Multi-Agent Route Planning Using Delegate MAS

Hoang Tung Dinh, Rinde R. S. van Lon, Tom Holvoet
iMinds-DistriNet, KU Leuven, 3001 Leuven, Belgium
{hoangtung.dinh, rinde.vanlon, tom.holvoet}@cs.kuleuven.be

Abstract

Multi-agent route planning (MARP) is a problem that occurs in many applications such as automated guided vehicles, robotics, intelligent transportation networks and airplane taxiing. MARP becomes especially challenging when the application domain is dynamic, large scale and requires continual planning. Due to its decentralized nature, a multi-agent system (MAS) is an ideal candidate for solving dynamic and large scale MARP problems. Delegate MAS is a coordination mechanism based on the idea of intention propagation via the environment inspired by ant behavior. We evaluate delegate MAS on automated guided vehicle routing under realistic conditions. Delegate MAS is compared with context-aware routing, a state-of-the-art centralized approach for dynamic MARP. Two variants of MARP are considered, single-stage where vehicles each have to visit a single destination and multi-stage where a sequence of destinations has to be visited. The experiment results show that delegate MAS and context-aware routing have comparable solution quality while delegate MAS is more scalable for multi-stage routing in dynamic environments and offers higher throughput when continual planning is required.

1 Introduction

Automated guided vehicle (AGV) systems are widely used in many industrial areas, including manufacturing, aviation, retail and transportation logistics (Ullrich 2015). In such systems, a fleet of autonomous vehicles coordinate in order to efficiently transport goods throughout a plant or warehouse. Transportation requests need to be assigned to vehicles and each request contains information about pick-up and delivery locations. Vehicles need to go to battery charging stations several times during execution. The goal of the vehicle fleet is to determine efficient routes that minimize transportation time and maximize throughput. This is a challenging task (Vis 2006). Unexpected obstacles such as humans may block a road; vehicles may temporarily fail; new vehicles may become available and operating vehicles may leave the system for maintenance. This results in unpredictable travel time and transportation requests may change during execution. Additionally, the infrastructure and the number of vehicles can be large; the physical constraints of the infrastructure and vehicles may lead to deadlocks, preventing some vehicles to reach their destinations.

AGV systems, along with robotics, intelligent transportation and airplane taxiing, are typical applications of multi-agent route planning (MARP). In MARP, there is a set of agents in a shared environment. The problem involves planning a conflict-free route for each agent from its current position to one or multiple destinations. Ter Mors (2010, pp. 46–50) proves that MARP is NP-complete, or even PSPACE-complete under additional constraints such as maintaining a minimum distance between agents. Typical challenges of MARP are dealing with dynamism, scalability, communication limitation and deadlock situations. MARP requires a flexible and scalable solution that seamlessly copes with unexpected events and failures. Agents should collaboratively plan routes to resolve conflicts before they occur. These characteristics motivate the feasibility study of a decentralized multi-agent approach.

In this paper we implement and evaluate an online, anytime and continual planning approach for MARP, called delegate MAS. Delegate MAS is an environment centric coordination mechanism for coordination and control applications, inspired by food foraging behavior in ant colonies. It was first introduced in the context of manufacturing control (Holvoet and Valckenaers 2007). Since then, delegate MAS has been applied in different areas such as pick-up and delivery (Hanif et al. 2011) and anticipatory vehicle routing (Weyns, Holvoet, and Helleboogh 2007; Claes, Holvoet, and Weyns 2011). We compare delegate MAS with a state-of-the-art centralized decoupled approach, called context-aware routing (see Section 2). To the best of our knowledge, context-aware routing is the only approach that solves both single-stage (Ter Mors, Zutt, and Witteveen 2007) and multi-stage routing (Ter Mors, Van Belle, and Witteveen 2009), as well as routing in a dynamic environment where unexpected incidents occur regularly (Ter Mors and Witteveen 2009). We evaluate the performance of both approaches in static and dynamic environments, in single as well as multi-stage routing. In single-stage routing, an agent has exactly one destination to go to. In multi-stage routing, an agent has a sequence of destinations. We make abstraction of the way agents get to know their destinations and assume that they are informed when the destinations need to be adapted. The experiment results show that delegate MAS and context-aware routing have comparable solution quality while delegate MAS is more scalable for multi-stage routing in dy-

namic environments and offers higher throughput when continual planning is required.

The rest of this paper is organized as follows. We first discuss related work (Section 2). Then, we formulate the model of the MARP problem (Section 3). After that, we outline the delegate MAS approach (Section 4). We then describe the experiment setup and analyze experiment results (Section 5). Finally, we draw our conclusion and detail possible future work (Section 6).

2 Related Work

There are coupled and decoupled approaches for solving MARP.

Coupled approaches combine the configuration spaces of all individual agents into one composite configuration space which is then searched for a solution. Several coupled approaches (Ryan 2008; Standley and Korf 2011) can solve MARP optimally using the *A** algorithm (Hart, Nilsson, and Raphael 1968). Such approaches do not scale well because the branching factor of search spaces grows exponentially as the number of agents increases. Sharon et al. (2013; 2015) exploit the sparsity in interactions among agents to improve the efficiency in finding the optimal solution. Their approaches perform poorly when there is a high rate of conflicts among agents.

Decoupled approaches decompose the problem of searching for a global solution for all agents into a sequence of individual planning problems. Decoupled approaches offer scalability but are sub-optimal and often incomplete. One type of decoupled approach, called rule-based planning, defines a set of specific movement rules for agents to reduce the computational complexity and guarantee the completeness at the cost of solution quality (De Wilde, Ter Mors, and Witteveen 2013; Wang and Botea 2008). Another type of decoupled approach, prioritized planning, assigns a unique priority to each agent. Agents plan routes in decreasing priority order. An agent finds a route that does not create a conflict with the plans of higher priority agents. Silver (2005) proposed Hierarchical Cooperative *A** and its variant, Windowed Hierarchical Cooperative *A**, where an agent searches for a route in a three-dimensional space-time reservation table. Wang and Goh (2011) proposed an approach where agents search for routes in a two-dimensional map with an adaptive priority re-assignment strategy. These approaches are not complete and their solutions may be far from optimal. Wang and Goh (2013) then proposed the Guided Iterative Prioritized Planning approach that can increase the success rates at the cost of computational time. Hatzack and Nebel (2014), Lee, Lee, and Choi (1998) and Ter Mors et al. (2012) presented the Fixed-Path Scheduling (FPS) approach where each agent, in a given priority, calculates conflict-free schedules along one or multiple predetermined paths and takes the shortest-time schedule as its plan. The pre-determined path(s) for each agent can be the shortest-length path (Hatzack and Nebel 2014), k-shortest-length paths (Lee, Lee, and Choi 1998) or k-disjoint paths (Ter Mors et al. 2012).

Ter Mors, Zutt, and Witteveen (2007) proposed a state-of-the-art prioritized planning approach called context-aware routing. In context-aware routing, an agent finds its optimal plan that does not create a conflict with existing plans of other agents on a free time window graph using an *A**-like algorithm. A free time window on a location is the maximal time interval that a new agent can make a reservation for traversing the location without making any conflict with the existing reservations of other agents. Ter Mors et al. (2007; 2012) show that context-aware routing is better in terms of travel time than all variants of FPS.

All the work we have discussed so far only considers the single-stage routing problem. To the best of our knowledge, the paper by Ter Mors, Van Belle, and Witteveen (2009) is the only work in the literature that deals with the multi-stage routing problem. Extended from their single-stage routing algorithm, Ter Mors, Van Belle, and Witteveen proposed the context-aware multi-stage routing algorithm that is also a prioritized planning approach.

To deal with incidents that delay agents, different approaches (Maza and Castagna 2005; Ter Mors and Witteveen 2009; Ter Mors 2011) have been developed to integrate with context-aware routing. In those approaches, after all agents make their plans, the schedule at each location is converted to a visiting order (or priorities) of agents. Maza and Castagna (2005) show that if agents maintain their priorities at each location, no conflict occurs even if the arrival times of agents are not guaranteed. Hence, an approach to avoid conflict is to let each agent respect its priority at each location. This approach requires non-delayed agents to wait for delayed agents. To achieve better solution, the work in (Maza and Castagna 2005; Ter Mors and Witteveen 2009) increases the priorities of non-delayed agents over delayed ones. In (2011), Ter Mors proposes another approach where agents re-plan routes every time an incident occurs. According to the comparisons in (Ter Mors and Witteveen 2009; Ter Mors 2011), the increasing-priority approach of Ter Mors and Witteveen (2009) achieves the best solution quality.

3 Problem Formulation

In this section we present the formal model used throughout the paper. We adopt the model introduced by Ter Mors, Van Belle, and Witteveen (2009) with several modifications to make the model more realistic. In (2009), Ter Mors, Van Belle, and Witteveen assume that agents have zero length. Such assumption is unrealistic. In our model, we assume that agents occupy physical space. This assumption leads to the differences in the resource capacity constraints and the agent plan constraints between our model and Ter Mors's model.

MARP consists of a set A of agents operating in an infrastructure. The infrastructure is a bidirectional graph $G = (V, E)$, where V is the set of vertices representing locations that agents can visit such as intersections or pick up and delivery locations, and E is the set of edges representing lanes connecting locations. The set of resources is $R = V \cup E$. Each resource $r \in R$ has a capacity $C(r)$. Each vertex has unit capacity. The capacity of each edge e is:

$$C(e) = \lfloor length(e) / \big((1 + \Delta) \times length(a)\big) \rfloor$$

where $length(e)$ is the length of the edge, $length(a)$ is the

length of an agent and Δ is the minimum separation between two agents. We assume homogeneous agents. If multiple agents occupy an edge at the same time, they must all travel in the same direction and must not overtake each other. An agent cannot change its direction when it travels along an edge. In Figure 1, agent 2 is not allowed to overtake and cannot exit from the edge before agent 3.

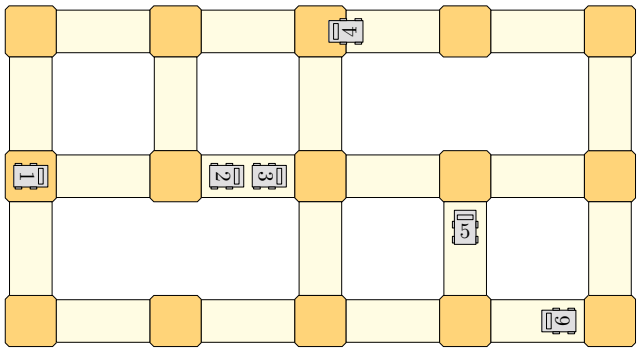


Figure 1: Agents in an infrastructure graph. Dark yellow squares represent vertices (intersections and locations). Light yellow rectangles represent edges (lanes).

In single-stage routing, an agent $a \in A$ has one start location $s \in V$ and one destination location $d \in V$. In multi-stage routing, an agent a has a tuple of destinations $D = \{\langle d_1, \ldots, d_m \rangle | d_i \in V\}$, where m is the number of destinations. The route plan of an agent is a sequence:

$$(\langle r_1, [t_1, t'_1) \rangle, \ldots, \langle r_n, [t_n, t'_n) \rangle)$$

of n plan steps. A plan step $\langle r_i, [t_i, t'_i) \rangle$ consists of a resource r_i, the entry time t_i and the exit time t'_i of agent a on r_i. In single-stage routing, the last plan step of an agent must be in the destination resource, that is, $r_n = d$. In multi-stage routing, the plan of an agent must include all the destinations in a given order and the last plan step must be in the last destination resource, that is, $r_n = d_m$. Two resources r_i and r_{i+1} of two consecutive plan steps must be adjacent in G. In the model of Ter Mors, Van Belle, and Witteveen (2009), because they assume that agents have zero length, each agent only occupies one resource at the same time. Therefore, the exit time and the entry time of two consecutive plan steps must meet each other, that is, $t'_i = t_{t+1}$. In our model, because each agent has a length greater than zero, an agent can occupy two adjacent resources simultaneously. For example, in Figure 1, agent 4 is occupying two resources. Thus, in our model, two intervals of two consecutive plan steps $[t_i, t'_i)$ and $[t_{i+1}, t'_{i+1})$ must overlap, that is, $t'_i > t_{t+1}$. The duration $\delta_i = t'_i - t_i$ must be sufficient for the agent to traverse through the resource r_i. The schedule of a resource consists of a set of plan steps. A resource has a consistent schedule if the load of the resource never exceeds the resource's capacity.

Unexpected incidents that delay agents may occur. We model incidents as events that make agents temporarily inactive. Each incident has a start time t and a duration δ_t. While suffering an incident, an agent cannot move. Agents do not have prior knowledge of incidents, that is, they do not know when incidents happen nor their duration.

4 MARP Using Delegate MAS

In this section we propose the delegate MAS approach for MARP. Delegate MAS consists of a number of autonomous agents situated in a shared environment. Agents coordinate in a decentralized way. The shared environment enables indirect communication between agents. An agent drops information of its plan to the relevant parts of the environment. Other agents can later use the information to create their plans. Such communication somewhat resembles the foraging behavior of ants, where an ant continuously drops pheromones on the environment and scents pheromones of other ants. Agents only collect directly relevant information that is distributed throughout the environment for making decisions. Delegate MAS self-organizes by continuously removing invalid information in the environment.

4.1 Multi-Agent Based Routing

Delegate MAS consists of two types of primary agents, *resource agents* and *vehicle agents* (Weyns, Holvoet, and Helleboogh 2007).

Each resource agent represents a resource of the infrastructure. A resource agent can observe changes such as unexpected obstacles at its resource. The main task of a resource agent is to manage a schedule on its resource and to provide free time windows according to the current schedule. A resource agent only allows a vehicle to enter the resource if it is consistent with the schedule. Also, a resource agent only accepts reservations that are consistent with its existing schedule. A reservation has a time-to-live. If a reservation is not confirmed regularly, the resource agent removes it. A resource agent can communicate with its neighboring resource agents. The network of resource agents establishes a *virtual environment*, that is, a software representation of the physical infrastructure graph.

Each vehicle agent represents an operating vehicle in the infrastructure. When a new vehicle enters the infrastructure, a corresponding vehicle agent is created and assigned to the vehicle. We assume that each vehicle agent knows the static graph structure of the infrastructure, but not the schedules on resources. A vehicle agent is responsible for planning routes and controlling its vehicle towards destinations. A vehicle agent continually explores alternative routes and reserves its intended route. The behavior of a vehicle agent is described in Algorithm 1. To explore routes, first, in line 2, a vehicle agent generates a set of feasible paths[1] from its current location to its destination(s) using the static infrastructure graph (Section 4.4). In line 3, it evaluates the quality of each path by asking relevant resource agents about the existing schedules (Section 4.2). Then, in line 4, it selects the most preferable one among the assessed routes. In line 5, the vehicle agent decides whether to deviate from its current plan to the new route. After that, in line 8, the vehicle agent makes reservations for its intended route and regularly refreshes the

[1] A route is a path with a schedule.

reservations via resource agents (Section 4.2). When planning routes, a vehicle agent must respect the existing reservations of other vehicle agents.

Algorithm 1 Behavior of a vehicle agent

Require: The infrastructure graph $G = (V, E)$, current location $start$, destination(s) $dest$

```
1:  loop
2:      paths ← getAlternativePaths(G, start, dest)
3:      routes ← evaluate(paths)
4:      preferredRoute ← select(routes)
5:      if revise(intendedRoute, preferredRoute) then
6:          intendedRoute ← preferredRoute
7:      end if
8:      if ¬makeReservation(intendedRoute) then
9:          goto 2
10:     end if
11: end loop
```

4.2 Agent Coordination

To achieve coordination, vehicle agents and resource agents communicate indirectly through the virtual environment using lightweight agents, called "ants". Basically, ants are smart messages that can autonomously move in the virtual environment and interact with resource agents (Holvoet, Weyns, and Valckenaers 2009). We use two types of ants: *exploration ants* and *intention ants*.

1) Exploration ants. Recall Algorithm 1. In line 3, a vehicle agent evaluates candidate paths by sending out exploration ants. Each exploration ant follows a candidate path through the virtual environment. For each resource agent that it travels through, the exploration ant queries for the existing schedule. After reaching the destination, based on all relevant reservations, the exploration ant calculates the optimal schedule along its path using the context-aware routing algorithm proposed in (Ter Mors, Zutt, and Witteveen 2007). Note that we only use context-aware routing to search for the optimal schedule on a given path, but not on the entire infrastructure graph as in (Ter Mors, Zutt, and Witteveen 2007). The exploration ant then informs the vehicle agent of its optimal schedule. In line 4, the vehicle agent compares all schedules reported by exploration ants and selects the best one. The criteria for the selection depend on concrete objectives. In this paper, a vehicle agent selects the schedule with the earliest arrival time at the destination.

2) Intention ants. In line 8 of Algorithm 1, a vehicle agent informs relevant resource agents of its plan by sending out an intention ant. The intention ant follows the intended path in the virtual environment and makes reservation with each resource agent along its path. Then, it reports to the vehicle agent whether it successfully made reservations on all resources. If the intention ant cannot reserve all the resources, the vehicle agent explores again (line 9). The reservations evaporate after a while if intention ants do not renew them. The vehicle agent therefore sends out intention ants regularly to refresh its reservations. A vehicle agent can freely change its intention. In line 5, if it finds a better route and decides to change the current plan, it stops sending intention ants on the current route and the incorrect reservations will disappear after a while. This is an example of the self-organizing property of delegate MAS. A vehicle agent repeats the exploration and reservation processes regularly, thus gradually improves the quality of its plan.

Different from context-aware routing that only allows agents to plan sequentially, delegate MAS allows agents to plan concurrently and can resolve conflicts caused by simultaneous exploring and reserving. If two intention ants arrive and request to make reservations at a resource at the same time, the resource agent randomly selects an order to process the requests. For example, in Figure 2, vehicle agent 1 and vehicle agent 2 explore routes at the same time. At that moment, no vehicle agent already placed reservations. Hence, after exploring, vehicle agent 1 selects the solid blue route and vehicle agent 2 chooses the dashed red route. They plan to travel with the same speed and therefore may collide at the resource marked by a star. They then send intention ants to make reservations. At the star resource, assuming that intention ant 1 arrives earlier, or at the same time with intention ant 2 but is selected by the resource agent to process request first. The resource agent accepts the reservation of intention ant 1 and then rejects the reservation of intention ant 2 because of schedule inconsistency. Intention ant 1 then continues making reservations while intention ant 2 stops and reports to vehicle agent 2. Vehicle agent 2 then explores new routes again. The reservations made by intention ant 2 in other resources are not refreshed and evaporate after a while thanks to the self-organizing effect.

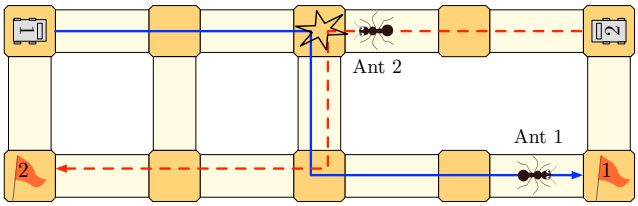


Figure 2: Two vehicle agents explore simultaneously and try to reserve resources for two conflicted plans. The resource agent at the first conflicted resource processes requests sequentially. Thus it only allows the reservation from intention ant 1 and rejects the reservation from intention ant 2.

4.3 Delay Propagation

If an incident occurs, the delayed vehicle agent estimates the delay duration δ. It then updates its current plan:

$$(\langle r1, [t1_s, t1_e)\rangle, \langle r2, [t2_s, t2_e)\rangle, \ldots, \langle rn, [tn_s, tn_e)\rangle)$$

to a new plan:

$$(\langle r1, [t1_s, t1_e + \delta)\rangle, \langle r2, [t2_s + \delta, t2_e + \delta)\rangle, \ldots, \langle rn, [tn_s + \delta, tn_e + \delta)\rangle)$$

where $r1$ is the resource that the vehicle is occupying. After that, the vehicle agent sends an intention ant to request relevant resource agents to update its reservations.

The updated reservations of a delayed vehicle may be inconsistent with the existing reservations of other vehicles. To resolve inconsistencies, a resource agent also delays the reservations of vehicles that enter the resource after the delayed one (see Figure 3a). The resource agent notifies its neighboring resource agents who apply the delay to all affected and succeeding vehicles recursively (see Figure 3b). Consequently, the changes propagate through the entire virtual environment. This delay propagation process guarantees that all updated plans are consistent and deadlock-free if the initial plans are consistent and deadlock-free. A vehicle agent updates its plan using the information reported by intention ants. After the delay is propagated, some non-delayed vehicles have to wait for the delayed one. Because the vehicle agents send out exploration ants regularly, alternative routes can be found around the delayed vehicle.

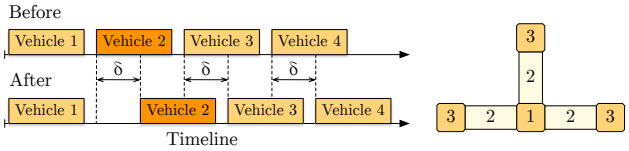


Figure 3: (a) The schedule of a resource agent before and after being requested to update the reservation of vehicle 2. The reservation of vehicle 2 and all succeeding reservations are delayed by $\boldsymbol{\delta}$. (b) Delay propagation process. Resource agent 1 receives a notification of delay. It recursively propagates the delay to its neighbors (marked by 2 and 3).

The estimated delay duration δ does not need to be the exact actual duration. However, δ should be as close to the actual delay duration as possible. Vehicle agents do not need to know the actual duration of an incident in advance, which is realistic in real-world applications. If the actual delay duration is shorter than δ, the vehicle agent explores new routes after the incident is over. If the actual incident duration is longer than δ, the vehicle agent, after estimation δ expires, estimates a new delay duration δ' and propagates it again. The process repeats until the incident is over. The more accurate δ is, the less disturbance in the plan of other vehicle agents. If an intention ant is making new reservations during the delay propagation process, the reservations may conflict with updated resource schedules. Resource agents then reject the inconsistent reservations of the intention ant. The intention ant reports the reservation failure to the vehicle agent and the vehicle agent explores again.

4.4 Alternative Path Finding

Generating possible candidate routes using the static infrastructure graph (line 2 of Algorithm 1) is an important step in exploration. Poor candidate paths lead to poor plans. Finding all paths between two locations is impractical. If we only select the shortest paths as candidate paths, congestion may occur in some central resources that have many shortest paths passing through (Ter Mors et al. 2012). However, long paths result in long travel time. Thus, it is necessary to have a diverse set of candidate paths. Moreover, because a vehicle agent explores regularly, the set of candidate paths should be different from time to time in order to increase the chance of finding a good plan.

In the literature, popular approaches to generate a set of feasible paths include k-shortest paths (Yen 1971), k-disjoint paths (Suurballe and Tarjan 1984), Pareto (Delling and Wagner 2009), Plateau (Bader et al. 2011) and Penalty (Chen, Bell, and Bogenberger 2007). For the details of these approaches, we refer readers to the review in (Bader et al. 2011). In lattice-like structures that are popular in warehouses or harbors, k-shortest paths are often similar. In k-disjoint paths, the set of paths depends much on the shortest path that is also the first path in the set. Bader et al. (2011) show that Pareto leads to solutions of low quality while Plateau and Penalty give comparable good results. We select the Penalty approach with some modifications because it is simpler than the Plateau approach.

The Penalty approach iteratively calculates the shortest path using the A^* algorithm. In each iteration, it adds the shortest path to the result set and penalizes the path by increasing edge weights. The approach terminates after receiving a sufficient number of alternative paths. In (2007), Chen, Bell, and Bogenberger, suggests increasing all edges in each found path by a relative penalty. We adopted their suggestion but preliminary experiments provide poor results. Therefore, we adapt the Penalty approach by adjusting the weight using a more probabilistic way. Algorithm 2 describes our alternative path finding algorithm.

Algorithm 2 Alternative Path Finding

Require: Infrastructure graph $G = (V, E)$, number of alternative paths N, current location $start$, destination(s) $dest$

1: $S \leftarrow \emptyset$
2: $numFails \leftarrow 0$
3: $maxFails \leftarrow 3$
4: $\alpha \leftarrow$ uniformly random value between 0 and 1
5: **while** sizeOf(S) $< N \wedge numFails < maxFails$ **do**
6: $p \leftarrow$ getShortestPath($G, start, dest$)
7: **if** $p \in S$ **then**
8: $numFails \leftarrow overlap + 1$
9: **else**
10: $S \leftarrow S \cup p$
11: $numFails \leftarrow 0$
12: **end if**
13: **for all** vertex $V_i \in p$ **do**
14: $\beta \leftarrow$ uniformly random value between 0 and 1
15: **if** $\beta < \alpha$ **then**
16: **for all** edge E_i directly connected to V_i **do**
17: weight(E_i) $\leftarrow penalty$
18: **end for**
19: **end if**
20: **end for**
21: **end while**
22: **return** S

In line 1, we initialize the solution set S. The variable $numFails$ in line 2 counts the current number of consecu-

tive times that the algorithm fails to find a new path. In line 4, we generate a uniformly random number α between 0 and 1. The while loop of line 5 iterates until we find a sufficient number of alternative paths or the algorithm fails to find a new path $maxFails$ times consecutively[2]. First, in line 6, we calculate the shortest path p using the $A*$ algorithm. Then, we check whether path p is already in the solution set S in line 7. If it is the case, we increase $numFails$ by one. If p is not in S yet, we add p to S and reset $numFails$ to zero. To penalize each found path p, the for loop in line 13 iterates over all the vertices belonged to p. For each vertex V_i, with probability α, we set the weight of all edges directly connected to V_i to $penalty$, a large value in comparison with the initial weight. If α is large, resulting paths are different from each other. If α is small, they tend to be similar. We generate different α value for each exploration process.

In single-stage routing, a vehicle agent calculates the shortest path between its current position and its only destination using the $A*$ algorithm (line 6 of Algorithm 1). In multi-stage routing, because a vehicle agent has multiple destinations $d_1, d_2, \ldots, d_n$, it calculates the shortest path p by concatenating the shortest paths between each pair of destinations. Using the $A*$ algorithm, the vehicle agent calculates n shortest paths: path p_1 between the origin and d_1, path p_2 between d_1 and d_2, ..., path p_n between d_{n-1} and d_n. Then, the candidate path p is the concatenation of all the shortest paths $p_1, p_2, \ldots, p_n$.

5 Experiments

In this section, we compare delegate MAS with context aware routing in single-stage and multi-stage routing scenarios, in static and dynamic scenarios, and in scenarios where vehicles receive new destinations continually. We implemented all algorithms in RinSim (Van Lon and Holvoet 2012), version 4.00 (Van Lon 2015), an open-source discrete-time simulator for logistics that supports MARP. We conducted experiments on a machine with four Intel Xeon X5677 3.47GHZ processors and 12GB RAM.

5.1 Experiment Setup

We evaluate the two approaches on a 30×30 lattice infrastructure with 900 vertices and 1740 edges. Each edge has capacity of two. In total, there are 2640 resources. Each vehicle has a speed of one meter per second.

In single-stage routing, we assume that at the beginning, all vehicles did not enter the infrastructure yet. A vehicle can decide when to enter the infrastructure at its origin. After reaching its destination, a vehicle leaves the infrastructure. In multi-stage routing, we assume that each vehicle has its own parking place. A parking place is a terminal vertex connected to the infrastructure and has unit capacity. A parking place of a vehicle cannot be the destination of other vehicles. Also, a vehicle cannot visit the parking places of other vehicles. The multi-stage routing scenarios reflect the AGV routing problem, where a vehicle cannot leave or enter the infrastructure at every resource. A vehicle always includes its parking place as the last destination in its plan. Initially, each vehicle stays at its parking place. We assign three different destinations for each vehicle. After receiving destinations, a vehicle starts from its parking place, visits all three destinations in the given order and then comes back to the parking place. Therefore, a vehicle has to plan for visiting four destinations (the last destination is its parking place). Figure 4 illustrates a small lattice infrastructure with four vehicles at their parking places.

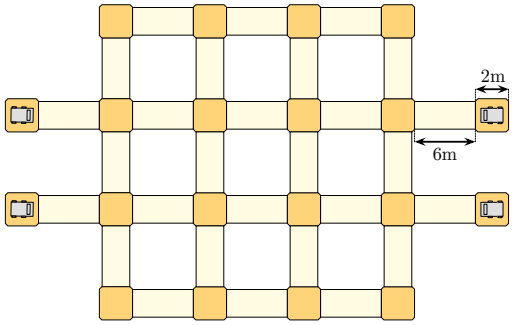


Figure 4: A $\mathbf{4 \times 4}$ lattice graph structure with four vehicles at their parking places.

In static scenarios, we compare delegate MAS with context-aware single-stage routing (CA) (Ter Mors, Zutt, and Witteveen 2007) and context-aware multi-stage routing (CA) (Ter Mors, Van Belle, and Witteveen 2009). In such scenarios, there is no disturbance that makes the operation of a vehicle deviate from its plan. Hence, the traveling time of a vehicle is always consistent with its reservations. In dynamic scenarios, we compare delegate MAS with (1) the baseline approach (CA-Baseline) where the priorities of vehicles on a resource never change and with (2) the increasing-priority approach (CA-IP) (Ter Mors and Witteveen 2009), which is the best strategy that vehicles can employ to deal with incidents according to the studies in (Ter Mors and Witteveen 2009; Ter Mors 2011) (see Section 2). We use a homogeneous Poisson process to generate 20 incidents per 10000 seconds for each vehicle. The duration of an incident is a uniformly random value between one and 100 seconds. If two incidents overlap, they form a longer incident. If a vehicle suffers from an incident, it cannot move until the incident is over.

For each setting combination (static / dynamic, single / multi-stage), we vary the number of vehicles from 10 to 100 with steps of 10 and generate 10 problem instances per number of vehicles. A problem instance consists of a start location, a destination (single-stage routing) or a sequence of destinations (multi-stage routing), and a list of incidents (the list is empty in static scenarios) for each vehicle. A vehicle knows its destination(s) at the beginning. We stop a simulation after all vehicles reached their destinations and measure the *average travel time*:

$$\overline{\tau} = \frac{\sum_{i=1}^{|A|} \tau_i}{|A|}$$

where $|A|$ is the number of vehicles and τ_i is the travel time of vehicle i. In single-stage routing, τ_i is the time when ve-

[2]It can be the case that the graph does not have enough alternative paths.

hicle i first reaches its destination. In multi-stage routing, τ_i is the time when vehicle i arrives at its parking place after reaching all of its destinations in a given order. In the most realistic setting, multi-stage routing in dynamic environments, we also measure the *running time* of each delegate MAS simulation and CA-IP simulation.

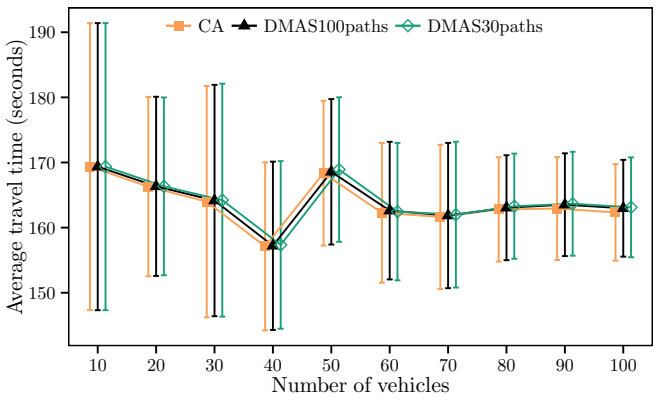
To evaluate our approach on an AGV routing scenario, where vehicles receive new destinations regularly in a dynamic environment, we compare delegate MAS to CA-IP. In the AGV routing problem, new transportation tasks may appear continually during runtime. Hence, in this scenario, at the beginning, we assign three different destinations for each vehicle. After a vehicle reached all the destinations in a given order and came back to its parking place, we assign it three new destinations. A simulation stops after a specified period (10000 seconds). We measure the *throughput*, that is, the total number of reached destinations by all vehicles after the simulation stops. In reality, the throughput represents the number of tasks completed by the system.

In each problem instance, we execute delegate MAS with two different settings where a vehicle agent generates 30 and 100 alternative paths each time it explores new routes. A vehicle agent explores new routes and refreshes its reservations with a period of eight seconds.

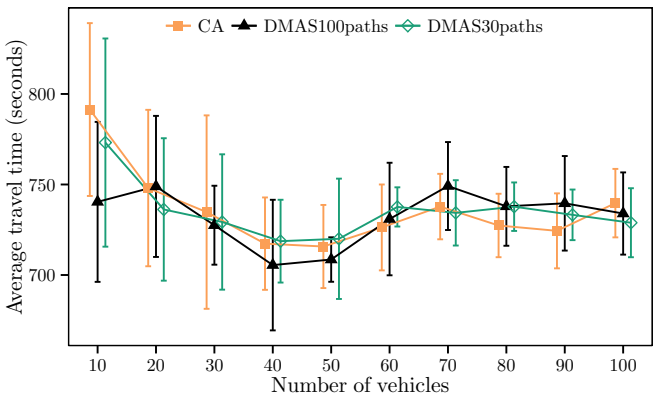
5.2 Experiment Results

Figure 5 shows the *average travel time* of delegate MAS and context-aware routing. In static scenarios, delegate MAS and CA provide comparable results. In static single-stage routing (Figure 5a), CA and delegate MAS achieve similar average travel time. We observed that there are only few interference interactions among vehicles in this scenario. Therefore, the shortest-time paths are often also the shortest-length paths. In delegate MAS, when exploring new routes, a vehicle agent always includes the shortest-length path in the set of alternative paths. Hence, delegate MAS achieves similar performance to context-aware routing. In static multi-stage routing (Figure 5b) where there are more interactions among vehicles, no approach is consistently superior to the other. The reasons that delegate MAS and CA have comparable results while CA computes single-agent optimal route and delegate MAS only samples several routes in the environment can be explained as follows. CA computes single-agent optimal routes sequentially leading to a global Pareto-optimal solution and there is no guarantee about global optimality. Delegate MAS regularly samples the environment and its solution also gradually converges to a Pareto-optimum.

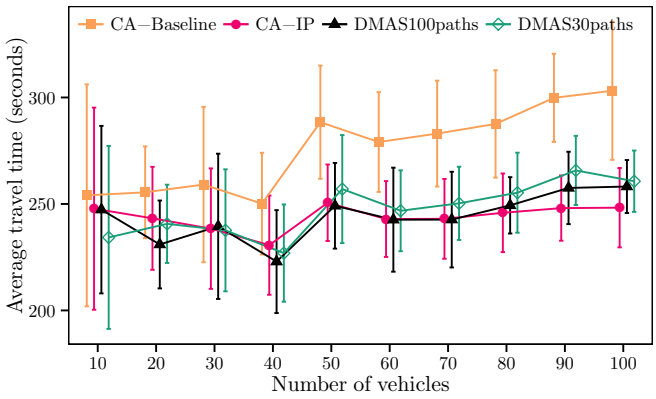
In dynamic scenarios (Figure 5c and 5d), both delegate MAS and CA-IP achieve lower average travel time than that



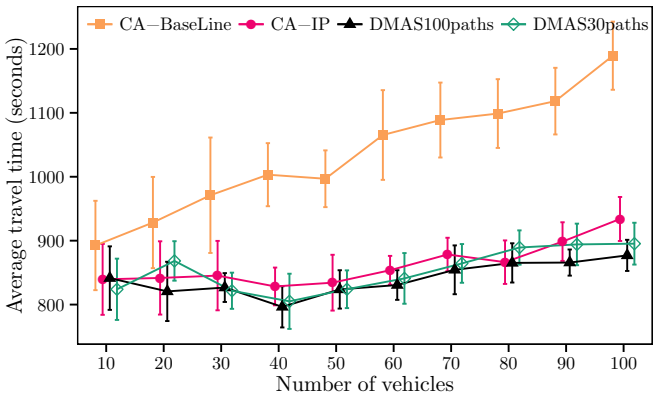
(a) Static single-stage routing



(b) Static multi-stage routing



(c) Dynamic single-stage routing



(d) Dynamic multi-stage routing

Figure 5: Average travel time per vehicle of each approach in different scenarios. Each data point is the average of results from 10 independent simulations.

of CA-Baseline. As the number of vehicles increases, the average travel time of CA-Baseline goes up while those of delegate MAS and CA-IP remain stable. The trend is more obvious in multi-stage routing in comparison with single-stage routing. It can be explained that CA-Baseline requires non-delayed vehicles to wait for delayed vehicles. The more incidents occur, the more delay that vehicles suffer from. As the number of vehicles increases, more incidents happen. Also, there are more incidents if vehicles travel in longer routes. Delegate MAS and CA-IP have comparable results. In single-stage routing, the plan cost of CA-IP tends to be lower than that of delegate MAS as the number of vehicles increases. In multi-stage routing, delegate MAS with 100 alternative paths consistently achieves lower mean than CA-IP, except for the 10-vehicle setting. However, the difference is not significant.

Figure 6 shows that in dynamic multi-stage routing, as the number of vehicles increases, the *running time* of CA-IP increases super-linearly while that of delegate MAS only increases linearly. Moreover, delegate MAS offers a trade-off between running time and solution quality. Decreasing the number of alternative paths for exploration yields a faster running time at the cost of plan quality. From these experiments, we conclude that delegate MAS is more scalable while providing comparable results with CA-IP.

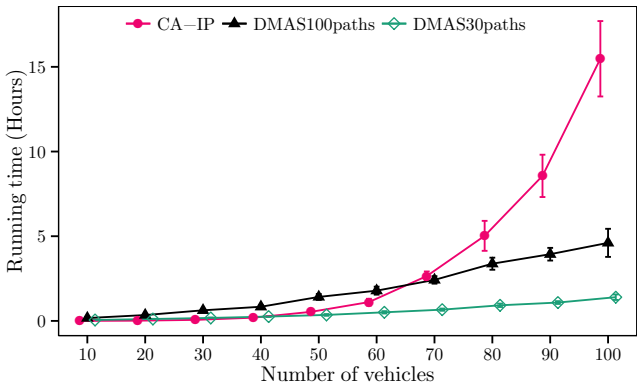


Figure 6: Simulation running time of the dynamic multi-stage routing scenario.

In the last experiment where vehicles receive new destinations continually, the result in Figure 7 shows that delegate MAS achieves higher *throughput* than CA-IP. It can be explained that CA-IP only considers the priorities of vehicles on each resource. After changing the visiting order of vehicles, the existing reservations are invalid. Therefore, a vehicle cannot plan a new route or a new vehicle cannot make a plan when other vehicles are still operating. After a vehicle finishes its tasks and comes back to its parking place, it has to wait until all other vehicles complete their tasks and arrive at their parking places so that all vehicles can start making new plans together. In delegate MAS, the self-organizing capability guarantees the validity of reservations. Therefore, a vehicle can plan a new route at any time. After being assigned new destinations, a vehicle can immediately make a plan without having to wait for other vehicles. Thus, delegate MAS can deal with application domains where vehicles have to plan new routes, new vehicles enter the infrastructure or vehicles have to change destinations during execution.

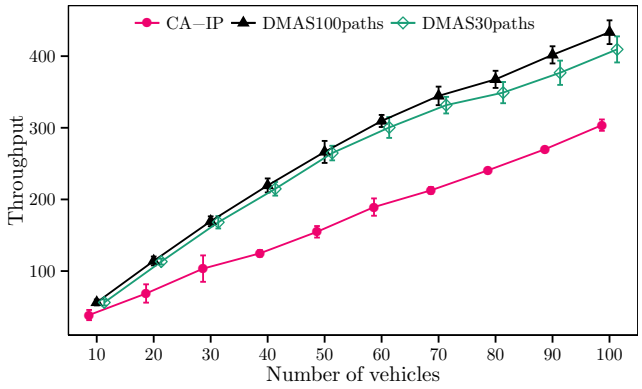


Figure 7: Throughput in the dynamic multi-stage routing scenario after 10000 seconds.

6 Conclusion

In this paper we present the delegate MAS approach for MARP. Delegate MAS can solve single-stage as well as multi-stage routing, continual routing and routing in a dynamic environment. In delegate MAS, the control is decentralized and the system self-organizes to adapt to changes in the environment, thus allowing concurrent activities of different agents. Moreover, it does not require global knowledge about the environment for each agent. Therefore, delegate MAS can be physically deployed and operated as a distributed software system.

Our evaluation is more realistic than previous work. In comparison with a state-of-the-art centralized decoupled approach, delegate MAS provides comparable solution quality while it offers better scalability in a dynamic environment. Moreover, delegate MAS achieves higher throughput when vehicles are required to plan new routes continually. Delegate MAS also offers a trade-off between computational complexity and solution quality.

Our future work will focus on tuning parameters for the exploration process, especially for the alternative path finding algorithm. We also plan to incorporate negotiation mechanisms in our approach. For example, when exploring routes, a vehicle agent may negotiate with other vehicle agents to achieve lower global cost. We will extend delegate MAS so that it can cope with both routing and charging tasks in the AGV transportation problem.

Acknowledgments

This research is partially funded by the Research Fund KU Leuven. We thank Adriaan ter Mors for sharing his implementation of the context-aware routing algorithms.

References

Bader, R.; Dees, J.; Geisberger, R.; and Sanders, P. 2011. Alternative Route Graphs in Road Networks. In *Theory and Practice of Algorithms in (Computer) Systems*. 21–32.

Chen, Y.; Bell, M.; and Bogenberger, K. 2007. Reliable Pretrip Multipath Planning and Dynamic Adaptation for a Centralized Road Navigation System. *IEEE Transactions on Intelligent Transportation Systems* 8:14–20.

Claes, R.; Holvoet, T.; and Weyns, D. 2011. A Decentralized Approach for Anticipatory Vehicle Routing Using Delegate Multiagent Systems. *IEEE Transactions on Intelligent Transportation Systems* 12:364–373.

De Wilde, B.; Ter Mors, A. W.; and Witteveen, C. 2013. Push and rotate: cooperative multi-agent path planning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 87–94.

Delling, D., and Wagner, D. 2009. Pareto paths with SHARC. In *Experimental Algorithms*. 125–136.

Hanif, S.; van Lon, R. R. S.; Gui, N.; and Holvoet, T. 2011. Delegate MAS for Large Scale and Dynamic PDP: A Case Study. In *Intelligent Distributed Computing V*. 23–33.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on* 4:100–107.

Hatzack, W., and Nebel, B. 2014. The operational traffic control problem: Computational complexity and solutions. In *Sixth European Conference on Planning*.

Holvoet, T., and Valckenaers, P. 2007. Exploiting the Environment for Coordinating Agent Intentions. In *Environments for Multi-Agent Systems III*. Springer Berlin Heidelberg. 51–66.

Holvoet, T.; Weyns, D.; and Valckenaers, P. 2009. Patterns of delegate mas. In *Self-Adaptive and Self-Organizing Systems, 2009. SASO'09. Third IEEE International Conference on*, 1–9. IEEE.

Lee, J. H.; Lee, B. H.; and Choi, M. H. 1998. A real-time traffic control scheme of multiple AGV systems for collision free minimum time motion: a routing table approach. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 28:347–358.

Maza, S., and Castagna, P. 2005. A performance-based structural policy for conflict-free routing of bi-directional automated guided vehicles. *Computers in Industry* 56:719–733.

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

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence* 195:470–495.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40–66.

Silver, D. 2005. Cooperative Pathfinding. In *AIIDE*, 117–122.

Standley, T., and Korf, R. 2011. Complete Algorithms for Cooperative Pathfinding Problems. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*.

Suurballe, J. W., and Tarjan, R. E. 1984. A quick method for finding shortest pairs of disjoint paths. *Networks* 14:325–336.

ter Mors, A., and Witteveen, C. 2009. Plan Repair in Conflict-Free Routing. In *Next-Generation Applied Intelligence*. 46–55.

ter Mors, A.; Witteveen, C.; Ipema, C.; de Nijs, F.; and Tsiourakis, T. 2012. Empirical Evaluation of Multi-Agent Routing Approaches. In *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 2, 305–309.

ter Mors, A.; Van Belle, J.; and Witteveen, C. 2009. Context-aware multi-stage routing. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 49–56.

ter Mors, A.; Zutt, J.; and Witteveen, C. 2007. Context-Aware Logistic Routing and Scheduling. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, 328–335.

ter Mors, A. W. 2010. *The world according to MARP: Multi-Agent Route Planning*. Ph.D. diss. Delft: Technische Universiteit Delft.

ter Mors, A. 2011. Conflict-free route planning in dynamic environments. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2166–2171.

Ullrich, G. 2015. Modern Areas of Application. In *Automated Guided Vehicle Systems*. 15–96.

van Lon, R. R. S., and Holvoet, T. 2012. RinSim: a simulator for collective adaptive systems in transportation and logistics. In *Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference on*, 231–232.

van Lon, R. R. S. 2015. RinSim: v4.0.0. doi:10.5281/zenodo.27360.

Vis, I. F. 2006. Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research* 170:677–709.

Wang, K. H. C., and Botea, A. 2008. Fast and Memory-Efficient Multi-Agent Pathfinding. In *ICAPS*, 380–387.

Wang, W., and Goh, W. B. 2011. Spatio-temporal A* algorithms for offline multiple mobile robot path planning. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, 1091–1092.

Wang, W., and Goh, W. B. 2013. Time optimized multi-agent path planning using guided iterative prioritized planning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 1183–1184.

Weyns, D.; Holvoet, T.; and Helleboogh, A. 2007. Anticipatory Vehicle Routing using Delegate Multi-Agent Systems. In *IEEE Intelligent Transportation Systems Conference, 2007. ITSC 2007*, 87–93.

Yen, J. Y. 1971. Finding the K Shortest Loopless Paths in a Network. *Management Science* 17:712–716.