Comparing context-aware routing and local intersection management

Adriaan W. ter Mors Delft University of Technology
Mekelweg 4, Delft, The Netherlands
a.w.termors@tudelft.nl

ABSTRACT

In multi-agent routing, there is a set of mobile agents each with a start location and destination location on a shared infrastructure. An agent wants to reach his destination as quickly as possible, but conflicts with other agents must be avoided. We have previously developed a single-agent route planning algorithm that can find a shortest-time route that does not conflict with any previously made route plans.

In this paper, we want to compare this route planning approach with non-planning approaches, in which intersection agents determine which agent may enter an intersection next, and where the agent will subsequently go (given its destination). When making these decisions, the intersection agents use only locally observed traffic information.

Our experiments show that context-aware routing produces more efficient results in case no incidents disrupt the execution. However, in the face of unexpected incidents, the performance of the intersection management policies proves very robust, while context-aware routing only produces good results when coupled with effective plan repair mechanisms.

Categories and Subject Descriptors

I.2.11 [**Computing methodologies**]: Multiagent systems

General Terms

Algorithms, Experimentation

Keywords

route planning, traffic agents, simulation

1. INTRODUCTION

In this paper we will discuss the problem of multi-agent route planning, in which there are multiple mobile agents each with a start and destination location on a roadmap. The roadmap consists of intersections and lanes connecting the intersections, and each agent wants to find a route that will bring it to its destination as quickly as possible.

In previous work [20], we developed a prioritized route planning approach in which agents are first assigned a priority (typically randomly, or based on their arrival time), and subsequently plan a route that is optimal for themselves and does not create any deadlock with any of the higher-priority agents. We named our algorithm *context-aware routing*, as each planning agent is aware of its context, which is the set of reservations from route plans of higher-priority agents. Deadlock prevention is especially relevant in roadmaps with bi-directional roads that can be traversed in only one direction at the same time (e.g., when the roads are not wide enough for two vehicles to travel side by side), for instance in application domains of automated guided vehicles [9] or airport taxi routing [6].

In this paper, however, we will focus on infrastructures in which all roads are directed, such as common in urban traffic control (cf. [1]), and investigate how different routing approaches influence congestion, and therefore the times the agents reach their destinations. We will compare our conflict-free routing approach with a number of local intersection management policies that we will define in section 4. These intersection management policies make routing decisions for the vehicles only on the basis of information that is local to the intersection, namely how many vehicles are waiting to enter the intersection, and how long they have been waiting.

1.1 Related work

The problem of finding an optimal set of conflict-free route plans is NP-hard [17], and all approaches that guarantee an optimal solution[1] that we know of only manage to find solutions for a handful of agents. In domains with larger numbers of agents, a common approach is to let the agents plan for themselves, usually one after the other, or by iteratively communicating about conflicting plans (cf. [14]). Silver [15], for example, presents an approach that is based on the straightforward idea of letting an agent perform an A* search, in which it checks whether the nodes it visits during search are not occupied by other agents at the time the agent would reach those nodes.

[1]Even for optimal approaches, there are often simplifying assumptions, for example dividing time up into 15-second intervals [3].

A problem with a straightforward A*-with-time approach is that it is unclear how many times a particular node must be visited during the search. The shortest-time path that finds the node unoccupied might not be extendible to the destination node in case all of the node's neighbours are occupied at the time (see also the example in section 3). We must therefore introduce the notion of a *free time window*, which is an interval during which the node is unoccupied. During search, we need only consider the shortest-time paths to each of the free time windows of a node, and then the single-agent routing problem can be solved optimally in polynomial time. Kim and Tanchoco [9] first developed and analysed such an algorithm, with a runtime complexity of $O(|R|^2|\mathcal{A}|^4)$, where R is the set of infrastructure *resources* (lanes and intersections), and $\mathcal{A}$ is the set of agents. Our context-aware route planning algorithm lowered that time complexity to $O(|R||\mathcal{A}|\log(|R||\mathcal{A}|) + |R|^2|\mathcal{A}|)$.

Further reductions in computation time can be achieved in case path and velocity planning are separated. In other words, if an agent first determines a path from start to destination, and then finds a conflict-free schedule along this path. Hatzack and Nebel [6] presented such an approach in an airport taxi routing scenario, whereas Lee et al. [10] consider an automated-guided-vehicle setting, in which agents first determine the k shortest paths between their respective start and destination locations, and then find conflict-free schedules along each of these paths, and choose the quickest. In previous work [19], we compared these *fixed-path scheduling* approaches with context-aware routing, and found that the performance of the former seriously degrades if too many agents plan to make use of the same roads; only if the k alternative routes constitute relevant alternatives can fixed path scheduling outperform context-aware routing.

So far, we have not compared our context-aware routing approach with non-planning approaches because these are either in some way restrictive of the infrastructure or how agents use it (e.g., in case agents are required to traverse the infrastructure in a loop [8]), or because the mechanisms to avoid or prevent deadlocks are time-consuming to run and implement (for instance the Petri-net-based approach from Fanti [5]). In this paper, we restrict ourselves to infrastructures that are less deadlock-prone, so we can compare the efficiency (in terms of global plan quality) of context-aware routing with other approaches.

In *urban traffic control*, most intersection management approaches make use of traffic lights, where the focus is on learning efficient behaviour for individual intersections [1]. Coordination is often limited to neighbouring intersections, although the implementation of higher-level agents to support the decision-making is also considered [2]. Another interesting line of work is that into Automated Intersection Management from the group of Peter Stone (see for instance [4]), in which intersections are not light-controlled, but vehicle agents place reservations for conflict-free trajectories in space and time over the intersection. Up until recently, work had focussed on the operation of a single intersection, but recent work by Hausknecht et al. [7] studies traffic phenomena when multiple intersections are linked together. Vasirani and Ossowski [22, 23] propose a market-based approach, in which intersection managers set prices according to current and future demand, and driver agents adapt their routes based on time and cost considerations. Although inspired by Dresner and Stone's Automated Intersection Management, Vasirani's research is moving from microscopic models, in which vehicle behaviour is affected by the movements of immediate neighbours, to mesoscopic models in which average traffic densities on roads determine traversal speeds. A difference between these urban traffic control approaches and our work (and other planning approaches like it — often originating from Automated Guided Vehicle (AGV) research) is the number of vehicles per intersection; in AGV applications, an intersection can be full with one vehicle (for instance in container terminal domains), while for automated intersection management, the intersections can hold many.

1.2 Contributions and organization

This paper makes two contributions to field of route planning and traffic control:

1. A comparison of the context-aware routing approach with local intersection management policies, both in terms of efficiency (measured in e.g. makespan and sum of agent plan costs) and in terms of robustness, i.e., how well the methods perform when unexpected incidents may disrupt the (planned) execution.

2. The definition of a set of simple local intersection management policies.

In section 2, we first present our model for context-aware routing, and then in section 3 we describe the context-aware route planning algorithm, as well as two plan repair mechanisms that are required when incidents can occur during plan execution. Section 4 presents our intersection management policies, and in section 5 we discuss our experimental results. Section 6 contains the conclusions and the ideas for future work.

2. MODEL

We assume a set $\mathcal{A}$ of agents that each have to find a quickest-time route from one location in the infrastructure to another. We model the infrastructure as a *resource graph* $G_R = (R, E_R)$, where resources in R can be roads, intersections, or interesting locations that the agents can visit. Formally, an agent can directly go from resource $r \in R$ to resource $r' \in R$ if the pair (r, r') is in the *successor relation* $E_R \subseteq R \times R$. A resource r has a capacity $c(r)$, denoting the maximum number of agents that can simultaneously make use of the resource, and a duration $d(r) > 0$ which represents the minimum time it takes for an agent to traverse the resource.

In this paper, we will restrict ourselves to (non-toroidal) *grids*, where two uni-directional lanes connect each pair of adjacent intersections. For these infrastructures, intersection resources have unit capacity and lane resources have capacity 8; minimum traversal times are 2 time units for the intersections and 7 for the lanes. In previous work (e.g. [18]), we have focused on bi-directional lanes, i.e., lanes on which travel in both directions is possible, though *not at the same*

time. In such a setting, however, the local intersection management policies we will evaluate in this paper would cause a deadlock almost instantly.

DEFINITION 1 (DEADLOCK). *Let $A^c = \{A_1, \ldots, A_m\} \subseteq \mathcal{A}$ be a set of agents, and let $R^c = \{r_1, \ldots, r_m\} \subseteq R$ be a set of resources such that, $\forall i \in \{1, \ldots, m\}$, agent A_i is on resource r_i, and $\forall i \in \{1, \ldots, m-1\} : (r_i, r_{i+1}) \in E_R$ and $(r_m, r_1) \in E_R$. The agents A^c are in a* deadlock *if and only if:*

1. *A_i's next resource is r_{i+1} ($\forall i \in \{1, \ldots, m-1\}$, or r_1 if $(i = m)$, and*
2. *$\forall i \in \{1, \ldots, m\}$ the number of agents on r_i equals $c(r_i)$.*

In our context-aware routing approach, deadlocks are prevented by ensuring that agents never make plans that exceed the resource capacities. An agent's plan consists of a sequence of resources, and a corresponding sequence of intervals in which to visit them.

DEFINITION 2 (ROUTE PLAN). *Given a start resource r, a destination resource r', and a release time t, a route plan is a sequence $\pi = (\langle r_1, \tau_1 \rangle, \ldots, \langle r_n, \tau_n \rangle)$, $\tau_i = [t_i, t_i')$, of n* plan steps *such that $r_1 = r$, $r_n = r'$, $t_1 \geq t$, and $\forall j \in \{1, \ldots, n\}$:*

$$meets(\tau_j, \tau_{j+1})\,(j < n) \tag{1}$$

$$|\tau_j| \geq d(r_j) \tag{2}$$

$$(r_j, r_{j+1}) \in E_R\,(j < n) \tag{3}$$

The first constraint states that the exit time of the j^{th} resource in the plan must be equal to the entry time into resource $j+1$. The second constraint requires that the agent's occupation time of a resource is at least sufficient to traverse the resource in the minimum travel time. The third constraint states that if two resources follow each other in the agent's plan, then they must be adjacent in the resource graph. The cost of an agent's plan is defined as the difference between the end time and the release time.

In sequential route planning, an agent must respect the plans of all the agents that came before it. From the set of existing agent plans, we can infer how many agents will be in each of the resources for each point in time.

DEFINITION 3 (RESOURCE LOAD). *Given a set Π of agent plans and the set of all time points T, the* resource load *λ is a function $\lambda : R \times T \to \mathbb{N}$ that returns the number of agents occupying a resource r at time point $t \in T$:*

$$\lambda(r, t) = |\{\langle r, \tau \rangle \in \pi \,|\, \pi \in \Pi \wedge t \in \tau\}| \tag{4}$$

In our approach the resource load represents the information that agents need to know about the plans of the other, higher-priority agents. This is similar to other reservation-based approaches such as from Silver [15], but subtly different from the approach in Nishi et al. [14], in which agents inspect each others' plans, in order to detect conflicts.

An agent may only make use of a resource in time intervals when the resource load is less than the capacity of the resource. In such a *free time window*, an agent can enter a resource without creating a conflict with any of the existing agent plans.

DEFINITION 4 (FREE TIME WINDOW). *Given a resource-load function λ, a free time window on resource r is a maximal interval $w = [t_1, t_2)$ such that:*

$$\forall t \in w : \lambda(r, t) < c(r) \tag{5}$$

$$(t_2 - t_1) \geq d(r) \tag{6}$$

The above definition states that for an interval to be a free time window, there should not only be sufficient capacity at any moment during that interval (condition 5), but it should also be long enough for an agent to traverse the resource (condition 6). Within a free time window, an agent must enter a resource, traverse it, and exit the resource. Because of the (non-zero) minimum travel time of a resource, an agent cannot enter a resource right at the end of a free time window, and it cannot exit the window at the start of one. We therefore define for every free time window w an *entry window* $\tau_{\text{entry}}(w)$ and an *exit window* $\tau_{\text{exit}}(w)$. The sizes of the entry and exit windows of a free time window $w = [t_1, t_2)$ on resource r are constrained by the minimum travel time of the resource: $\tau_{\text{entry}}(w) = [t_1, t_2 - d(r))$, and $\tau_{\text{exit}}(w) = [t_1 + d(r), t_2)$.

An agent that wants to go from resource r to (adjacent) resource r' should find a free time window for both of these resources. By definition 2 of a route plan, the exit time out of r should be equal to the entry time into r'. Hence, for a free time window w' on r' to be *reachable* from free time window w on r, the *entry* window of w' should overlap with the *exit* window of w.

DEFINITION 5 (FREE TIME WINDOW GRAPH). *The* free time window graph *is a directed graph $G_W = (W, E_W)$, where the vertices $w \in W$ are the set of free time windows, and E_W is the set of edges specifying the* reachability *between free time windows. Given a free time window w on resource r, and a free time window w' on resource r', it holds that $(w, w') \in E_W$ if the following two conditions hold:*

$$(r, r') \in E_R \tag{7}$$

$$\tau_{exit}(w) \cap \tau_{entry}(w') \neq \varnothing \tag{8}$$

2.1 Plan execution assumptions

We make the simplifying assumption that vehicles do not require any acceleration or deceleration. That is, an agent can reach its desired speed instantaneously (whether this is its maximum speed or a standstill). In addition, we assume that an agent can see whether there is another vehicle directly in front of it on the road (or on the next road, in case the vehicle is on an intersection). This means that the problem of avoiding collisions is taken care of in the simulator, allowing us to focus on the problems of route planning and/or intersection management for the rest of this paper.

A final simplifying assumption concerns the start and finish locations of the vehicles. We assume that vehicles only

enter the infrastructure once they are granted permission to enter their start resource (an intersection, in our experiments), and leave the infrastructure as soon as the destination resource has been traversed (also an intersection). This assumption can be realistic in application domains such as airport taxi routing (where planes land and take off) and urban traffic, where vehicles enter and exit a city, but less realistic in, e.g., warehousing domains. The routing problem for vehicles that must occupy a location of the infrastructure at all times is sometimes referred to as *cooperative pathfinding* in the literature, and considerable effort must already be spent in finding feasible routes for all the vehicles, let alone efficient ones (cf. [16, 11]).

3. ROUTE PLANNING ALGORITHM

In classical shortest path planning, if a node v is on the shortest path from node s to node t, then a shortest path to v can always be expanded to a shortest path to t. Figure 1 shows that in prioritized multi-agent route planning, it is not the case that a shortest route to an intermediate resource can always be expanded to the destination: we see a blue agent that wants to go to the rightmost resource, and a black agent that has a plan to travel rightwards at least until the middle intersection. At time 1 (indicated by the numbers inside the vehicles), the blue agent might make a reservation for the leftmost intersection (i.e., slotting in just ahead of the black agent without hindering it), and expand this plan to the middle intersection. From the middle intersection, at time 2, it cannot plan to go right, because that road is momentarily full with vehicles. However, the blue agent must vacate the intersection, because the black agent has a reservation to use it. Hence, the earliest plan to the middle intersection can only be expanded in the upwards direction, which is a detour in space, and possibly time depending on how quickly the grey agents will start moving.

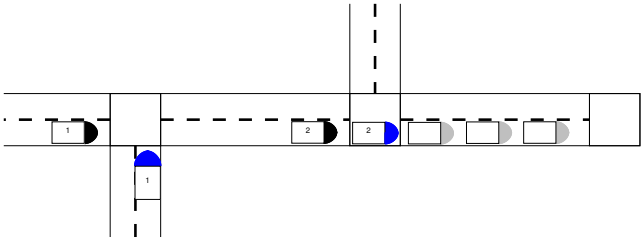


Figure 1: If the blue agent enters the intersection before the black agent, at time 1, then at time 2 it has to drive upwards in order to vacate the intersection for the black agent.

The idea behind our algorithm is that we only need to consider shortest partial plans to the free time windows on a resource: if we have a partial plan that arrives at resource r at time t that lies within free time window w, then all other partial plans to r that arrive at time t', $(t' \geq t) \wedge (t' \in w)$, can be simulated by waiting in resource r from time t to time t'. Waiting is possible because no conflict will ensue as long as the agent exits r before the end of w.

Our route planning algorithm performs a search through the free time window graph that is similar to A*: In each iteration, we remove a partial plan from an open list of partial plans with a lowest value of $f = g + h$, where g is the actual cost of the partial plan, and h is a heuristic estimate of reaching the destination resource. In algorithm 1 below, we will write $\rho(r, t)$ to denote the set of free time windows (directly) reachable from resource r at earliest exit time t.

Algorithm 1 Plan Route

Require: start resource r_1, destination resource r_2, start time t; free time window graph $G_W = (W, E_W)$
Ensure: shortest-time, conflict-free route plan from (r_1, t) to r_2.

```
 1: if ∃w [w ∈ W | t ∈ τ_entry(w) ∧ r_1 = resource(w)] then
 2:     mark(w, open)
 3:     entryTime(w) ← t
 4: while open ≠ ∅ do
 5:     w ← argmin_{w'∈open} f(w')
 6:     mark(w, closed)
 7:     r ← resource(w)
 8:     if r = r_2 then
 9:         return followBackPointers(w)
10:     t_exit ← g(w) = entryTime(w) + d(resource(w))
11:     for all w' ∈ {ρ(r, t_exit) \ closed} do
12:         t_entry ← max(t_exit, start(w'))
13:         if t_entry < entryTime(w') then
14:             backpointer(w') ← w
15:             entryTime(w') ← t_entry
16:             mark(w', open)
17: return null
```

In line 1 of algorithm 1, we check whether there exists a free time window on the start resource r_1 that contains the start time t. If there is such a free time window w, then in line 2 we mark this window as open, and we record the entry time into w as the start time t. In line 5, we select the free time window w on the open list with the lowest value of $f(w)$. As in the original A* algorithm, the function $f(w) = g(w) + h(w)$ is a combination of the actual cost $g(w)$ of the partial plan to w, plus a heuristic estimate $h(w)$ to reach the destination from w. If the resource r associated with w equals the destination resource r_2, then we have found the shortest route to r_2, for the following reason: all other partial plans on open have a higher (or equal) f-value, and if the heuristic is consistent[2], expansion of these partial plans will never lead to a plan with a lower f-value. We return the optimal plan in line 9 by following a series of backpointers.

If r is not the destination, we expand the plan. First, in line 10, we determine the earliest possible exit time out of r as the cost of the partial plan: $g(w) = \text{entryTime}(w) + d(r)$. Then, in line 11, we iterate over all reachable free time windows that are not closed. When expanding free time window w to free time window w', we determine the entry time into w' as the maximum of the earliest exit time out of resource r, and the earliest entry time into w'. We only expand the

[2]Because we make use of a closed list, it is not sufficient to require that the heuristic is merely admissible (i.e., that it would never overestimate the cost of reaching the destination). For a consistent heuristic, it should hold that $h(w) \leq g(w, w') + h(w')$, where $g(w, w')$ is the actual cost of getting from w to w'.

plan from w if there has been no previous expansion to free time window w' with an earlier entry time (initially, we assume that the entry times into free time windows are set to infinity). In line 14, we set the backpointer of the new window w' to the window w from which it was expanded. Then, we record the entry time into w' as t_{entry}, and we mark w' as open. Finally, in case no conflict-free plan exists, we return null in line 17. The worst-case complexity of algorithm 1 is $O(|W|\log(|W|)+|E_W|)$. In case no cyclic plans are allowed, then $|W| \leq (|A|+1)|R|$, and the complexity of algorithm 1 is $O(|\mathcal{A}||R|\log(|\mathcal{A}||R|)+|\mathcal{A}||R|^2)$ (proof in [17]). The worst-case complexity of maintaining the free time window graph G_W is $O(|\mathcal{A}||R|^2)$: for each of at most R reservations of the new plan, one or two new free time windows must be connected to $O(|W| = |\mathcal{A}||R|)$ existing free time windows.

3.1 Plan repair mechanisms

We will now briefly discuss two plan repair mechanisms that can be used to guarantee conflict-free execution for context-aware planners in dynamic environments. The first has been developed by Maza and Castagna [12] and can be considered a baseline approach in the sense that it guarantees conflict-free running without trying to find a repair solution that will result in efficient plan execution. The second is an extension of the first, in which agents can increase their priority over other, delayed agents.

Both plan repair mechanisms rely on the fact that, after all agents have made their plans, it is known for each resource (lane or intersection) in which order the agents are scheduled to visit it. The mechanism of Maza and Castagna is simply to adhere to this *resource priority* (not to be confused with the order in which the agents plan) during plan execution. So, for example, if agent A_1 in figure 2 is delayed, then agent A_2 (and all agents behind it) must wait in case A_1 was the first to enter intersection the middle intersection.

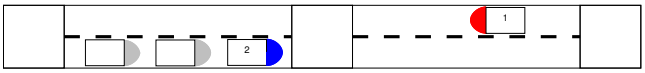


Figure 2: If the red agent A_1 is delayed, then the blue agent A_2 must wait its turn to enter the intersection.

In later work, Maza and Castagna developed a repair mechanism that allowed agents to increase their resource priority over delayed agents in such a way that no new deadlocks were introduced [13]. Note that in our current setting, it is not so obvious why such a change in priorities might lead to a deadlock, but for infrastructures with bi-directional resources, attempting a deadlock-free priority change often involves increasing priority over multiple agents for a whole corridor of resources. The second plan repair mechanism we will employ in this paper is an improvement over the algorithm from Maza and Castagna [13] in the sense that it identifies more deadlock-free priority changes, and also leads to a greater reduction in global delay; see [21].

3.2 Planning shortest paths

We will combine local intersection management policies with agents that follow a shortest path between start and destination locations (for those cases that the intersection does not determine the next road to be taken). In a grid infrastructure, there are many shortest paths (as we assume no cost for turning), so we let each agent construct a random shortest path.

4. INTERSECTION MANAGEMENT

In this section, we will first describe two types of intersection management policies, applied locally at each of the intersections in the infrastructure. The first, most basic type determines which agent is allowed to enter an intersection next, out of the agents ready to enter. The second type of policy then subsequently determines which lane an agent will drive into when it leaves the intersection. Recall from section 3 that a pre-determined path is followed in case only the first type of policy is applied. We now describe the three *intersection entry policies* that we have defined.

DEFINITION 6 (FCFS). *Under* First-Come First-Served *the agent with the earliest* entry request time *may enter first (ties broken arbitrarily); an agent may request entry once it has reached the intersection.*

One should note that an agent cannot request entry when it is waiting behind another agent; only when the agent is first in line can it request entry. The FCFS policy is simple and fair, but it does not take into account congestion formation on the infrastructure.

DEFINITION 7 (LQF). *Under* Longest Queue First*, the agent that forms the head of the longest queue of vehicles waiting to enter, is allowed to enter.*

Longest Queue First (LQF) aims to reduce congestion in the system by reducing the number of vehicles on the fullest of the roads leading into the intersection. In addition to the roads leading into an intersection, another source of vehicles wanting to enter the intersection is formed by those agents that have their starting point at this intersection. However, this set of vehicles is only counted as a queue of length 1; this means that the LQF policy gives precedence to vehicles already on the infrastructure.

DEFINITION 8 (WLQF). *Let t^* be the current time, t_i the time at which agent A_i requested entry to the intersection, and n_i the number of agents on the same road as A_i at time t^*. Under* Weighted Longest Queue First*, the agent that is next to enter is selected according to the formula:*

$$\text{argmax}_{i\in\{1,\ldots,|A|\}}\, n_i + f(t^* - t_i) \qquad (9)$$

for a given function f.

In this paper, we have chosen the function f to divide the argument $t^* - t_i$ by the minimum travel time of the intersection. Hence, when the function f returns a value of, say, 5, then it means that a particular agent has been waiting long enough for (at most) five agents to traverse the intersection since the time it requested entry.

We will now describe an intersection management policy that directs agents to their next lane resource, which we call the Routing Table Approach (RTA), inspired by the way internet routers send packets along their way over the internet. Under RTA, an intersection will select one of at most three outgoing lanes for the next part of the route of an agent, thus not including the direction the agent just came from. When an agent enters an intersection, it announces its destination to the intersection agent, which then computes a value for each of the eligible lanes. Note that the routing table approach only uses information from the lane resources adjacent to the intersection.

DEFINITION 9 (RTA). *Let t^* be the time at which agent A_k, with destination z is ready to leave the intersection, and let $L = \{l_1, \ldots, l_m\}$, $L \subset R$, be the eligible outgoing lane resources, and let $n(l_i)$ denote the number of agents on lane l_i at time t^*. Then the* Routing Table Approach *selects the next lane resource according to the formula:*

$$\text{argmin}_{i \in \{1,\ldots,|L|\}}\, g(l_i, z) + \alpha\left(\frac{n(l_i)}{\sum_{j=1}^{|L|} n(l_j)}\right) \qquad (10)$$

for some constant α and function g that returns the value of the shortest path between its arguments.

In our experiments, we settled on a value of 5.0 for α; by comparison, the maximum difference, in our setting, between the road with the shortest path and the road with the longest path was 4. This means that if only one outgoing lane has vehicles on it, then this lane will not be chosen.

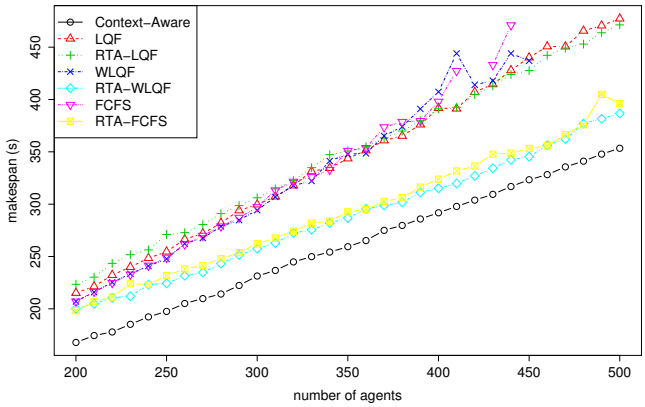
5. EXPERIMENTAL RESULTS

In this section, we describe a set of experiments conducted to compare the performance of context-aware routing to local intersection management strategies. Our principal performance measure is the makespan[3], but we also look at the sum of agent plan costs (where the cost of one agent plan is the time at which it reaches its destination), the distance travelled, and the number of times an intersection management policy will lead the agents into a deadlock.

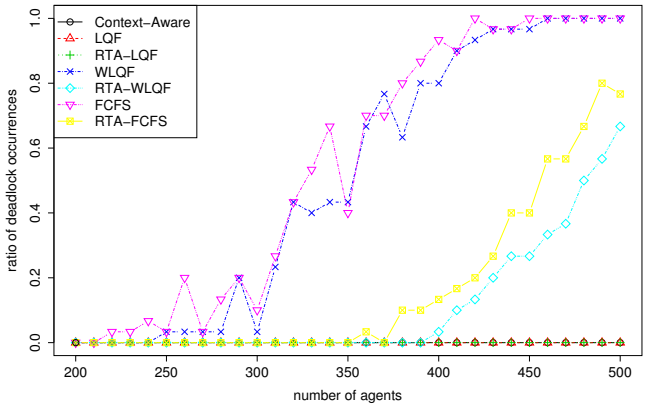
Figure 3 presents the first batch of experiments in which we compared performances for increasing number of agents on a grid infrastructure of five rows and five columns. Each data point in figure 3(a) is the average of 30 runs, or as many as were completed deadlock-free out of those 30 problem instances. The first conclusion we can draw from figure 3 is that context-aware route planning is invariably faster than any of the local intersection management policies. A second conclusion is that the attempt of the routing table approach to reduce congestion (by selecting a next road with congestion in mind) pays off for two out of three intersection *entry* policies; RTA combined with Weighted Longest Queue First seems to be the fastest of the local intersection management policies, although there is not much difference with the basic FCFS entry policy.

[3]All agents have a release time of 0, which means that all agents will either try to obtain a reservation for that time. The makespan is then simply the time at which all agents have reached their destination.

Figure 3(b) shows, however, that RTA-FCFS and RTA-WLQF are not the best from a completeness point of view; from about 350 to 400 agents, an increasing percentages of experiment runs result in a deadlock situation. If only intersection entry management is employed, then from about 300 agents the ability to route agents reduces drastically, in case either First-Come First-Served, or Weighted Longest Queue First is employed. Curiously, when the entry policy Longest Queue First is employed, intersection management has a zero-deadlock rate. This can be explained from figure 5, in which we see a screenshot from the execution of the same instance by RTA-LQF and RTA-WLQF. The main difference is that the latter method, by taking into account the waiting time of a vehicle wanting to enter the infrastructure, will now and then release a new vehicle into the infrastructure even when long queues of vehicles already on the infrastructure have formed at the intersection. The LQF approach, by contrast, will only release a new vehicle when the longest queue of vehicles waiting to enter is at most 1. Hence, using the LQF approach the number of vehicles simultaneously on the infrastructure will be lower, significantly reducing the probability of a deadlock.



(a) makespan



(b) deadlock ratio, with CA, LQF, and RTA-LQF at 0

Figure 3: Performance comparison on (5, 5) grid infrastructure, measured in makespan, and the percentage of deadlock occurrences.

As figure 3 is too cluttered to include confidence intervals, we have plotted the standard deviations for this batch of experiments in figure 4. The spike in the WLQF line (with

the 'x' symbol) is due to the fact that it only managed a handful of deadlock-free runs for 400 or more agents. Overall, we can conclude from figure 4 that context-aware routing is more predictable in its performance than the intersection management policies (RTA-FCFS briefly dips below the Context-Aware line at close to 500 agents, though by that point only around 25% of runs were deadlock free).

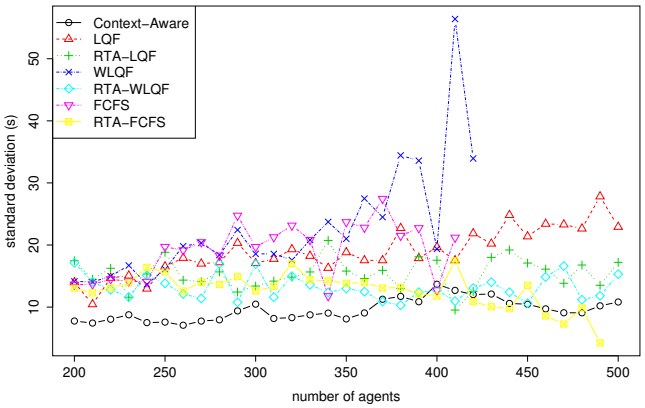


Figure 4: Standard deviations for the experiments of figure 3.

5.1 Cost and distance performance measures

We will now briefly look at the the results of the experiments from different cost perspectives, in figure 6. In figure 6(a) we see the cost per agent divided by the minimum attainable cost (i.e., the cost of traversing the shortest path when no other agents are around), averaged over all agents. This cost measure is a good indicator of the extent agents suffer from the presence of other vehicles, and we see it increases linearly with the number of agents in the system, pretty much regardless of which method is used. Figure 6 shows relative performances that are very similar to those in figure 3 for the makespan measure, although perhaps the best of the intersection-entry-only policies is closer to the best of the RTA policies.

Figure 6(b) shows the distances travelled by each agent (divided by the minimum distance, and averaged over all agents), for each of the methods. For intersection management without RTA, the agents always follow a fixed, and shortest path, so the distance ratio is always 1.0. Agents using context-aware routing have the option of taking a slightly longer route if the shortest one is congested, and this results in routes that are on average 5% longer than the shortest path. The routing table approach has agents travelling the greatest distances, directing agents away from congested areas. If, however, there is no way around the congested area, then it may happen that agents are kept circling in uncongested areas of the infrastructure until the congestion clears.

Another interesting aspect of figure 6(b) is that for RTA-FCFS and RTA-WLQF, the average distance travelled per agent decreases as the number of agents in the system increases. One explanation might be that, as the system becomes heavily congested, the difference between congestion levels on lanes decreases (i.e., if all are very congested). Certainly, if all outgoing lanes are equally congested, then the RTA approach will always select a lane with minimum distance.

5.2 Unexpected incidents

We will now investigate robustness, i.e., the ability of each of the methods to cope with unexpected delays. We will introduce *vehicle incidents* that render vehicles immobile for a fixed period of time. Incidents are generated according to a *rate* parameter, which specifies the average number of incidents per vehicle per time unit. Vehicles can only receive incidents when active, i.e., not before they have entered their start location, and not after they have vacated their destination location (recall the assumption regarding agents and their start and destination locations from section 2.1).

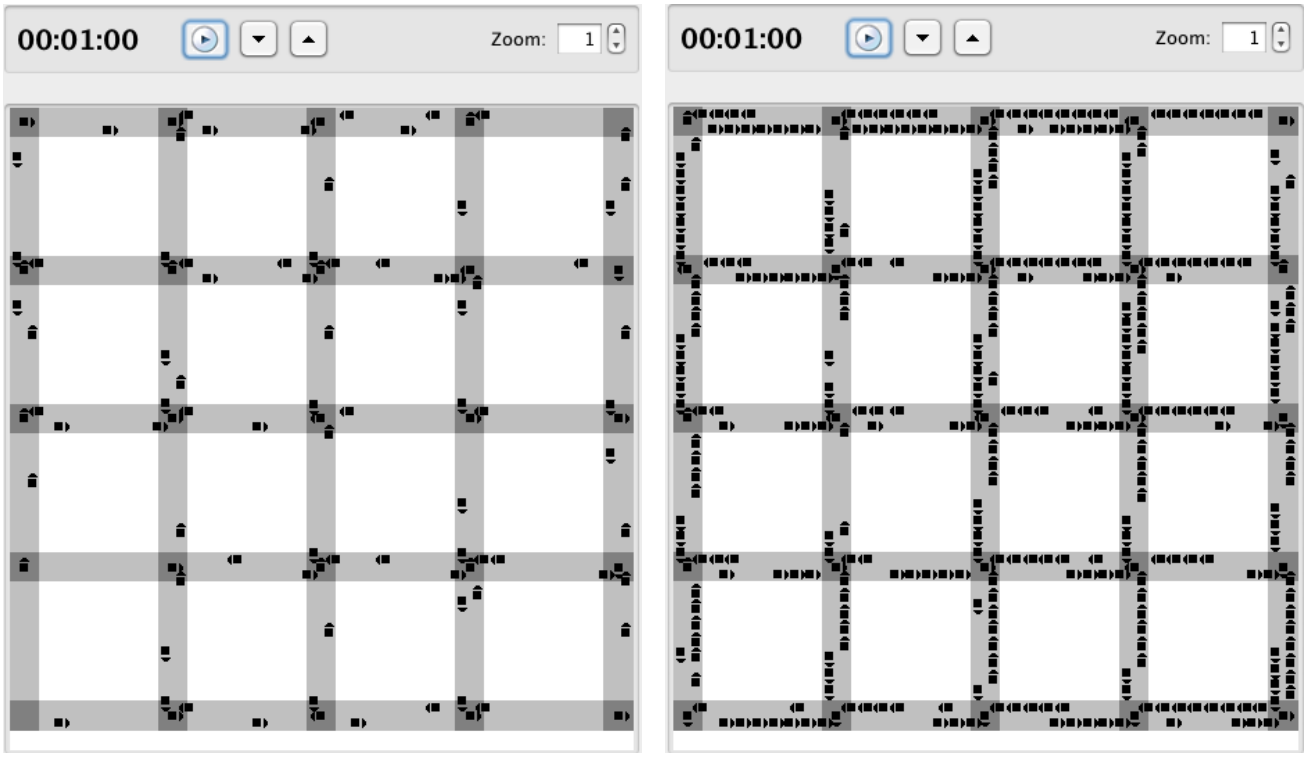
In figure 7, we vary the rate of incidents from 0 to 60 incidents per agent, per hour[4], and we try two different incident durations: 10 seconds per incident in figure 7(a), and 30 seconds for figure 7(b). All incident-experiments were conducted with 400 agents, about the number of agents for which RTA-WLQF is still able to produce a large percentage of deadlock-free runs.

In previous experiments [21, 17, 18], context-aware routing approaches were shown to be fairly robust under incidents of this magnitude, but for these types of infrastructures, standard context-aware quickly loses its advantage, especially for longer incidents. An explanation would be that on this type of grid infrastructure, there is a lot of interaction between the agents on a relatively small number of intersections. This means that if one agent is delayed, many other agents have to wait for it. Increasing the priority with the agent order swap mechanism (CA-AOS in figure 7) restores much of the performance of context-aware routing, although for incidents of longer duration it is now matched by the best intersection management policies. What is also interesting to note from figure 7 is that the local intersection management policies, and in particular LQF, are very robust in the face of vehicle incidents; although figures 7(a) and 7(b) represent different problem instances (i.e., different pairs of start-and-destination locations), it is interesting to see that the makespan barely increases for longer incidents of 30 seconds. Apparently, when one lane of cars is stuck behind a stricken vehicle, an intersection can use that to simply process more vehicles from the remaining lanes.

6. CONCLUSIONS AND FUTURE WORK

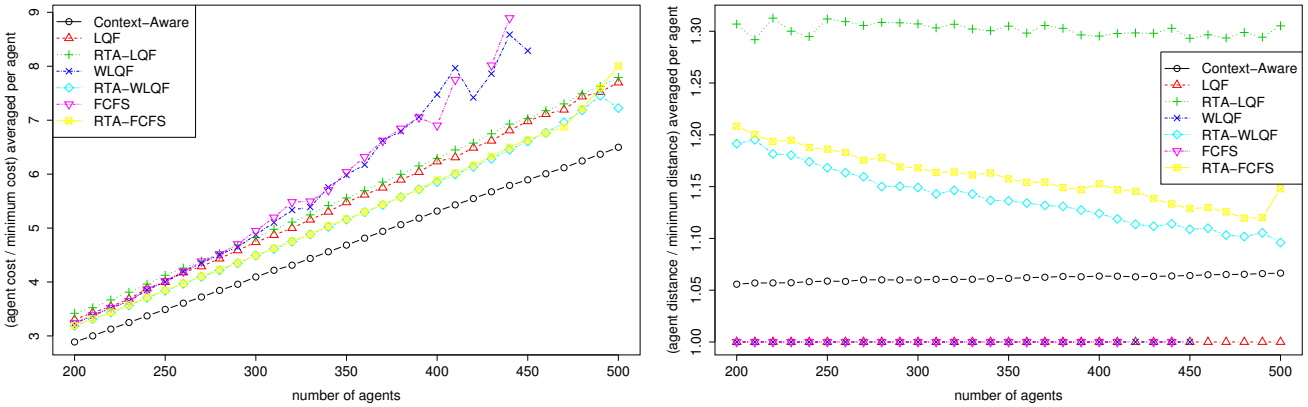
In this paper we compared context-aware routing, in which agents sequentially find locally optimal and conflict-free route plans, with local intersection management, in which an intersection agent decides which vehicle is the next to enter, and possibly directs it along the next lane. Our experiments show that, without any incidents disrupting plan execution, context-aware routing produces the most efficient route plans, when measured in makespan or agent traversal time, while only covering on average 5% more distance than always following the shortest path. Moreover, the most efficient intersection management policies are prone to produce deadlock situations.

[4]To put 60 incidents per agent per hour into perspective, note that the total simulation time equals the makespan, which from figure 7 can be seen to vary from around 200 to 700 seconds.



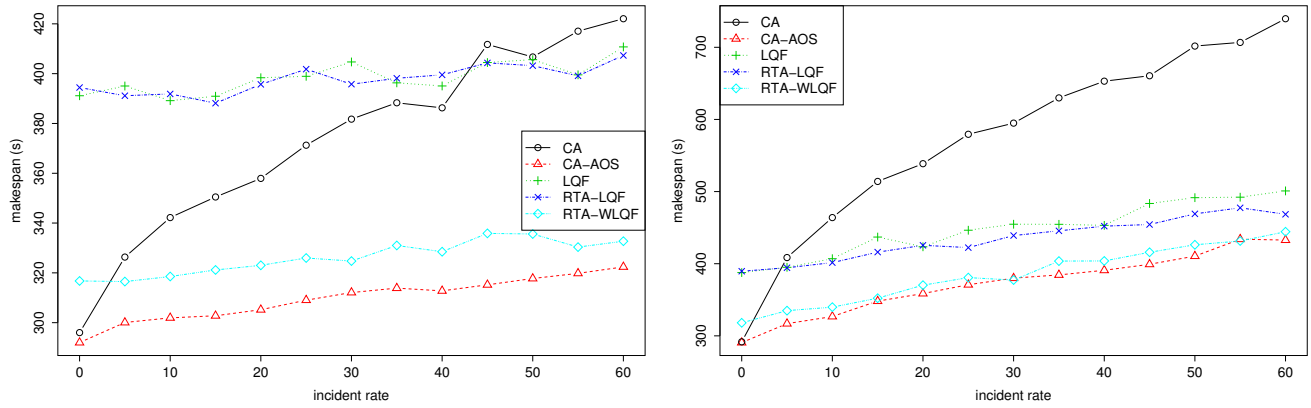
(a) RTA-LQF (b) RTA-WLQF

Figure 5: Execution screenshot of RTA-LQF(a) and RTA-WLQF(b) on the same instance, at one minute into the run.



(a) cost ratio (b) distance ratio

Figure 6: Performance comparison on (5, 5) grid infrastructure, measured in agent cost and distance travelled, divided by a lower bound on cost (and distance), which is the shortest path when other vehicles are not taken into account.



(a) Incident duration = 10s

(b) Incident duration = 30s

Figure 7: Performance comparison on (5, 5) grid infrastructure with 400 agents and makespan performance measure, with unexpected incidents during the execution.

If we do allow unexpected incidents to occur during plan execution, the performance of context-aware routing (with the standard deadlock-prevention mechanism of waiting for delayed vehicles) degrades fairly sharply as many agents end up waiting for one delayed agent — not only directly behind it, but also at an intersection where the delayed agent has failed to appear. The application of the agent order swap mechanism, which increases the priority of timely agents over delayed ones, can return the performance of context-aware routing to a good level. It does mean, however, that context-aware routing needs some kind of plan repair mechanisms in order to be applied in realistic settings. By contrast, the local intersection management policies can be used 'as is' (although there is still the possibility of deadlock, of course).

For future work, there are a number of lines of research we would like to pursue. First of all, we can look into different repair schemes for context-aware routing. The agent order swap mechanism employed in this paper changes the priorities of the agents, but keeps each agent to its originally planned path. Full plan repair, in which an agent computes a completely new route, has been tried in [18] with mixed results. On the one hand, each time an agent successfully makes a new plan it improves its own performance without hindering others (because the new reservations may not conflict with existing ones, adjusted for delays), so full plan repair should be able to improve performance considerably. On the other hand, continual re-planning by all agents has not led to significant global improvement, with agents going back and forth between plans, occasionally covering the same ground multiple times. Hence, a cleverer way of managing the re-planning process is required in order to gain real benefits.

The intersection management policies presented in this paper are of course only a first step in providing intelligent intersection control. One interesting area of possible extensions is to allow limited communication and cooperation between intersection management agents. We could see that the current routing table approach in particular suffered from the limitations of its myopic approach, with agents being directed around congested areas that they had no possibility of avoiding. In addition, when developing policies for intersection management, we must try to find a sensible solution to the possibility of deadlocks. In the Automated Guided Vehicle domain, for instance, a common approach is to model the infrastructure system as a Petri net (cf. [5]), although full deadlock prevention can be computationally expensive in such settings.

7. REFERENCES

[1] Ana L. C. Bazzan. A distributed approach for coordination of traffic signal agents. *Autonomous Agents and Multi-Agent Systems*, 10:131–164, 2005.

[2] Ana L.C. Bazzan, Denise de Oliveira, and Bruno C. da Silva. Learning in groups of traffic signals. *Engineering Applications of Artificial Intelligence*, 23(4):560 – 568, 2010.

[3] Guy Desaulniers, André Langevin, Diane Riopel, and Bryan Villeneuve. Dispatching and conflict-free routing of automated guided vehicles: An exact approach. *International Journal of Flexible Manufacturing Systems*, 15(4):309–331, November 2004.

[4] Kurt Dresner and Peter Stone. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research*, pages 591–656, 2008.

[5] Maria P. Fanti. A deadlock avoidance strategy for AGV systems modelled by coloured Petri nets. In *Proceedings of the Sixth International Workshop on Discrete Event Systems (WODES'02)*, 2002.

[6] Wolfgang Hatzack and Bernhard Nebel. The operational traffic problem: Computational complexity and solutions. In A. Cesta, editor, *Proceedings of the 6th European Conference on Planning (ECP'01)*, pages 49–60, 2001.

[7] Matthew Hausknecht, Tsz-Chiu Au, and Peter Stone. Autonomous intersection management: Multi-intersection optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, pages 4581–4586, San Francisco, CA, USA, September 25–30 2011. IEEE.

[8] Ying-Chin Ho. A dynamic-zone strategy for

vehicle-collision prevention and load balancing in an AGV system with a single-loop guide path. *Comput. Ind.*, 42(2-3):159–176, 2000.

[9] Chang W. Kim and Jose M.A. Tanchoco. Conflict-free shortest-time bidirectional AGV routeing. *International Journal of Production Research*, 29(1):2377–2391, 1991.

[10] Jung H. Lee, Beom H. Lee, and Myoung Hwan Choi. A real-time traffic control scheme of multiple AGV systems for collision-free minimum time motion: a routing table approach. *IEEE Transactions on Man and Cybernetics, Part A*, 28(3):347–358, May 1998.

[11] Ryan Luna and Kostas E. Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[12] Samia Maza and Pierre Castagna. Conflict-free AGV routing in bi-directional network. In *Proceedings of the 8^{th} IEEE International Conference on Emerging Technologies and Factory Automation*, volume 2, pages 761–764, Antibes-Juan les Pins, France, October 2001.

[13] Samia Maza and Pierre Castagna. A performance-based structural policy for conflict-free routing of bi-directional automated guided vehicles. *Computers in Industry*, 56(7):719–733, 2005.

[14] Tatsushi Nishi, Masakazu Ando, and Masami Konishi. Experimental studies on a local rescheduling procedure for dynamic routing of autonomous decentralized AGV systems. *Robotics and Computer-Integrated Manufacturing*, 22(2):154–165, April 2006.

[15] David Silver. Cooperative pathfinding. In *Proceedings of the 1st Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005.

[16] Trevor Standley. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 173–178, 2010.

[17] A. W. ter Mors. *The world according to MARP*. PhD thesis, Delft University of Technology, March 2010.

[18] Adriaan W. ter Mors. Conflict-free route planning in dynamic environments. In *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2166–2171, San Francisco, USA, September 2011.

[19] Adriaan W. ter Mors, Cees Witteveen, Jonne Zutt, and Fernando A. Kuipers. Context-aware route planning. In *Proceedings of the Eighth German Conference on Multi-Agent System Technologies (MATES)*, Lecture Notes in Artificial Intelligence, Leipzig, Germany, September 2010. Springer.

[20] Adriaan W. ter Mors, Jonne Zutt, and Cees Witteveen. Context-aware logistic routing and scheduling. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 328–335, 2007.

[21] A.W. ter Mors and C. Witteveen. Plan repair in conflict-free routing. In Been-Chian Chien, Tzung-Pei Hong, Shyi-Ming Chen, and Moonis Ali, editors, *Proceedings of the The Twenty Second International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems IEA-AIE 2009*, Lecture Notes in Artificial Intelligence, pages 46–55, Berlin, Heidelberg, June 2009. Springer Verlag LNAI. June 24-27, 2009.

[22] Matteo Vasirani and Sascha Ossowski. A market-inspired approach to reservation-based urban road traffic management. In K. Decker, J.S. Sichman, C. Sierra, and C. Castelfranchi, editors, *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, volume I, pages 49–56, Richland, SC, May 2009. IFAAMAS. May 10-15, 2009.

[23] Matteo Vasirani and Sascha Ossowski. A computational market for distributed control of urban road traffic systems. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):313–321, June 2011.