Context-Aware Route Planning

Adriaan W. ter Mors, Cees Witteveen, Jonne Zutt, and Fernando A. Kuipers

Delft University of Technology, The Netherlands

Abstract. In context-aware route planning, there is a set of transportation agents each with a start and destination location on a shared infrastructure. Each agent wants to find a shortest-time route plan without colliding with any of the other agents, or ending up in a deadlock situation. We present a single-agent route planning algorithm that is both optimal and conflict-free. We also present a set of experiments that compare our algorithm to finding a conflict-free *schedule* along a *fixed path*. In particular, we will compare our algorithm to the approach where the shortest conflict-free schedule is chosen along one of k shortest paths. Although neither approach can guarantee optimality with regard to the total set of agent route plans — and indeed examples can be constructed to show that either approach can outperform the other — our experiments show that our approach consistently outperforms fixed-path scheduling.

1 Introduction

Consider a transportation problem in which each agent wants to reach its destination location in the shortest possible time, while avoiding collisions and deadlocks involving other agents. This problem arises in the deployment of Automated Guided Vehicle Systems (AGVSs), for instance in manufacturing where the vehicles carry materials between production stations, or at container terminals such as Hamburg and Singapore, where they carry containers to and from ships [1]. Another application domain of multi-agent transportation is taxi routing at airports [2, 3], where aircraft have to taxi, e.g. from a runway to a gate, while avoiding close proximity with other aircraft.

Avoiding collisions and deadlocks can be achieved by constructing a set of conflict-free route plans[1]. A route plan for a single agent specifies which infrastructure *resources* (such as roads and intersections) the agent will visit, and at which times it will visit these resources. The set of agent route plans should ensure that there are never more agents in a resource than its *capacity* allows. Finding an optimal set of conflict-free route plans is an NP-hard problem [4], and optimal centralized approaches have difficulty finding plans for more than a handful of agents (four agents, in [5]). Fortunately, there exist ways to trade off plan quality for reduced computation times.

[1] Other ways of preventing collisions and deadlocks, such as assigning agents to non-overlapping parts of the infrastructure, are described in the AGV survey paper by Vis [1].

In *context-aware* route planning (CARP), agents plan one after another, with agent n finding an optimal (shortest-time) route that does not create a conflict with any of the $n-1$ existing plans (the context). Kim and Tanchoco [6] presented a context-aware route planning algorithm with a worst-case complexity of $O(|\mathcal{A}|^4|R|^2)$ (i.e., for a single agent), where $\mathcal{A}$ is the set of agents that already have a plan and R is the set of roads and intersections from the infrastructure. A further trade-off between plan quality and computation time can be achieved by finding an optimal *schedule* along a *fixed path*. In fixed-path scheduling (FPS), an agent has one or more pre-determined paths from its start location to its destination location, and it will choose the path along which it can find the shortest-time conflict-free schedule. Hatzack and Nebel [2] presented a fixed-path scheduling algorithm which they applied in an airport taxi routing scenario, with each agent always choosing the shortest path (i.e., the shortest-distance path) from start to destination. Lee et al. [7] suggest finding a conflict-free schedule along one of k shortest paths, determined using Yen's algorithm [8]. The fixed-path scheduling approach cannot guarantee individually optimal route plans, because it may be faster to take a longer but less congested path.

In this paper we present a context-aware routing algorithm with a significantly lower worst-case complexity of $O(|\mathcal{A}||R|\log(|\mathcal{A}||R|) + |\mathcal{A}||R|^2)$. Although no complexity results have been published for fixed-path scheduling approaches, we can convert our own CARP algorithm to an FPS algorithm, and the resulting worst-case complexity is $O(|\mathcal{A}||R|\log(|\mathcal{A}||R|))$. Hence, fixed-path scheduling is faster, but it is not easy to rank both approaches in terms of plan quality, especially if we consider global plan cost, i.e., the cost of a set of agent plans. In this paper we will therefore present a set of experiments that show the performances of both methods on a variety of inputs.

This paper is organized as follows. In section 2, we present a model for the multi-agent route planning problem. The main idea is that the infrastructure is a graph of *resources*, each with a capacity that specifies how many agents may simultaneously occupy a resource. In section 3 we will present our route planning algorithm, which is based on the idea of performing a search through a graph of free time windows. Section 4 describes experiments that try to determine the relative performances of context-aware routing and fixed-path scheduling on a variety of infrastructures, including a realistic airport taxi routing scenario. Finally, section 5 concludes this paper.

2 Model

We assume a set $\mathcal{A}$ of agents that each have to find a quickest-time path 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. 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. An agent's plan consists of a sequence of resources, and a corresponding sequence of intervals in which to visit them.

Definition 1 (Route Plan). *Given a start resource r, a destination resource r', and a start 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 $\forall j \in \{1, \ldots, n\}$: (i) interval τ_j meets interval τ_{j+1} $(j < n)$, (ii) $|\tau_j| \geq d(r_j)$, (iii) $(r_j, r_{j+1}) \in E_R$.*

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 a single agent's plan is defined as the difference between the start time and the end time. For the cost of a set of agent plans, we define two measures. The *makespan* is the difference between the earliest starting time and the latest finish time; the *joint agent plan cost* is simply the sum of the individual agents' plan costs.

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 2 (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 $t \in T$: $\lambda(r, t) = |\{\langle r, \tau \rangle \in \pi \,|\, \pi \in \Pi \wedge t \in \tau\}|$*

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 3 (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: (i) $\forall t \in w : \lambda(r, t) < c(r)$, (ii) $(t_2 - t_1) \geq d(r)$.*

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 (i)), but it should also be long enough for an agent to traverse the resource (condition (ii)). 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 1 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 4 (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: (i) $(r, r') \in E_R$, and (ii) $\tau_{exit}(w) \cap \tau_{entry}(w') \neq \varnothing$.*

The free time window graph encodes the relevant information of the plans of the first $n-1$ agents (allowing agent n to plan its route), but it does not contain any information on the possible movements of agents $n + j$, $j \geq 1$. To ensure that agent n will not make a plan that will make it impossible for any subsequent agent to find a plan, we need to make some simplifying assumptions regarding the start and destination locations of each agent: these locations must either have sufficient capacity to hold all the agents that might need it, or we need to assume that agents arrive and depart from the infrastructure, like airplanes landing on and taking off from an airport.

3 Route planning algorithms

In classical shortest path planning, e.g. using Dijkstra's algorithm, 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. This implies that once we have found a shortest path to v, then no other paths to this node need be considered. In context-aware route planning, it is not the case that a shortest route to an intermediate resource can always be expanded to the destination, as illustrated in figure 1. In figure 1 we see an agent A_1 that wants to go from r_1 to r_5, and an agent A_2 with source-destination pair r_5, r_3. All resources have unit capacity. Let us assume that A_2 has already made a plan, and now A_1 wants to find a plan. If the minimum traversal times of all resources are the same, then A_1 could reach r_2 before A_2 needs it. However, this shortest partial plan to r_2 cannot be expanded, because then the agents would meet head on. Agent A_1 must therefore find an alternative route to r_2, which is to wait in r_1 until A_2 has reached r_3. Hence, multiple route plans to an intermediate resource must be considered. A naive approach that would try all different routes to an intermediate resource would require exponential time to execute. 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'. This waiting is possible because no conflict will be introduced as long as the agent exits r before the end of w.

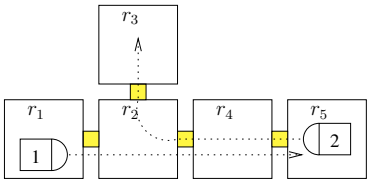


Fig. 1. If A_1 respects the plan of A_2, then the earliest route to r_2 cannot be expanded.

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 plan 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. We cannot directly apply an algorithm like A* to G_W, because the existence of a pair $(w, w') \in E_W$ does not guarantee that a partial plan, ending in w, can be expanded to free time window w'. The reachability of w' from w implies that there exists a time point $t \in (\tau_{\text{exit}}(w) \cap \tau_{\text{entry}}(w'))$, not that *all* time points in $\tau_{\text{exit}}(w)$ are also in $\tau_{\text{entry}}(w')$. Hence, when expanding a plan that ends in window $w = [t_1, t_2)$ at time t to free time window w', we must verify that $[t, t_2) \cap \tau_{\text{entry}}(w') \neq \varnothing$. 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.

if $\exists w\,[w \in W \mid t \in \tau_{\text{entry}}(w) \wedge r_1 = \text{resource}(w)]$ **then**
  mark(w, open)
  entryTime$(w) \leftarrow t$
while open $\neq \varnothing$ **do**
  $w \leftarrow \text{argmin}_{w' \in \mathsf{open}}\, f(w')$
  mark(w, closed)
  $r \leftarrow \text{resource}(w)$
  if $r = r_2$ **then**
    return followBackPointers(w)
  $t_{\text{exit}} \leftarrow g(w) = \text{entryTime}(w) + d(\text{resource}(w))$
  for all $w' \in \{\rho(r, t_{\text{exit}}) \setminus \mathsf{closed}\}$ **do**
    $t_{\text{entry}} \leftarrow \max(t_{\text{exit}}, \text{start}(w'))$
    if $t_{\text{entry}} < \text{entryTime}(w')$ **then**
      backpointer$(w') \leftarrow w$
      entryTime$(w') \leftarrow t_{\text{entry}}$
      mark(w', open)
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 Q 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 resource, we prepare to 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 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 $O(|W| \log(|W|) + |E_W|)$: the while-loop in line 4 runs for at most $|W|$ iterations (every free time window is expanded at most once), and removing the smallest element from a priority queue can be done in $O(\log(W))$ time. All other operations between lines 4 and 10 can be performed in constant time. The for loop in line 11 could inspect every connection between two free time windows exactly once, so lines 12 to 16 can run at most $|E_W|$ times. If we assume that agents are not allowed to make cyclic plans, then one resource can hold at most $|A|$ reservations, and consequently $|A| + 1$ free time windows. Hence, $W \leq (|A| + 1)|R|$, and the complexity of algorithm 1 is $O(|\mathcal{A}||R| \log(|\mathcal{A}||R|) + |\mathcal{A}||R|^2)$, which has been proved in [4], where the correctness is also proved[3].

3.1 Fixed-path scheduling algorithms

Algorithms to find a shortest-time schedule along a fixed sequence of resources can be found in [2] and [7]. It is also possible to use algorithm 1 to schedule along

[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'.

[3] Complexity analysis and correctness proof of an earlier version of algorithm 1 can be found in [9].

a fixed path, by presenting it with a reduced version of the free time window graph. In particular, the set of edges E_W should only contain a pair (w, w') in case the respective resources r and r' are successors in the path along which we want to find a conflict-free schedule. The complexity of running algorithm 1 on such a reduced free time window graph is $O(|\mathcal{A}||R| \log(|\mathcal{A}||R|))$. The reduction in complexity is achieved because a partial plan is only expanded to a single successor resource, rather than considering expansion to all adjacent resources. As a result, the for-loop in line 11 runs only for a single iteration (per partial plan). The while loop from line 4 runs for at most $|\mathcal{A}||R|$ iterations, and none of the lines in the algorithm contribute more than $O(\log(|\mathcal{A}||R|))$ time.

3.2 Examples

We will now present two examples to compare the context-aware approach to the fixed-path scheduling approach. The first example shows how a central resource can become a bottleneck in the fixed-path approach, while the second example demonstrates that a context-aware planner can sometimes select plans that make it harder for subsequent agents to find efficient plans. In figure 2 we see an

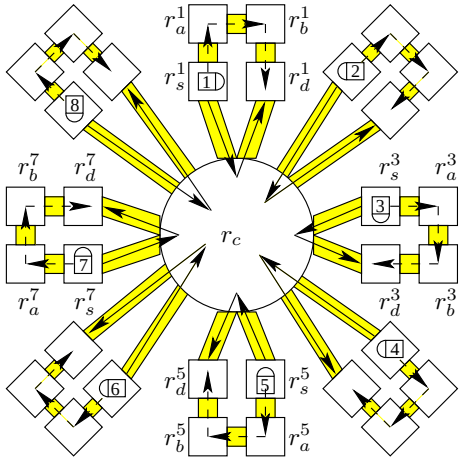


Fig. 2. FPS, with $k = 1$, always makes use of r_c, which leads to congestion.

infrastructure with a central resource r_c, and agents A_i with respective start locations r_s^i and destination locations r_d^i. For each agent, the shortest path from start to destination is via the central resource r_c. Each agent also has the option of taking a path that is one resource longer. A fixed-path scheduling approach (with $k = 1$) will select the shortest path for each agent, resulting in tremendous congestion on the central resource. A context-aware approach will result in one or two agents using r_c, while the other agents will take the alternative route.

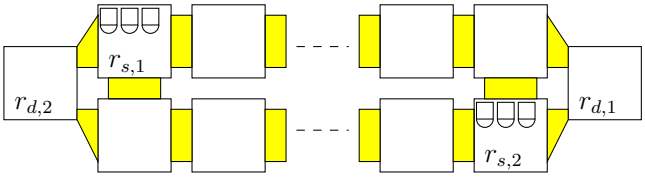


Fig. 3. An optimal multi-agent plan is for all agents to stay in their respective corridors.

In figure 3 we see an infrastructure with two long corridors of resources. Three agents in resource $r_{s,1}$ want to go to resource $r_{d,1}$, while the three agents in resource $r_{s,2}$ want to go to resource $r_{d,2}$. All locations, except for the start and destination locations, have capacity one. The shortest path for each agent is to travel to its destination along its initial corridor. The fixed-path approach with $k = 1$ will therefore direct each agent along its initial corridor, which will result in the optimal multi-agent plan. The behaviour of the context-aware approach depends on the order in which the agents plan. If all agents from one group are allowed to plan first, then for either the second or the third agent it will be fastest to select the other corridor; then all agents from the other group must wait until a corridor is empty.

4 Experiments

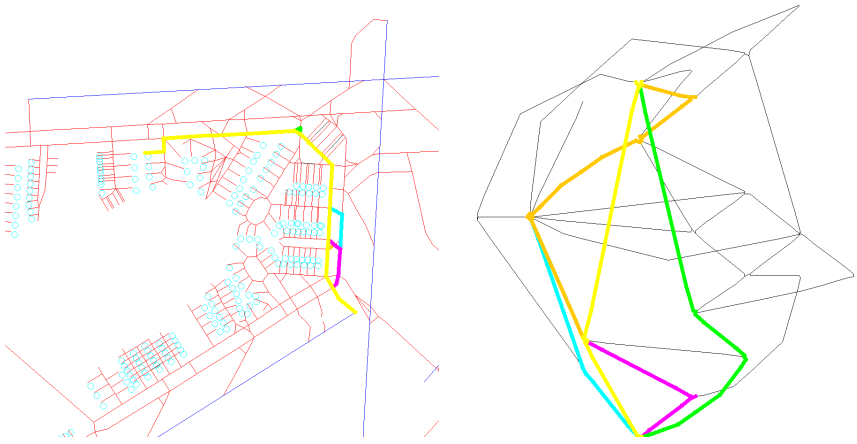
In this section we will compare the global plan cost resulting from k-shortest path scheduling and context-aware routing, and see how they compare to lower bounds on the cost of an optimal global plan. A lower bound on the makespan is the longest of the shortest paths between any of the agents' source-destination pairs, while a lower bound on joint plan cost is the sum of the lengths of the shortest paths between the agents' source-destination pairs.

One problem instance consists of an infrastructure, a set of agents each with randomly chosen start and destination locations, and a random ordering of the agents in which they will plan (in section 3.2, we saw that agent orderings can have an impact on global plan quality). In our experiments we varied the number of agents from 50 to 500, with steps of 50, and for each number of agents 400 different problem instances: 20 different sets of agent start and destination locations, and 20 different agent orderings for each 'task set'.

The first infrastructure we used is a model of Amsterdam Schiphol airport (see figure 4(a)), on which the start location of each agent was a gate (or a runway), and the destination location a runway (or a gate). Of the six runways available at Schiphol, three were randomly chosen to be departure runways, the remaining three arrival runways[4]. There are a total of around 200 gates in the

[4] At Schiphol airport, runways are not operated in *mixed mode*, which is to say they are never used for departure and arrival at the 'same' time.

Schiphol infrastructure, and around 800 taxiway resources, for a total of a little over 1000 resources. We generated two types of infrastructures: lattice networks, which are like grids only with variable-length connections between intersections, and random graphs, which are constructed by first creating a random spanning tree[5], and then adding edges between randomly chosen, as yet unconnected nodes (until the desired number of edges in the graph has been reached). See figure 4(b) for an example of a random graph. The minimum travel times of the resources were determined by setting the length of the median-length resource to 150 meters, and setting the maximum agent speed to 40km/h (as in the airport experiments). In figure 4, we see two examples of the kind of paths that the



(a) Schiphol airport

(b) Random graph on 21 nodes and 39 edges

Fig. 4. The thicker lines indicate the 5 shortest paths between two locations.

k-shortest paths algorithm returns. Figure 4(a) shows a section of the Schiphol infrastructure, and we can see that all five paths have much in common with the shortest paths: two alternative paths take a parallel taxiway, while two other paths make a very small detour. In figure 4(b), we see that the five paths found on a random graph are significantly different from each other, and they might constitute alternatives for an agent that tries to avoid congested roads.

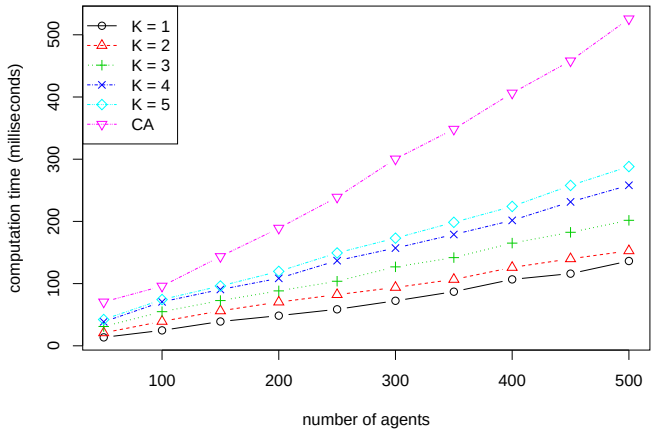
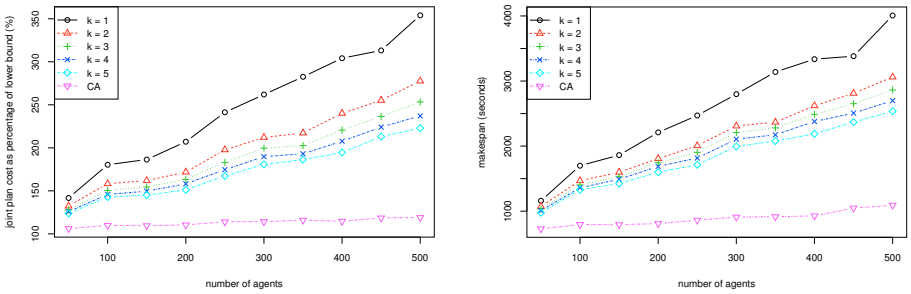


Fig. 5. CPU times for CARP and FPS (k = 1, 2,...,5) on random infrastructures.

4.1 Results

In figure 5, we can see that although fixed-path scheduling is faster, context-aware routing still manages to find plans for all 500 agents within half a second of computation time, when planning on random graphs[6]. On the larger Schiphol infrastructure, finding all 500 agent plans required around 5 seconds. Figure 6



(a) Joint plan cost (b) Makespan

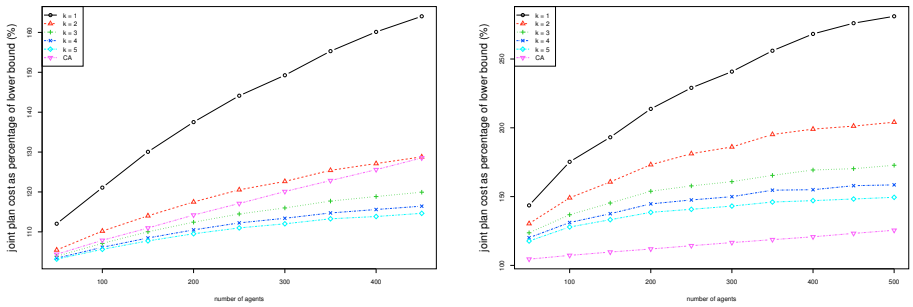
Fig. 6. Global plan cost of CARP and FPS on the Schiphol infrastructure.

shows the results from the comparison between context-aware routing and fixed-path scheduling on the Schiphol infrastructure. The most important conclusion

[5] We create a random spanning tree by iterating through the set of nodes, and in iteration i we connect the node with index i with a randomly chosen node with index smaller than i.

[6] All experiments were run on 4GB dual-CPU 2.4 GHz AMD Opteron machines.

we can draw from figure 6 is that context-aware routing outperforms fixed-path scheduling, for all k between 1 and 5, for both makespan and joint plan cost. Figure 6 also shows that for $k = 1$ (i.e., when agents always choose the shortest path), fixed-path scheduling can perform quite badly. Apparently, if each agent chooses the shortest path, then some resources become overused, resulting in long waiting times, even if alternative routes are available, which a context-aware planner would choose. The reason that for higher values of k fixed-path scheduling still does not approach the performance of context-aware is that, on the Schiphol infrastructure, a standard k-shortest path algorithm does not find useful alternatives, as we can see in figure 4(a). A second conclusion that we can draw is that context-aware routing stays quite close to the lower bounds on global plan cost. For 500 agents, the cost of the multi-agent plan (whether measured in makespan or in joint plan cost) is only 30% more expensive than the lower bounds. Figure 7 shows the results in terms of joint plan costs for the other



(a) Random graphs on 180 nodes and 300 edges.

(b) Lattice graphs of around 450 resources.

Fig. 7. Joint plan cost for CARP and FPS, on lattice and random infrastructures.

types of infrastructures. A quick glance reveals that fixed-path scheduling fares no better on the generated instances. Because of space considerations, we only show the joint plan cost results here, but for the makespan measure the same holds as for the Schiphol infrastructure: fixed-path scheduling performs even worse than for the joint plan cost measure. It seems that fixed-path scheduling is unable to find useful alternative paths, because the nature of Yen's [8] k-shortest path algorithm is such that all shortest paths are found by making minimal deviations from the same shortest path.

5 Conclusions

In this paper we presented our context-aware route planning algorithm, which finds an optimal (shortest-time) route plan that is conflict-free with regard to a

set of existing agent plans. We compared our algorithm to an approach that finds an optimal *schedule* along a fixed path. The advantage of fixed-path scheduling is that it requires less computation time than context-aware routing. In practice, however, this may not be of great importance, as context-aware routing can often find plans for hundreds of agents within a second.

With regard to the global plan cost resulting from the application of either context-aware routing or fixed-path scheduling, in our experiments context-aware routing consistently outperforms fixed-path scheduling. The fixed-path scheduling approach, in which we can choose from one of k shortest paths, seemed to suffer from the fact that the k shortest paths returned by Yen's algorithm [8] (which was also used by the fixed-path scheduling approach of Lee et al. [7]) are too similar. Using the k shortest disjoint paths (cf. [10]) can remove that concern, but there may not always be many disjoint paths.

Given the speed of context-aware routing, we do not believe, however, that trying to revive fixed-path scheduling by finding alternative sets of k paths is the most fruitful direction of future research. Instead, we could focus on determining which routes a context-aware route planner should *not* take. From our examples we know that context-aware route planners sometimes select routes that make it very difficult for subsequent agents to find good plans. In analogy to Stackelberg games (cf. [11]), if the first few agents select routes that are beneficial to others, then subsequent agents may join existing flows of agents on the infrastructure, which might lead to efficient global plans.

References

1. Vis, I.F.: Survey of research in the design and control of automated guided vehicle systems. European Journal of Operational Research **170**(3) (May 2006) 677–709
2. Hatzack, W., Nebel, B.: The operational traffic problem: Computational complexity and solutions. In: ECP'01. (2001) 49–60
3. Trüg, S., Hoffmann, J., Nebel, B.: Applying automatic planning systems to airport ground-traffic control - a feasibility study. In: KI. (2004) 183–197
4. ter Mors, A.W.: The world according to MARP: multi-agent route planning. PhD thesis, Delft University of Technology (March 2010)
5. Desaulniers, G., Langevin, A., Riopel, D., Villeneuve, B.: Dispatching and conflict-free routing of automated guided vehicles: An exact approach. International Journal of Flexible Manufacturing Systems **15**(4) (November 2004) 309–331
6. Kim, C.W., Tanchoco, J.M.: Conflict-free shortest-time bidirectional AGV routeing. International Journal of Production Research **29**(1) (1991) 2377–2391
7. Lee, J.H., Lee, B.H., Choi, M.H.: 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) (May 1998) 347–358
8. Yen, J.Y.: Finding the K shortest loopless paths in a network. Management Science **17**(11) (July 1971) 712–716
9. ter Mors, A.W., Zutt, J., Witteveen, C.: Context-aware logistic routing and scheduling. In: ICAPS. (2007) 328–335
10. Suurballe, J.: Disjoint paths in a network. Networks **4**(2) (1974) 125–145
11. Korilis, Y.A., Lazar, A., Orda, A.: Achieving network optima using Stackelberg routing strategies. IEEE/ACM transactions on networking **5**(1) (1997) 161–173