a secondary path which is generated by removing the first node of the primary path along with all incoming arcs.

In such dynamic environments, path planning is more complex when single-stage tasks move to multi-stage ones involving many iterations. There are several approaches to solve single-stage path planning; However, finding the efficient solutions of multi-stage routing specially in a warehouse AGV domain, where the number of agents have a sequence of destinations and should collaboratively plan for conflict-free paths is in demand. The Delegate MAS approach for multi-agent route planning (MARP) is proposed for both single-stage and multi-stage routing [17]. In this work which is inspired by the foraging behavior in ant colonies, each agent continuously drops the information of its plan to the relevant part of the infrastructure. This information can be collected and used by other agents to make their plans later. The plan step of each agent consists of a resource, the entry time and exit time on that resource. In this decentralized version of planning, each resource can be reserved by an agent if the corresponding reservation is consistent with existing schedule. Each agent explores alternative routes from its current location to its destination. Then it evaluates the quality of each route based on existing schedules and reserves the most preferable one. A cooperative A* algorithm introduced in [18] utilizes a three dimensional space-time reservation table. This table represents the knowledge of all agents' planned routes. Each agent performs searches through this table. After calculating a route, each cell of the grid along the route is marked as impassable for duration of intersection and consequently it prevents others from planning a colliding route. There are other strategies that avoid deadlocks by using a path re-planning or a master-slave approach [19].

III. Experimental Setup Details

A. Physical Testbed

We evaluate this experiment under realistic conditions using a prototype of transportation system with elements of real world applications such as model trains, tracks, switches, etc. Figure 1 shows a view of this infrastructure in which the tracks are connected to each other creating multiple paths. Switches enable the trains to be guided from one track to another. Different imaginary stations defined as destinations in multi-stage routing plan of trains.

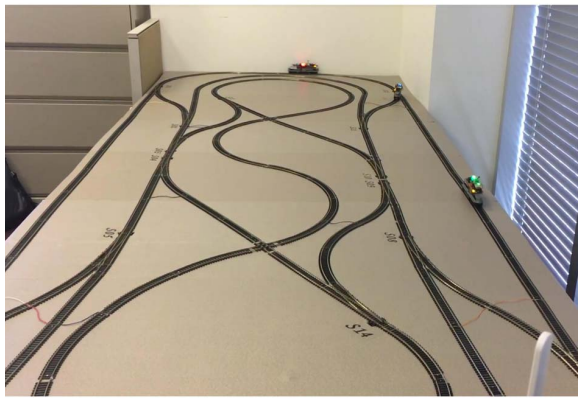

Fig. 1. A view of infrastructure showing model trains and resources (tracks, switches and stations)

B. Communication

An ad-hoc network is utilized to establish a wireless connection between server and elements of prototype setup. Programming of motions and sequences is typically sent to train and switch controllers over this network. Through this connection, each part of the system is aware of the others. The server accepts the requests from client using a Java API over TCP connection, then starts receiving and sending the commands of movement and state changing to trains and switches respectively. Furthermore, this system is capable of real time processing and accurate tracking of trains by applying image processing and computer vision techniques and provides us with a global view of environment.

IV. Framework and Modification of Dijkstra Algorithm

Figure 2 represents a comprehensive bidirectional graph $G = (V, E)$ associated with working environment, where V is a set of nodes (switches and stations) and E stands for a set of edges (tracks). Each resource of switch and station has unit capacity; in other words, it can be reserved just by one train at a time. Each edge is associated with two nodes, one at either end. The algorithm in this paper is based on a search through the proposed graph of all resources. The first step of the planner algorithm is finding the shortest path which minimizes the transportation time. There are different graph search algorithms like A* [20], Dijkstra [21] and D* [22] to find the shortest path for single autonomous vehicle. Our algorithm of choice is Dijkstra, which is improved for the special graph of infrastructure. Based on direction of a train, the design of switch in our prototype model, Figure 3, does not provide choice of path to all its points. For example, in Figure 3, if the train enters from point 1, it can be directed toward the straight path or the diverging path depending on the state of switch. If the train enters from point 2 or point 3, it can only leave node from point 1, otherwise it is going be derailed. However, Dijkstra algorithm is developed for regular graphs without any limitation. This causes generating some infeasible paths that cannot be traversed by the train. Consequently, if the regular Dijkstra algorithm is run on the represented graph in Figure 2; for instance, from station 19 to 14, it meets the unfavorable condition described above at switches 4 and 5 and generates the impossible path of 19-4-5-14 shown in bold line to be taken.

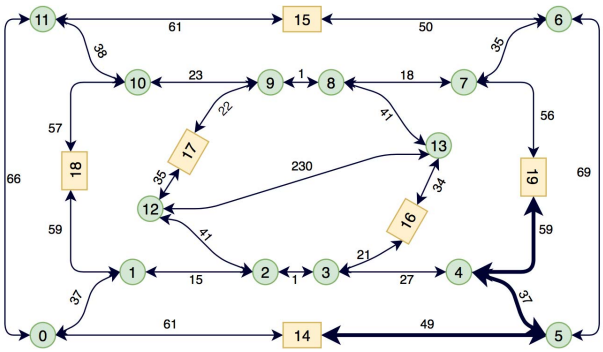

Fig. 2. Initial graph of system. Yellow rectangles and green squares represent stations and switches respectively.

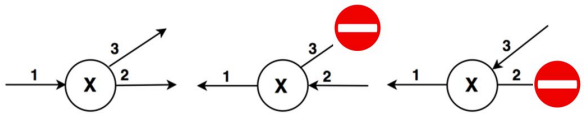

Fig. 3. Switch states and possible paths that train can take after entering the node from specific point.

Tackling the above-mentioned problem calls for modifications on the Dijkstra algorithm. As a brief introduction to Dijkstra, in the beginning of the algorithm, it sets the initial node as current with zero assigned distance value. Then, it marks all other nodes unvisited and creates a set of them called the unvisited set with infinity assigned as their initial distance values. For the current node, the algorithm picks all its neighbors and calculates their distances. After that, the newly calculated distance is compared to the current assigned value and the smaller one is selected and assigned. After considering all of the neighbors, the current node marked as visited and removed from unvisited set. The unvisited node with the smallest distance is selected for the next iteration. This process continues until the destination node has been marked visited.

This algorithm has no restriction on selecting the neighbors of the chosen node for expansion, which means all unsettled connected nodes to the popped one can be selected as the neighbors. However, the proposed version can ignore some neighbors related to infeasible paths in the process of taking the adjacent nodes. For the detection of these infeasible paths, we append an additional characteristic to each node of the graph by the definition of Entry and Exit pins. When a train is running on a track, it approaches the Entry pin of a switch or a station and leaves that node from Exit pin. Compared to stations with merely one Entry and Exit pins, switches have one Entry pin and the choices of two Exit pins. In some cases, based on the trains' direction, the algorithm should block one of these Exit pins (see Figure 3). As a rule, the fork side of all switches are labeled with the same values i. e. 2, 3. Allocation of these values to either side of each fork can be done in arbitrary order. The single side of the switch is labeled with value 1 as demonstrated in Figure 3. From notations in Table I, given switch SW_i, en and ex can either be an element of set $S = \{1, 2, 3\}$. Consequently, the pair (SWi_{en}, SWi_{ex}) results in set $R = \{(1,3), (1,2), (2,1), (2,3), (3,1), (3,2)\}$ and the algorithm can detect and eliminate impossible traverse when the pair (SWi_{en}, SWi_{ex}) results in $(2,3)$ or $(3,2)$. Although stations do not cause any restrictions in our track design, they are also labeled, as shown in Table I, and complied by new modification to make the algorithm general for class of node consisting both stations and switches.

Applying the above-mentioned adjustment is not the only modification to Dijkstra algorithm. In some cases, in the step of processing the unsettled nodes, the algorithm is stuck between two inaccessible nodes where one of them exits in an identified impossible path and the other reached by a shorter distance. For example, in Figure 4, when we run the algorithm from source 18 to destination 15, it is trapped in switch 6 with no way to get to the destination. In this case, in the list of unsettled nodes, node 15 is in infeasible path and node 5 is reached by node 4 with a shorter distance from the other underdevelopment path represented as bold dashed line. We introduce the second robust amendment by integrating virtual nodes and edges to the configuration of the initial graph. Figure 5 depicts a part of these transformations related to the problem mentioned in Figure 4.

TABLE I
SYMBOLS USED IN THIS PAPER

Symbols	Definition
SW	Switches
ST	Stations
R	Tracks
T	Trains
AT	Arrival time to end point of common path
CP	Common path
CPl_{s1}	Entry point 1 of common path l
CPl_{s2}	Entry point 2 of common path l
i	Switch index
j	Station index
k	Track index
l	Common path index
a	Train index
N_{TK}	Number of Tracks
N_s	Number of Stations
N_T	Number of Trains
N_{sw}	Number of Switches
en	Entry pin
ex	Exit pin
SWi_{en}	Entry pin of switch i
SWi_{ex}	Exit pin of switch i
STj_{en}	Entry pin of station j
STj_{ex}	Exit pin of station j

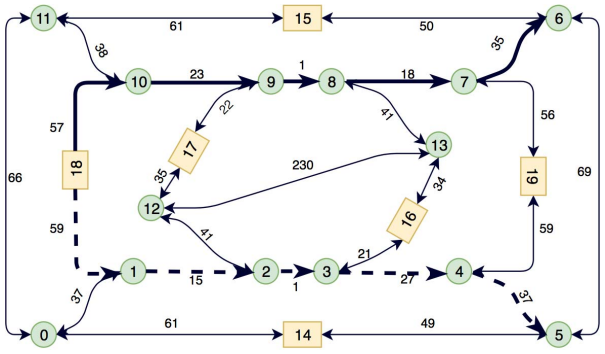

Fig. 4. The possible and shortest path from source 18 to 15.

These virtual components provide each switch with the second chance of being visited from another path by a different predecessor. Recall the previous scenario, when switch 5 selected for expansion, if there is still a possibility of visiting settled switch 6 by its virtual node V_6, it can reach the destination 15. After that the artificial path will be mapped to initial nodes and edges for implementation purpose.

V. PLANNING ALGORITHM

Given sequence of destinations $D = \{(d_1, d_2, ..., d_j)\}$ where d_j is a station and defined as an element of set $S = \{ST_j \mid j \in [1, N_s]\}$, an agent plan defined as a sequence of $PT_a = \{(d_1, t_1), ..., (d_n, t_n)\}$ of n (destination,interval) pairs. Each pair represents information about pick-up or delivery locations and the corresponding intervals to stop on them. The set of movements modes for each train defined as $M = \{forward, backward, stop\}$ where $M(T_a)$ is called the mode of train T_a. In the proposed coupled method, we split the problem into two phases: at the first phase, the

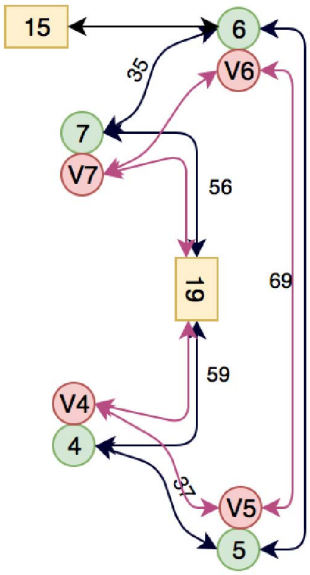

Fig. 5. Modified graph with virtual nodes and edges distinguished with the pink color.

shortest path is generated by modified version of Dijkstra algorithm for each agent based on the current origin and the chosen destination from the list of tasks. We assume all agents have the same priorities on resources. At the second phase, when a path is planned for each agent, the conflict resolution algorithm ensures that the collection of paths is collision free. It essentially checks if the transversion of a new path crashes with the transversion of all other generated paths to find the collision points. Each path consists of several tracks from set $TR = \{R_k \mid k \in [i, N_{TK}]\}$. The conflict resolution algorithm searches through a list of all paths generated so far to discover common tracks between the paths of each two trains. This list is updated continuously when a new train enters to the infrastructure and a new path is generated. In a case of finding two or more connected common tracks, the algorithm generates a common path of them. Each common path is distinguished by a pair of $\{(CPl_{s1}, CPl_{s2}) \mid CPl_{s1} \wedge CPl_{s2} \in (SW \cup ST)\}$ (see Table I). It can be a combination of tracks, switches and stations or each of these elements individually. In case of individual element of switch or station, CPl_{s1} and CPl_{s2} are overlapped. For example, in Figure 6, trains T_1 and T_2 explore and select two paths represented in bold solid blue and bold dashed red lines respectively. Both start from the same origin with the same speed and therefore may collide at the common path of (SW_{10}, SW_7). Catching-up the common paths does not necessarily guarantee the existence of collision. In the above-mentioned scenario if trains start at different times, in such a way that T_1 travels all the way of common path before T_2 meets the (SW_{10}), there is going be no collision. Therefore, we propose the second key factor in detection of conflict that is arrival time at which a train is scheduled to reach at the CPl_{s1} or CPl_{s2}. For this purpose, we apply the equation of $x = vt$ where v is the speed, x is the path length and t is time.

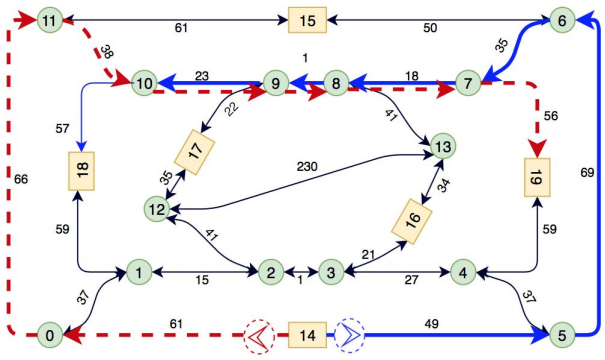

Fig. 6. Two trains try to reach the same tracks at the same time.

With the assumption of the constant same speed for each train in our framework, there is a liner relation between the length of each track and the travel time. So, we can say, the arrival time of each train can be calculated by adding all the head-on tracks length. Following the above stages performed, the planner faces two states:

1) The common path does not include station. In this case the following relations governs the planner scheduling:

if $AT_a > AT_{a'}$ **then**
$M(T_a) \rightarrow stop, M(T_{a'}) \rightarrow forward$
else if $AT_a < AT_{a'}$ **then**
$M(T_a) \rightarrow forward, M(T_{a'}) \rightarrow stop$
else if $AT_a = AT_{a'}$ **then**
$\begin{cases} M(T_a) \rightarrow stop, M(T_{a'}) \rightarrow forward \\ \qquad\qquad or \\ M(T_a) \rightarrow forward, M(T_{a'}) \rightarrow stop \end{cases}$
end if

Note that *stop* command is executed before start node of common path for respective train. In the case that common path belongs to more than two trains, the priority of the $forward$ command is given to the trains based on the AT_a array of trains in ascending order.

2) The common path includes station. In this case, the interval time on the station should be taken into account. The priority of the $forward$ command is given to the trains based on the $[AT_a + intervaltime]$ array of trains in ascending order. However, the other trains stop at resources until the prior train exits the station or take the next task, if available.
In described example in Figure 6, there is no station in common path and $AT_1 = AT_2$; therefore, the planner randomly sends *stop* command to T_1 to stop before SW_7 and lets the T_2 to travel the common path. Once T_2 passes the SW_{10}, a $forward$ command is sent to T_1 to perform rest of its tasks.

The efficiency of this algorithm is defined by how quickly the agents complete their tasks, or by the number of tasks accomplished in a fixed amount of time. In order to validate the performance of our algorithm, we use a simulation made in unity (Figure 7) and measure $turnaround\ travel\ time$:

$$\overline{T} = T_{out}(T_{a'}) - T_{in}(T_a)$$

Where $T_{in}(T_a)$ is the time when the first train starts from the initial station and $T_{out}(T_{a'})$ is the time when the last train arrives at its last destination after reaching all its destinations. Tasks are assigned to each agent specifically in a sequential order. For example, the measured value of $\overline{T}$ for three agents to complete their tasks is $141\ seconds$ where three trains start from the same initial state (station 14) shown in Figure 6 with the same speed based on the following plans:

$$PT_1 = \{(d_{18}, 10), (d_{16}, 8)\}$$
$$PT_2 = \{(d_{19}, 5), (d_{15}, 6)\}$$
$$PT_3 = \{(d_{15}, 7), (d_{17}, 9)\}$$

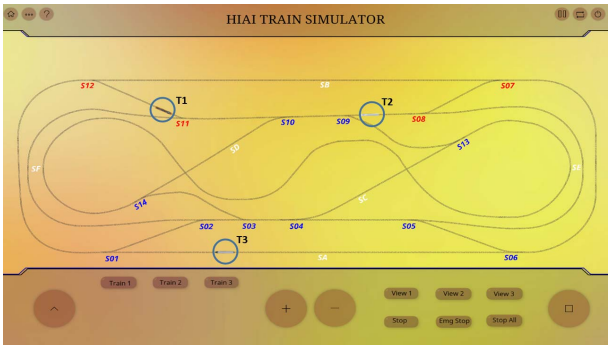

Fig. 7. Simulation with 3 agents.

VI. Conclusion

This paper has presented the approach to coordinate model trains, moving within a dynamic industrial environment. Each agent plans its own trajectory based on an improved Dijkstra algorithm using the graph information. Applying switches makes the graph of our infrastructure special in such a way that running the regular Dijkstra algorithm produces some impossible paths. Modifications to Dijkstra algorithm include adding an additional feature to each node of the graph along with using artificial nodes and arcs. Moreover, to ensure the safety of the system, a collision avoidance algorithm has been applied. This algorithm rests on the assumption of constant speed. In future we will take into account variable speed and conduct more polices and scenarios. Additionally, we can deal with this problem as an optimization problem and apply local search algorithms in which finding the best solution according to an heuristic cost function is the objective. Expansion of infrastructure and simulation in terms of complexity and number of agents is the next phase of this study. More numerical comparison and verification against other algorithms will be presented in that study.

References

[1] C. K. Verginis, Z. Xu, and D. V. Dimarogonas, "Decentralized motion planning with collision avoidance for a team of uavs under high level goals," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 781–787.

[2] M. Bernard, K. Kondak, I. Maza, and A. Ollero, "Autonomous transportation and deployment with aerial robots for search and rescue missions," *Journal of Field Robotics*, vol. 28, no. 6, pp. 914–931, 2011.

[3] D. Herrero-Perez and H. Matinez-Barbera, "Decentralized coordination of autonomous agvs in flexible manufacturing systems," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, 2008, pp. 3674–3679.

[4] R. Fierro, A. Das, J. Spletzer, J. Esposito, V. Kumar, J. P. Ostrowski, G. Pappas, C. J. Taylor, Y. Hur, R. Alur *et al.*, "A framework and architecture for multi-robot coordination," *The International Journal of Robotics Research*, vol. 21, no. 10-11, pp. 977–995, 2002.

[5] T. Arai, E. Pagello, and L. E. Parker, "Advances in multi-robot systems," *IEEE Transactions on robotics and automation*, vol. 18, no. 5, pp. 655–661, 2002.

[6] V. R. Lesser, "Cooperative multiagent systems: A personal view of the state of the art," *IEEE Transactions on knowledge and data engineering*, vol. 11, no. 1, pp. 133–142, 1999.

[7] L. E. Parker, "Path planning and motion coordination in multiple mobile robot teams," *Encyclopedia of complexity and system science*, pp. 5783–5800, 2009.

[8] T. Standley and R. Korf, "Complete algorithms for cooperative pathfinding problems," in *IJCAI*, 2011, pp. 668–673.

[9] M. Cirillo, F. Pecora, H. Andreasson, T. Uras, and S. Koenig, "Integrated motion planning and coordination for industrial vehicles." in *ICAPS*. Portsmouth, NH, 2014, pp. 463–471.

[10] I. Draganjac, D. Miklić, Z. Kovačić, G. Vasiljević, and S. Bogdan, "Decentralized control of multi-agv systems in autonomous warehousing applications," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 4, pp. 1433–1447, 2016.

[11] J. van Den Berg, J. Snoeyink, M. C. Lin, and D. Manocha, "Centralized path planning for multiple robots: Optimal decoupling into sequential plans." in *Robotics: Science and systems*, vol. 2, no. 2.5, 2009, pp. 2–3.

[12] R. Luna and K. E. Bekris, "Efficient and complete centralized multi-robot path planning," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 3268–3275.

[13] J. P. Van Den Berg and M. H. Overmars, "Prioritized motion planning for multiple robots," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE, 2005, pp. 430–435.

[14] A. ter Mors, J. Zutt, and C. Witteveen, "Context-aware logistic routing and scheduling." in *ICAPS*, 2007, pp. 328–335.

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

[16] L. Cancemi, A. Fagiolini, and L. Pallottino, "Distributed multi-level motion planning for autonomous vehicles in large scale industrial environments," in *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*. IEEE, 2013, pp. 1–8.

[17] H. T. Dinh, R. van Lon, and T. Holvoet, "Multi-agent route planning using delegate mas," in *Workshop on Distributed and Multi-Agent Planning*, 2016, pp. 24–32.

[18] D. Silver, "Cooperative pathfinding." *AIIDE*, vol. 1, pp. 117–122, 2005.

[19] S. Yuta and S. Premvuti, "Coordinating autonomous and centralized decision making to achieve cooperative behaviors between multiple mobile robots," in *Intelligent Robots and Systems, 1992., Proceedings of the 1992 lEEE/RSJ International Conference on*, vol. 3. IEEE, 1992, pp. 1566–1574.

[20] R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of a," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 505–536, 1985.

[21] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[22] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*. IEEE, 1994, pp. 3310–3317.