Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 54-1 (2021) 98–103

Decentralized and prioritized algorithm for AGV fleet management

Tímea Tamási * **Tamás Kis** **

* *ELTE Eötvös Loránd University, Doctoral School of Mathematics, Budapest, Hungary; ELKH SZTAKI Institute for Computer Science and Control, 1111 Budapest, Kende u. 13-17. Hungary (e-mail: tamasi.timea@sztaki.hu).*

** *ELKH SZTAKI Institute for Computer Science and Control, 1111 Budapest, Kende u. 13-17. Hungary (e-mail: kis.tamas@sztaki.hu).*

Abstract: In this paper we propose an algorithm based on prioritization for coordinating autonomously guided vehicles (AGVs) in a grid/mesh topology, where the tasks arrive online into the system. The method combines a decentralized control mechanism based on prioritization with an optimization subroutine to resolve eventually occurring deadlock situations. The merits of the method are evaluated on a set of benchmark instances.

Keywords: autonomously guided vehicles, decentralized control, cooperative navigation, prioritized algorithm, optimization.

1. INTRODUCTION

Autonomously guided vehicles play an essential role in many transportation systems. They are advanced and complex, since there are many tactical issues that have to be taken into consideration, such as environmental factors (guide-path design, location of pickup and delivery points, number of vehicles, format of the tasks), planning factors (scheduling the tasks, assigning the tasks to the vehicles, route planning), and collision-, deadlock- and livelock-avoidance.

There are multiple approaches for modelling AGV systems. One aspect is the structure of the underlying network or map where the AGVs are moving. Industrial and warehouse environments often operate with free-ranging vehicles (Das et al. (2016), Draganjac et al. (2016)). A common practice amongst them is roadmap-generation to restrict the freedom of motion (Digani (2016), van den Berg and Overmars (2005)).

The most widely used network follows a graph-like structure (De Wilde et al. (2014), Fazlollahtabar et al. (2015), Kornhauser et al. (1984), Luna and Bekris (2011), Möhring et al. (2005), Ryan (2008), Wang and Botea (2011)). The vehicles can move along the edges of the map, the vertices are the decision points in the path planning of the AGVs. A more specific format is the grid/mesh network, vehicles can travel along the edges of a grid (Stenzel (2008)), or move between the cells (Małopolski (2018), Regele and Levi (2006), Wang and Botea (2011)).

Several articles use zone-control scheme to build a model for AGV-coordination. The traffic system is represented by a partially directed graph, vehicles travel between zones in the network, and each zone allows one vehicle at a time. Fanti (2002) presented a real-time controller for AGV coordination using path-validation and zone-validation algorithms, Roszkowska and Reveliotis (2008) analysed the liveness of these closed traffic-system, and Fanti et al. (2018) designed a decentralized algorithm to coordinate the AGVs along the network to avoid deadlocks and collisions.

Another approach to distinguish between AGV fleet coordination methods is whether they use centralized or decentralized control. Centralized algorithms can guarantee completeness and optimal solution, however they need full information of the network layout and the tasks. They are also often exponentially slow considering the number of AGVs.

Kornhauser et al. (1984) studied the pebble motion coordination in graphs, which can also be viewed as multi-agent pathfinding. They proved that for a biconnected graph of n nodes with $n-1$ pebbles it is possible to solve the puzzle, and gave an $O(n^3)$ lower and upper bound for the number of moves required. Luna and Bekris (2011) designed an algorithm using push and swap moves, De Wilde et al. (2014) revealed several problems with this solution, and proposed the push and rotate algorithm to correct it. Wang and Botea (2011) propose a multi-agent path planning (MAPP) algorithm that is complete for a class of problems (called slidable) with quadratic running time in the network size and the number of vehicles. Centralized approaches can also use (mixed) integer programs to describe AGV coordinating problems, or classic vehicle routing and scheduling algorithms (Daniels (1988), Fazlollahtabar et al. (2015), Zheng et al. (2014)).

⋆ EFOP-3.6.3-VEKOP-16-2017-00001: Talent Management in Autonomous Vehicle Control Technologies - The Project is supported by the Hungarian Government and co-financed by the European Social Fund. This research was also supported by the ED 18-2-2018-0006 grant on "Research on prime exploitation of the potential provided by the industrial digitalisation", and the NKFIA 129178 grant.

10.1016/j.ifacol.2021.08.155

Decentralized ways are much faster and better scalable than centralized solutions, however they often lack completeness. A commonly used approach is prioritized planning first proposed by van den Berg and Overmars (2005), where the vehicles get an ordering, and the AGVs with lower priority should take into consideration the higher ranked vehicles. Considering the Shortest Path Problem with Time-Windows (SPPTW), Möhring et al. (2005) gave a generalized arc-based Dijkstra algorithm with labels and dominance test. Stenzel (2008) prioritized the requests and used a greedy dynamic routing algorithm with time-window adjustments.

Regele and Levi (2006) used a distributed cooperative path planning on local sections of the global environment map, where conflicts between the robots are solved by a heuristic adjustment of priority values. Ryan (2008) developed a simple centralized and a prioritized algorithm by decomposing the network into subgraphs and defining the operations between them. Velagapudi et al. (2010) created two distributed prioritized planner (reduced and sparse) to coordinate systems with large number of AGVs. Małopolski (2018) proposed an algorithm in a square topology by first creating a chain of elementary reservation for the AGVs in the map, and then using prioritization to serve the requests. This method combines centralized and decentralized approach since the reservations are known by every vehicle, but they use individual algorithm to move along their paths.

In the majority of research the set of tasks are given in advance, but it is also possible that assignments come in a sequential order and there is no information available before they enter the system. Yu and LaValle (2013) showed that even for fixed goal locations it is NP-complete to find feasible paths for the vehicles considering the minimum total arrival time, minimum makespan or minimum total distance as objective function.

We propose an algorithm that combines both centralized an decentralized approaches for online tasks, that are less studied in the literature. Vehicles prioritize themselves by considering the degree of the node they currently are on and whether they are carrying an item or not. Each vehicle has to take in consideration AGVs with higher priority when planning its next few steps, and if a conflicting situation occurs, it tries to find an alternative path by respecting the routes of the higher priority vehicles.

When this simple solution fails, an optimization subroutine is invoked to find disjoint paths locally for solving the conflict. We aim to find a solution for the online problem where the completion time of the tasks is minimal. We also try to minimize the difference between the actually travelled distance and the originally planned distance of the vehicles. The prioritization has the advantage of the decentralized methods such as speed and simplicity and in general it performs well even for larger number of AGVs. The completeness is guaranteed with the optimization algorithm, which ensures deadlock-free coordination. We run various tests to analyse the performance of the algorithm, which shows that even without the optimization subroutine, our method completes the tasks successfully in the majority of the cases. The costs stay low even when increasing the number of the vehicles, and it outperforms a highly used prioritization method proposed by van den Berg and Overmars (2005).

2. PROBLEM FORMULATION

In our problem, the network representing the map where the AGVs move is a grid graph $G = (V, E)$. The edges are bidirectional, the vehicles can move in both directions but they can only station in the nodes of the graph, i.e., once they start to move along an edge, they have to reach the other endpoint of it, and eventually stop there.

There are special nodes of the network, called *locations*, where the AGVs can pick up or drop off items. These can be located anywhere in the graph.

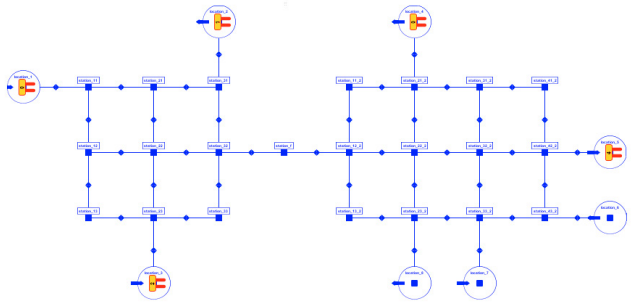

Fig. 1. An example of the network with 4 vehicles and 8 locations

The *tasks* to be performed by the AGV system are given by the following data. Each task has a *pickup location*, a *delivery location*, and an *arrival time*. A task becomes known at its arrival time, and it specifies that some *item* must be carried from the specified pickup location to the delivery location.

There are m identical vehicles in the system, each capable to transport one item at a time. They move with unit speed simultaneously, i.e., in every time unit, either they move from a node to a neighbouring node of the graph, or wait on their current node. There can be at most one AGV on a node at any time moment, and two AGVs cannot cross the same edge simultaneously.

There are three different states of the AGVs that indicate the cost of their movement or waiting:

- *free*: the AGV doesn't have a task, it is waiting for a new item to arrive in the system. In this case, travelling along an edge costs an unit, and the waiting has no cost.
- *collect*: the AGV has a task, and it is on its way to pick up the order. The moving and the waiting both has unit cost.
- *deliver*: the AGV has picked up the item, and it is travelling to the endpoint. The costs are the same as in the *collect* state.

Each task has the following possible states:

- *free*: there is no AGV already serving it.
- *in progress*: there is an AGV moving towards the start location, or already carrying the item,
- *finished*

The list of free tasks is denoted by F. A free AGV can choose its next task from F.

Our main goal is to create a conflict-free algorithm i.e. to avoid the following situations:

- *collision*: two vehicles reaches a node at the same time, or they start to move along an edge towards each other,
- *deadlock*: there are vehicles that can not move further on their paths without colliding each other,
- *livelock*: vehicles can move without collision or deadlock, but their movement is cyclic, and the tasks are not being served in the long run.

3. ALGORITHM

In this section we first propose an algorithm based on prioritization to coordinate the vehicles. Then we provide a sufficient condition for the prioritized algorithm to be complete, and we analyse the situations where deadlocks and livelocks are likely to occur. Lastly, we design an optimization subroutine for the deadlock situations that can not be resolved with the simple prioritization.

We divide the running of the algorithm into so called *synchronization rounds*: in each round, the vehicles run the same algorithm individually in parallel, while communicating with each other.

Throughout the algorithm, each vehicle has a dynamically changing edge-list representing the planned path denoted by P. The start node of the first edge is always the current location of the AGV. Waiting is realized with a loop edge.

3.1 Prioritized algorithm

A synchronization round consists of three main phases:

- planning phase together with task search
- synchronization phase
- moving or waiting

In the task search the vehicle chooses randomly from the free tasks available in the system, and plans a shortest route to its start location. The ShortestPath(v) method finds the shortest path between the current node of the AGV and the node v using Dijkstra's algorithm.

The planning phase deals with item pickup and drop-off, together with path planning and task search belonging to them. When a vehicle dropped of an item at an end location, but failed to find a new task, it will move away from the location in order to let other AGVs enter the location.

In the synchronization phase vehicles first get an initial priority number chosen randomly between 0 and 1. Then, extra values are added to this based on the vehicle's actual place in the graph and its state. There are two types of prioritization: prioritization A takes both aspects in consideration (except for nodes with degree one, since vehicles staying on those nodes have to get the biggest priority to be possible to leave from them). Prioritization B considers node degree only, state of the AGV can only decide between vehicles with the same node degree.

Algorithm 1 Planning phase

```
e := P(1).
P := P − e
if state = free then
    Task search
if (state = free or state = collect) and
we can pick up an item then
    Pick up the item.
    Let v_e the end location of the task.
    P := ShortestPath(v_e)
    state := deliver
if state = deliver and P = ∅ then
    Put down the item (we reached its end location)
    state := free
    Task search
    if we did not find any task then
        Let f ∈ δ(v), where v is the current position.
        P := f
```

After prioritization, each vehicle has to take in consideration all AGVs with higher priority when finding an alternative path. Synchronization has a parameter T that gives us how many steps/edges we have to consider for avoiding other vehicle paths. We create a list consisting of edge-lists called *RestrictedEdges* (RE): the t-th item ($1 \leq t \leq \min(T, |P|)$) gives the edges that can not be used in the t-th step of the path planning, since a vehicle with higher priority is already using it.

When looking for a shortest path avoiding edges of the RE list, we will use the ShortestPath(v, RE) method. This is a greedy algorithm using breadth search, where we also give time labels to the nodes. We expand the search while we still have restricted edges, and then we use Dijkstra's method from each node reached. We pick the shortest one from the obtained paths. If no route can be found with this condition, we return an empty path.

The (v, t) node-timestamp pair is restricted (and therefore edges starting at v are put in the RE list), if a vehicle with higher priority uses an edge starting from v in time t, where $t \in \{1, \ldots, T\}$. Since T is constant, building RE and finding the shortest path avoiding these will be polynomial.

If the vehicle didn't find an alternate path, but the node the AGV is currently on will be free in the next step, we will wait one time unit in order to avoid deadlock. If this is not possible, we couldn't resolve the conflict, and the algorithm stops with deadlock-detection.

During an avoiding manoeuvre another AGV can take an order a vehicle is currently collecting, therefore at the end of the synchronization round we also need to check if the task is still available, and if not, we perform another task search in the beginning of the next round.

3.2 Correctness

We would like to find conditions for the completeness of the simple prioritized algorithm, and detect the occurrence of deadlocks and livelocks. The algorithm we propose is trivially collision-free, since if the conflict can not be resolved, all the AGVs will stop and therefore they will

Algorithm 2 Synchronization

Prioritization
Send the priority to other vehicles, and collect their priority.
Calculate the rank from the priority list: *rank*.
$RE := \emptyset$
for $r \in \{1, \dots, rank - 1\}$ **do**
  Let $P_r :=$ the path of the AGV with rank r.
  for $j \in \{1, \dots, \max\{i, |P_r|\}\}$ **do**
    $e = (u, v) := P_r(j)$
    $RE(j) := RE(j) \cup \{e\}$
Check if our path is conflicting with paths of vehicle with higher rank.
if yes **then**
  if $state = free$ **then**
    Let $f \in \delta(v) - RE(1)$, where v is the current position.
    $P' := f$
  if $state = collect$ **or** $state = deliver$ **then**
    Let v_e be the end location of the last edge of P.
    $P' :=$ ShortestPath(v_e, RE)
  if $P' \neq \emptyset$ **then**
    $P := P'$
  else (no avoiding path is found)
    if the node v is free in the next step **then**
      $P(1) :=$ loop(v)
    else
      STOP, the conflict could not be resolved.

not collide. The following condition can be formed for the algorithm to be deadlock-free:

Claim 1. If for all $v \in V : d(v) \geq k$ or $d(v) = 1$, and the one-degree nodes don't have common neighbour, then k AGVs can travel deadlock-free, without unnecessary waiting.

Assuming that we have a grid graph on the plane, this can only assure the free movement of three vehicles on the network (by extending the 2-degree corner nodes with one-degree nodes in the grid), this is a strong condition, and the number of AGVs seems to be too small for the real-life industrial applications. However, we will see that in practice this simple algorithm still performs well, and deadlocks don't occur frequently.

3.3 Deadlock- and livelock-detection

We also want to detect the situations where deadlocks and livelocks can occur. For k conflicting vehicles we can conclude the following:

Claim 2. If for k conflicting AGVs we sort the vertices by their priority, and for the i-th node $d(v) \geq i$, then the conflict can resolved.

From this we can see that it is possible to resolve the conflict of four AGVs in the middle of the block, three in the edge of the block, and two in the corners. This can not be improved, an example for priority B is shown on Fig. 2 .

In reality it is hard to detect livelocks, since for short term it seems like 'everything is working', but in the long run the tasks are not being served. However, we can define

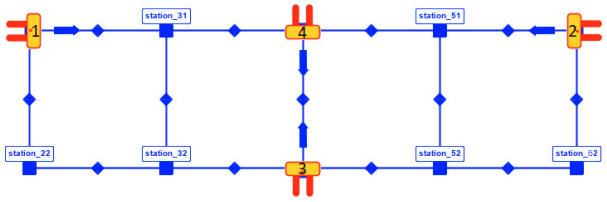

Fig. 2. An example of a deadlock with 4 vehicles: numbers denote the rank of the AGVs

a situation on symmetrical subgraph structure with two vehicles travelling towards each other, where livelocks are likely to happen:

Consider three vertices in G, u_1, u_2, u_3, for which $e = u_1u_2, f = u_2u_3 \in G$ and $d(u_1) = d(u_3) < d(u_2)$. Let AGV_1 and AGV_2 two AGVs with *collect* or *deliver* state, and assume the following:

- AGV_1 is currently on u_1, AGV_2 is on u_2
- e is the next edge for both AGV_1 and AGV_2
- neighbours of u_2 are already restricted by other vehicles

Since $d(u_1) < d(u_2)$, AGV_1 will get higher priority, therefore AGV_2 has to step back, and this can be done only through edge f. In the beginning of the next round, AGV_1 is on u_2, and AGV_2 is on u_3, the next step of their path is edge f. But then AGV_2 will have the bigger priority, because $d(u_3) < d(u_2)$, it will travel through f to u_2, and AGV_1 through e to u_1. We ended up in the same situation with the same conditions, therefore a livelock will occur.

From the results above, we could assume that our initial prioritized algorithm has strong theoretical bounds, and allows only a few vehicles to travel deadlock-free in a grid-like network. However, it has the following advantages: it is simple and quick, and in most of the cases can solve the tasks efficiently (see in Section 4).

In the next section we propose an optimization subroutine which deals with the problem when it was not possible to resolve the conflict between the AGVs with the simple prioritization.

3.4 Optimization subroutine

We have to handle the case where k AGVs are in a conflict that they couldn't resolve. We will use a centralized optimization method to find conflict-free paths for the vehicles. The algorithm consists of two parts: first we generate several possible routes for each AGV, and then try to find k conflict-free paths among them with the minimum total length.

The first part can easily be solved: for each AGV we generate every possible routes in depth T (including the wait-edges), then run the ShortestPath method from every node reached.

Finding the best k avoiding paths (i.e. the solution where the total route lengths is the shortest possible) is more difficult. The problem can be formulated the following way: given k AGVs, with n_i paths $(i = 1, \dots, k)$: $\mathcal{P}_i$. Denote with $P(t)$ the t-th edge of the path P. Two paths are said to be disjoint, if for each $t \in \{1, \dots, \min(|P_1|, |P_2|, T)\}$:

$P_1(t) \neq P_2(t)$, and the start- and end-nodes of them also differ (where T denotes the number of steps in forward-looking). The goal is to find $P_i \in \mathcal{P}_i$, such that $\forall\, i, j : P_i \cap P_j = \emptyset$.

Claim 3. The problem can be reformulated as a minimum-weight k-independent set problem in a graph.

Proof. Consider a graph G with $n = \sum_{i=1}^{k} n_i$ vertices, where each node corresponds to a path in the original problem. Define the edges the following way: $(p_1, p_2) \in E \Leftrightarrow P_1 \cap P_2 \neq \emptyset$.

Find a minimum-weight independent set of size k. Trivially, the paths of the same AGV are intersecting therefore, they form cliques (complete subgraphs) in G. So we can pick at most one path for each vehicle, and k independent vertices give exactly k disjoint path. □

An initial solution can be found easily if the graph consists of at least $m+1$ nodes (where m is the number of AGVs), since the routes where at least one vehicle is moving to a free neighbouring node and the other vehicles wait in the first T steps is a trivial solution, but finding the optimum might be NP-hard. We are going to use the k-independent set formulation to solve the problem exactly.

When the k AGVs get in an unresolved conflict, the vehicle with the lowest priority triggers the optimization subroutine. The AGVs in the conflict will generate the paths and the optimization subroutine is invoked to find the conflict-free routes with the shortest total length.

This concludes the main result of the paper:

Corollary 4. The prioritized algorithm together with the subroutine is deadlock-free.

It is important to note that during a synchronization round, the optimization subroutine could be used multiple times, if after resolving a conflict, another lower priority vehicle fails to find an avoiding path.

4. NUMERICAL RESULTS

We tested the prioritized algorithm without and with the optimization subroutine for 103 test instances containing different graph structures and vehicle numbers (usually between 2-8). The set of tasks is fixed for each instance, but the online nature is preserved: there are 33 tasks, coming in each 3 time units between 1 and 100. We considered four different configurations of the algorithm: using one or two steps to look ahead in the conflict-resolving (i.e. $T \in \{1, 2\}$), and with priority A and B.

When only the simple prioritization algorithm was used, around 70 % of the test instances finished with success, and the number of livelocks occurred was negligible. This shows that even though our prioritized solution is not strong in the mathematical sense, it performs well in practice. We also applied the prioritization proposed by van den Berg and Overmars (2005) in our model, and received significantly worse results with around 55 % success rate.

We also analysed different measurements to rate the efficiency of the algorithm depending on the number of vehicles on the same network. For ten graphs, and number

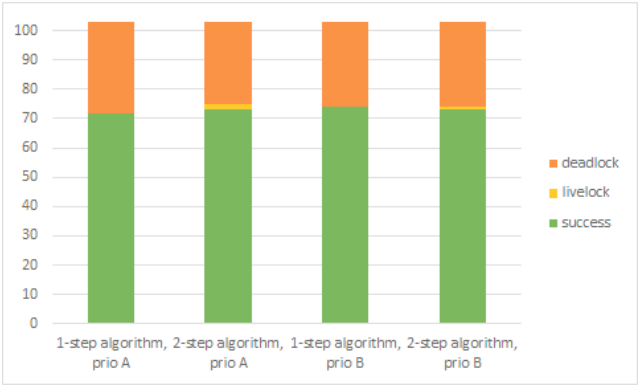

Fig. 3. Results for the simple prioritization algorithm.

of AGVs between 2 and 8, we considered the following for the simple prioritized algorithm:

- rate of the deadlock occurrence
- number of tasks completed
- algorithm termination time (considering the deadlocks as well)
- difference between the travelled distance and the initially planned route length
- number of useful steps (when the vehicle is in the *collect* or *deliver* state)
- number of extra steps (when the AGV has to move randomly in the *free* state, or its task is completed by another vehicle)

Naturally, the number of deadlocks increased when more AGVs tried to travel on the same network. However, even though deadlocks occurred, the number of tasks completed hasn't dropped drastically when increasing the number of vehicles. Around 5 AGVs, the efficiency started to fall, therefore the termination of the algorithm and the number of steps decreased. The travelled/planned distance rate slowly increased, but even for 8 vehicles it doesn't pass 1.8.

By using the optimization subroutine, all of the deadlocks were eliminated, and the number of livelocks hasn't increased significantly either. Therefore, combined with the original simple algorithm, it gives a good coordinating mechanism for AGV fleet.

Although livelocks were not avoided completely, this can be also solved by constantly checking the last few positions of the vehicles. Notably, if the occurrence of a position in the movement of the AGV's exceeds a fixed threshold, we use the optimization subroutine in order to step out from the livelock.

For analysing the algorithm together with the optimization subroutine, as the optimization subroutine eliminates deadlocks and uncompleted tasks, the first two measures are irrelevant. We replace them by the following metrics:

- optimization subroutine usage rate
- average number of calls of the optimization subroutine

The rate of the optimization subroutine usage increases with the number of vehicles, but only the case of 8 AGVs needed multiple calls of the subroutine. The termination time stayed low for almost every AGV numbers, the trav-

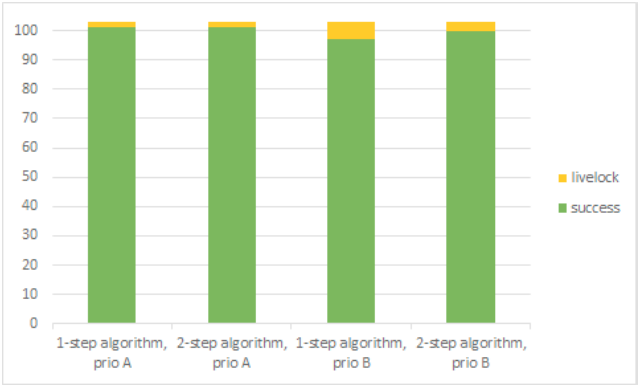

Fig. 4. Results for the prioritized algorithm together with the optimization subroutine.

elled/planned distance rate hasn't changed significantly compared to the simple prioritized algorithm. The number of useful steps stagnated, and the number of extra steps increased linearly.

Conclusively, these quantitative results demonstrate that our prioritization method shows superior performance compared to van den Berg and Overmars (2005).

5. CONCLUSION

We can conclude that although simple prioritization has strong mathematical bounds to ensure deadlock-free coordination, in practice it works efficiently and it has favourable properties like the decentralized control mechanism and the fact that we only look a couple of steps ahead, implying that communication between the vehicles and path planning don't require much computational resource. Coupled with the optimization subroutine (which might not be of polynomial complexity, but it is quite efficient in practice, and not used frequently) we get an efficient deadlock-free method for coordinating autonomously guided vehicles.

REFERENCES

Daniels, S.C. (1988). *Real time conflict resolution in automated guided vehicle scheduling*. Ph.D. thesis, Pennsylvania State University.

Das, P.K., Behera, H.S., Jena, P.K., and Panigrahi, B.K. (2016). Multi-robot path planning in a dynamic environment using improved gravitational search algorithm. *Journal of Electrical Systems and Information Technology*, 3(2), 295–313.

De Wilde, B., Ter Mors, A.W., and Witteveen, C. (2014). Push and rotate: a complete multi-agent pathfinding algorithm. *Journal of Artificial Intelligence Research*, 51, 443–492.

Digani, V. (2016). *Traffic Coordination for AGV Systems: An Ensemble Modeling Approach*. Ph.D. thesis, University of Modena and Reggio Emilia.

Draganjac, I., Miklić, D., Kovačić, Z., Vasiljević, G., and Bogdan, S. (2016). Decentralized control of multi-agv systems in autonomous warehousing applications. *IEEE Transactions on Automation Science and Engineering*, 13(4), 1433–1447.

Fanti, M.P. (2002). Event-based controller to avoid deadlock and collisions in zone-control agvs. *International Journal of Production Research*, 40(6), 1453–1478.

Fanti, M.P., Mangini, A.M., Pedroncelli, G., and Ukovich, W. (2018). A decentralized control strategy for the coordination of agv systems. *Control Engineering Practice*, 70, 86–97.

Fazlollahtabar, H., Saidi-Mehrabad, M., and Masehian, E. (2015). Mathematical model for deadlock resolution in multiple agv scheduling and routing network: a case study. *Industrial Robot: An International Journal*, 42(3), 252–263.

Kornhauser, D.M., Miller, G., and Spirakis, P. (1984). Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *25th Annual Symposium onFoundations of Computer Science, 1984.*, 241–250. doi:10.1109/SFCS.1984.715921.

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

Małopolski, W. (2018). A sustainable and conflict-free operation of agvs in a square topology. *Computers & Industrial Engineering*, 126, 472–481.

Möhring, R.H., Köhler, E., Gawrilow, E., and Stenzel, B. (2005). Conflict-free real-time agv routing. In *Operations Research Proceedings 2004*, 18–24. Springer.

Regele, R. and Levi, P. (2006). Cooperative multi-robot path planning by heuristic priority adjustment. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5954–5959. IEEE.

Roszkowska, E. and Reveliotis, S.A. (2008). On the liveness of guidepath-based, zone-controlled dynamically routed, closed traffic systems. *IEEE Transactions on Automatic Control*, 53(7), 1689–1695.

Ryan, M.R.K. (2008). Exploiting subgraph structure in multi-robot path planning. *Journal of Artificial Intelligence Research*, 31, 497–542.

Stenzel, B. (2008). *Online disjoint vehicle routing with application to AGV routing*. Ph.D. thesis, Technische Universität Berlin.

van den Berg, J.P. and Overmars, M.H. (2005). Prioritized motion planning for multiple robots. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 430–435. IEEE.

Velagapudi, P., Sycara, K., and Scerri, P. (2010). Decentralized prioritized planning in large multirobot teams. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 4603–4609. IEEE.

Wang, K.H.C. and Botea, A. (2011). Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research*, 42, 55–90.

Yu, J. and LaValle, S.M. (2013). Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27.

Zheng, Y., Xiao, Y., and Seo, Y. (2014). A tabu search algorithm for simultaneous machine/agv scheduling problem. *International Journal of Production Research*, 52(19), 5748–5763.