Context-Aware Logistic Routing and Scheduling

A. W. ter Mors[1,2] and **J. Zutt**[1] and **C. Witteveen**[1]
[1]Delft University of Technology, [2]Almende BV
{A.W.terMors, J.Zutt, C.Witteveen}@tudelft.nl

Abstract

In context-aware route planning, agents have to plan their route on a common infrastructure in such a way that plans made by other agents are not invalidated, and no conflicts are introduced. Previous research on context-aware routing, mostly in the domain of automated guided vehicle (AGV) routing, has reported on an $O(n^4v^2)$ (n the number of vehicles, v the number of infrastructure resources) algorithm. In this paper we present an improved algorithm with a complexity of only $O(nv\log(nv) + nv^2)$.

Our free path routing approach is based on a search through the graph of free time windows on the resources, rather than a search through the infrastructure itself. Our algorithm can be used to find a set of conflict-free routes for a number of agents by finding the route for a single agent at a time. As a consequence, the order in which agents plan their route will determine the quality not only of the individual agent plans, but also of the global plan.

Our experimental results confirm that for an individual agent, its position in the planning queue can make a significant difference; for the total throughput of the airport, however, the order in which the agents make their plans is not highly significant. Also the experiments compare our free path routing approach to fixed path scheduling approaches. We show that for a reasonable amount of extra computation time (required to investigate alternative routes), a free path routing approach finds more efficient plans, because it manages to avoid bottlenecks in the infrastructure.

Introduction

The problem we address in this paper is *context-aware* route planning. In its most general form this problem is about finding a collectively optimal and feasible set of routes for several agents on an infrastructure with limited-capacity resources. Context awareness refers to the fact that an agent has to be aware of the consequences of the route planning by other agents, since his individually optimal route choice might be seriously affected by the route choices of other agents. Examples of applications where this route planning problem plays a role are automated guided vehicle (AGV) routing, scheduling in flexible manufacturing systems (FMS), waterway management, scheduling of transportation vehicles in a container port, and airport taxiing.

Due to the intractability of the general problem and the obvious need for efficient algorithms to solve it, several approaches can be distinguished. One of the most simple approaches to "solve" this problem is to forget the planning aspect and simply to ensure that the individual routes chosen will be conflict-free. An example of such an approach is the use of *social laws* (Shoham & Tennenholtz 1995) which, by constraining the behaviour of the agents, only ensure conflict-free routing without pretending any form of optimality with respect to the achievement of the individual objectives. In this approach conflicts are avoided during execution by following a set of rules.

Other more sophisticated approaches split the problem into a first phase where routes are found for each of the agents individually, and a second phase ensuring feasibility of the collection of routes found in the first phase. An important difference with the first approach is that conflict-resolution is applied before the individual plans are executed. In fact, this conflict-resolution can be considered as a kind of plan repair, modifying individual plans if they are in conflict. For example Broadbent et al. (Broadbent *et al.* 1985) employ a simple shortest path algorithm to find a set of initial routes. In case of *catching-up* conflicts, some vehicles are slowed down; for *head-on* conflicts, an alternative route is found that does not make use of the road at which the conflict occurred. Broadbent's algorithm can be used both on unidirectional and bidirectional infrastructures, but in the latter case it need not find the optimal solution. The approach proposed by Hatzack and Nebel (Hatzack & Nebel 2001) also can be considered as a two-phase approach to this problem. Compared with earlier approaches, they use a more refined model of the common infrastructure by considering parts of the infrastructure (such as lane segments and intersections) as resources having a limited capacity. Once the individual routes have been chosen, conflict resolution can be modeled as a job-shop scheduling problem with blocking to ensure that the constraints imposed by the resources are not violated.

A third type of approach to the problem aims at integrating the route planning and the conflict-resolution process. Typically, such algorithms consider route planning problems using a *free path* of resources from the origin to a destination, taking into account reservations that have been made by other agents using the same infrastructure. For example, the algorithm proposed by Huang et al. (Huang, Palekar, & Kapoor 1993) finds a path through the (graph of) free time windows on the resources, rather than directly through the graph of resources. Huang's algorithm is optimal both for unidirectional and bidirectional networks, but it assumes unit capacity for all resources. Fujii et al. (Fujii, Sandoh, & Hozaki 1989) combine the search through free time windows with a heuristic that calculates the shortest path from the current resource to the destination resource, assuming no other traffic. The solution method proposed should result in an optimal, polynomial-time algorithm, but their description of the algorithm contains a number of ambiguities. Additionally, the authors do not provide any complexity analysis of the algorithm. The work of Kim and Tanchoco (Kim & Tanchoco 1991) is similar to the work of Fujii et al., but their treatment of the problem and the analysis of their algorithm is more comprehensive. Kim and Tanchoco's algorithm finds the (individually) optimal solution for both uni- and bidirectional networks, and they give an $O(n^4v^2)$ time complexity for their algorithm, where n is the number of vehicles in the system, and v is the number of resources in the infrastructure network. Due to this relatively high run-time complexity (especially given the limited computational resources of an early 90s PC), Taghaboni-Dutta and Tanchoco (Taghaboni-Dutta & Tanchoco 1995) developed an approximation algorithm that decides at every intersection to which resource to go next, based on the estimated traffic density of the resources from the current intersection to the destination. The authors show a small loss of plan quality, but they claim that the algorithm consumes significantly fewer computational resources; however, they do not quantify the run-time complexity of the approximation algorithm, nor do they present any CPU time comparisons.

Recent approaches have been focused towards special cases of AGV routing, where additional constraints have to be taken into account. For example, Beaulieu and Gamache (Beaulieu & Gamache 2006) delve into the problem of routing underground mining carts. Their method takes into account the displacement mode of the mining carts, either forward or in reverse, and the carts must be in forward mode when they reach a service point. The authors describe their solution method as a dynamic programming approach, without providing any details on the complexity of their algorithm. Möhring et al. (Möhring *et al.* 2004) present a routing algorithm in which the physical dimensions of the AGV are taken into account: an AGV travelling along one arc may 'spill over' onto a neighboring arc. To avoid conflicts, the authors associate a polygon with each arc to represent the area that an AGV uses when travelling along the arc, and they prohibit the simultaneous use of two arcs if their polygons intersect. The algorithm presented is a generalized arc-based Dijkstra algorithm, which runs in polynomial time if waiting and transit times are the only measure of cost; other cost measures like distance are also supported, but a combination of cost measures results in an exponential time-complexity.

The main contributions of this paper are: (i) an $O(nv\log(nv) + nv^2)$ algorithm that performs a search through the graph of free time windows, based on a concise model of free time windows, and reachability between free time windows. The algorithm finds the optimal conflict-free route on both uni-and bidirectional networks. (ii) An analysis of the free time window graph that allows us to derive the worst-case complexity of the algorithm. (iii) A set of experiments, in which we compare the two-phase approach to route planning — in which first a route is planned, and any conflicts are solved later — to our *free path* approach, where routing and conflict resolution are integrated. For the two-phase approach, we make use of the algorithm by Hatzack and Nebel (Hatzack & Nebel 2001). Our algorithm does not guarantee a globally optimal solution for all route planning agents together and the total performance of the resulting route plans will be dependent on the exact sequence in which the agents will plan their individual route. Therefore, in a second experiment we investigate the effect different sequences of planning agents have on both individual route plans as well as on the total performance of the set of route plans.

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 in terms of a set of *resources* R (Hatzack and Nebel (Hatzack & Nebel 2001) propose a similar resource-based model). Travel from resource r_i to resource r_j is possible if the pair (r_i, r_j) is in the *successor relation* $S \subseteq R \times R$. A resource r_i has a capacity $\mathrm{C}(r_i)$, denoting the maximum number of agents that can simultaneously make use of the resource, and a traversal time $\mathrm{D}(r_i) > 0$ 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 (Agent Plan) An *agent plan* is a sequence $P = ((r_1, \tau_1), \ldots, (r_n, \tau_n))$ of n (resource, interval) pairs such that $\forall j \in [1, \ldots, n-1]$:

1. interval τ_j meets interval τ_{j+1},
2. $|\tau_j| \geq D(r_j)$,
3. $(r_j, r_{j+1}) \in S$. □

The first constraint in the above definition makes use of Allen's interval algebra (Allen 1983)[1], and states that the exit time of the j^{th} resource in the plan must be equal to the entry time into resource $j + 1$.

Reservations and free time windows

To ensure that an agent can perform its plan as intended without interference from other agents, it should make reservations on the resources in its plan for the duration of the intended use. Reserving a resource for a certain interval will use up one unit of the capacity for the duration of the interval. The route planning algorithm we will present in the next section plans around the reservations of other agents

[1]We make use of the *meets* predicate, which means that the end of one interval is equal to the start of the second, and the *precedes* predicate, which means that the end of one interval is earlier than the start of the second.

that have previously made their plans. To plan around the reservations of other agents, we can only make use of the *free time windows* on the resources.

Definition 2 (Free Time Window) Given resource r_i and a bag[2] of reservation-intervals $I = \{\tau_1, \ldots, \tau_m\}$ on resource r_i, a free time window on r_i is an interval $f_{i,v} = [\sigma_{i,v}, \phi_{i,v}]$ such that:

1. $\forall t \in f_{i,v} : |\{\tau_j \in I | t \in \tau_j\}| < \mathrm{C}(r_i)$,
2. $(\phi_{i,v} - \sigma_{i,v}) \geq \mathrm{D}(r_i)$. □

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 1), but it should also be long enough for an agent to traverse the resource (Condition 2). Note that the set of free time windows F_i on resource r_i is a vector $(f_{i,1}, \ldots, f_{i,m})$ of disjoint intervals such that for all $j \in [1, \ldots, m-1]$, $f_{i,j}$ precedes $f_{i,j+1}$.

The route planning algorithm of the next section is based on the idea of going from one free time window on one resource, to another free time window on the next resource. We now define when one free time window can be reached from another.

Definition 3 (Free Time Window Reachability) Given a resource r_i, a free time window $f_{i,v}$ on this resource, and a time t, we say that free time window $f_{j,w}$ on resource r_j is reachable from r_i at time t, denoted $f_{j,w} \in \rho(r_i, t)$, if:

1. $(r_i, r_j) \in S$,
2. $t \in (f_{i,v} \cap f_{j,w})$,
3. $(t - \sigma_{i,v}) \geq \mathrm{D}(r_i)$
4. $(\phi_{j,w} - t) \geq \mathrm{D}(r_j)$ □

The third condition in the definition above ensures that we do not try to leave $f_{i,v}$ at time t until we have had time enough to traverse r_i; the fourth condition requires that there will be enough time to traverse r_j when we enter it from r_i at time t. To express the set of free time windows reachable from $f_{i,v}$, we write

$$\rho(f_{i,v}) = \bigcup_{t \in [\sigma_{i,v} + \mathrm{D}(r_i), \phi_{i,v}]} \rho(r_i, t)$$

Context-aware routing

In Dijkstra's shortest path algorithm, when a node is selected for expansion, we know that the current path to this node is the shortest, and the algorithm does not need to consider any other paths leading to this node. In context-aware routing, on the other hand, the first (and shortest) path to reach a resource is not necessarily the one that will result in the shortest path from the start to the destination, via the current resource. The example in Figure 1 shows why we sometimes need to consider more than one visit to a resource. From the first (and only) free time window on start resource r_s, we can reach both free time windows on resource r_1 (which is on a direct path to the destination resource r_d). However, from the first free time window on r_1, $f_{1,1} = [0, 2]$, we cannot reach any free time window on r_d, because on the destination resource there is a reservation until time 5. Hence, we must go from r_s to r_1 at time 4 (assuming travel times of 1 for all resources, by time 4 we will have had enough time to traverse r_s). Then, we can leave r_1 at time five, entering r_d at time 5, at the start of the free time window on the destination resource.

Algorithm 1 Context-aware routing.

```
Pre: start resource $r_s$, destination resource $r_d$, start time $t_s$.
Post: entry time into $r_d$ for the shortest path from $r_s$ to $r_d$.
 1: if $\exists v\, [f_{s,v} \in F_s \mid t_s \in f_{s,v}]$ then
 2:     $Q \leftarrow \{(r_s, t_s)\}$
 3: while $Q \neq \varnothing$ do
 4:     $(r_i, t_i) \leftarrow \mathrm{argmin}_{(r,t) \in Q}\, t + D(r))$
 5:     $Q \leftarrow Q \setminus \{(r_i, t_i)\}$
 6:     if $r_i = r_d$ then
 7:         return $(r_i, t_i)$
 8:     for all $f_{j,v} \in \rho(r_i, t_i)$ do
 9:         $t_{\text{entry}} = \max(t_i + D(r_i), \sigma_{j,v})$
10:         if constraints_ok$(r_i, r_j, t_{\text{entry}})$ then
11:             $Q \leftarrow Q \cup (r_j, t_{\text{entry}})$
12:             $F_j \leftarrow F_j \setminus \{f_{j,v}\}$
```

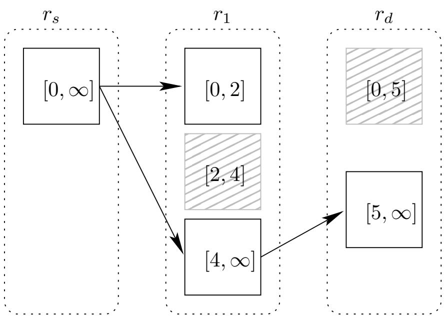

Figure 1: The first arrival at resource r_1, at time 1, will not lead to the shortest path to destination resource r_d. Instead, we must also consider the path that visits r_1 during its second free time window, which starts from 4.

Our context-aware routing algorithm (Algorithm 1) expands a partial plan by looking at which free time windows can be reached, rather than by expanding the plan by reachable resources.

In Line 2, we initialize the open list Q of free time windows to the start resource and the start time[3]. For the simplicity of the specification of the algorithm, we specify partial plans only with (resource, free window entry time) pairs. In the actual implementation of the algorithm, an open list element has a backpointer to the open list element from which it was expanded, which is nil for the initial plan. In Line 4, we retrieve the open list element (r_i, t_i) with the lowest cost $t_i + \mathrm{D}(r_i)$. Hence, the open list elements are

[2]Note that in a bag, we may have $\tau_i = \tau_j$ for $i \neq j$.

[3]Due to condition 1 from Definition 1, we can represent a plan by a vector $((r_1, t_1), \ldots, (r_n, t_n))$ of (resource, resource entry time) pairs.

sorted in order of increasing (minimum) *exit* times.

To expand the current free time window, we consider, in Line 8, all (resource, free time window) pairs that are in $\rho(r_i, t_i)$. The entry time into a reachable free time window $f_{j,v} = [\sigma_{j,v}, \phi_{j,v}]$ is either the entry time into the previous resource r_i plus the time it takes to traverse r_i, or, in case $f_{j,v}$ starts after $t_i + \mathrm{D}(r_i)$, the start time $\sigma_{j,v}$ of $f_{j,v}$.

In Line 10, we check whether we need to take into account any additional constraints with regard to the current expansion candidate. For now, we will simply assume that constraints_ok returns true. In Line 11 we simply add the new element to the open list Q, and, in Line 12, we remove the free time window $f_{j,v}$ from resource r_j's set of free time windows F_j. This is an important step, as it guarantees that we do not consider any free time window for expansion more than once.

Correctness of the Algorithm

The correctness of the algorithm can be shown by considering the algorithm as a modified version of a shortest path algorithm not on the original infrastructure graph (R, S) but on the modified free time window reachability graph (F, ρ). Here, for each free time window $f_{i,v}$ belonging to resource r_i that is reachable from the origin a cost is assigned: the (earliest) time this free time window is entered.

Proposition 1 *Algorithm 1 returns an optimal solution.* $\square$

PROOF: First of all, we prove by induction that for any $k \geq 0$ during the k-th execution of the while-loop algorithm's execution, each pair $(r_i, t_i) \in Q$ represents the earliest time to reach the free time window $f_{i,k}$ such that $t_i \in f_{i,k}$, having started from (r_s, t_s).

Initially, the open list contains only (r_s, t_s), and the induction hypothesis holds for $k = 0$.

Suppose now that after $k \geq 0$ iterations of the while-loop, the pair (r_i, t_i) is retrieved from the open list in Line 4. Let $f_{j,y} \in \rho(r_i, t_x)$, and let $t_x = t_i + \mathrm{D}(r_i)$. Now there are two cases to consider:

case 1: $t_x \leq \sigma_{j,y}$**.** In Line 9, the entry time into $f_{j,y}$ is determined to be $\sigma_{j,y}$. Clearly, the free time window $f_{j,y}$ can be entered no earlier than its start time $\sigma_j y$, so the induction hypothesis also holds for the pair $(r_j, \sigma_{j,y})$ that is added to the queue.

case 2: $t_x > \sigma_{j,y}$**.** The entry time into $f_{j,y}$ will be t_x. To see that no earlier entry time into $f_{j,y}$ is possible, note that $\forall k \neq i, (r_k, t_k) \in Q : t_x \leq t_k + \mathrm{D}(r_k)$. Hence, for any pair (r_k, t_k) such that $f_{j,w} \in \rho(r_k, t_k)$, the entry time into $f_{j,y}$ would be larger than t_x.
A second point to note is that there will be no iteration $m \geq k$ such that a pair (r_m, t_m) can be inserted into Q, such that $t_m + \mathrm{D}(r_m) < t_x$. For all $(r_k, t_k) \in Q$, we have $t_x \leq (t_k + \mathrm{D}(r_k))$, according to Line 4. If a new element (r_m, t_m) is inserted into the open list Q as a result of expanding (r_k, t_k), then $t_m \geq (t_k + \mathrm{D}(r_k))$, and, since travel times are greater than 0, $(t_m + \mathrm{D}(r_m)) > (t_k + \mathrm{D}(r_k)) \geq t_x$.
Hence, there is no earlier entry time possible into window $f_{j,y}$ than t_x, and the pair (r_j, t_x) satisfies the induction hypothesis.

The proposition now follows since in each step of the algorithm, we expand a pair (r_i, t_i) to all free time windows reachable from the free time window determined by (r_i, t_i). Hence, we are guaranteed to find the first entry into the first reachable time window on destination resource r_d. $\blacksquare$

Complexity analysis

In the proof of complexity, we make an assumption regarding the maximum number of free time windows on one resource. Note that the assumption holds in case each agent in the system makes at most one, acyclic plan[4].

Assumption 1 *Each agent makes at most a constant number of reservations per resource.*

Before we discuss the complexity of Algorithm 1, we will first present two propositions that serve to bound the size of the reachability relation ρ. In line 8 of the algorithm, a current free time window is expanded according to the reachability relation ρ. The number of free time windows that can be reached from a particular time window is an important factor in the complexity of the algorithm.

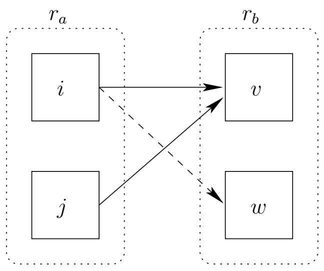

Figure 2: Free time window $f_{b,w}$ on resource r_b is not reachable from $f_{a,i}$, by Proposition 2.

Proposition 2 states that if one free time window $f_{b,v}$ can be reached from two different free time windows $f_{a,i}$ and $f_{a,j}$, on resource r_a, then the first free time window $f_{a,i}$ cannot reach any subsequent free time windows $f_{b,w}$ on resource r_b, because its end time is too early. For an illustration, see Figure 2.

Proposition 2 *[Interval reachability] If* $f_{b,v} \in \rho(f_{a,i})$ *then* $\exists j > i : f_{b,v} \in \rho(f_{a,j})$ *implies* $\neg \exists w > v : f_{b,w} \in \rho(f_{a,i})$ *and vice-versa.* $\square$

PROOF: Let $f_{b,v} \in \rho(f_{a,i})$. We consider two cases.

case 1. if $\phi_{a,i} \leq \phi_{b,v}$ then for every $w > v$, $\sigma_{b,w} > \phi_{a,i}$, so $f_{b,w} \notin \rho(f_{a,i})$;

case 2. if $\phi_{a,i} > \phi_{b,v}$ then for every $j > i$, $\sigma_{a,j} > \phi_{b,v}$, so $f_{b,v} \notin \rho(f_{a,j})$. $\blacksquare$

Intuitively, the number of free time intervals on a resource b that can be reached from free time windows on another resource r_a is bounded by $|F_a| \times |F_b|$. Using Proposition 2, however, we can improve this upper bound considerably.

[4]Kim and Tanchoco (Kim & Tanchoco 1991) assume in their complexity analysis that each agent makes at most one reservation per resource.

Proposition 3 *Given two resources r_a and r_b, and sets of free time windows F_a and F_b, then*

$$\left(\textit{r-sum}(r_a, r_b) = \sum_{i=1}^{|F_a|} |\, \rho(f_{a,i})| \right) \leq (|F_a| + |F_b|) - 1$$

□

PROOF: Let $m = |F_a|$ and $n = |F_b|$. We will prove the proposition with induction on $n+m$. For $n = 1$ and $m = 1$, the result is trivial. Suppose that the property holds for $k = n + m$. Let $k = n + m + 1$. We distinguish the following cases.

- $|F_a| = n + 1$. Let $f_{a,z}$ be the last time window in F_a. Let $O_b \subseteq F_b$ be the set of all time windows having an overlap with $f_{a,z}$ and let $p = |O_b|$. If $p = 0$ then, clearly, r-sum $\leq n + m - 1$. If $p > 0$, let $f_{b,j}$ be the first time window in O_b. By Proposition 2, we have that r-sum is the number of tuples obtained from $(F_a, (F_b - O_b) \cup \{f_{b,j}\})$ plus the number of tuples obtained from $(\{f_{a,z}\}, O_b)$. By induction hypothesis, the first number of tuples is at most $n + (m - p + 1) - 1$ and the second number is at most p. Hence, r-sum$(r_a, r_b) \leq n + (m - p + 1) - 1 + p = (n + 1) + m - 1$.
- $|F_b| = m + 1$. Analogous to the previous case. ■

Proposition 4 *Algorithm 1 has a run-time complexity of* $O(|F| \log(|F|) + |S| \cdot |\mathcal{A}|)$. □

PROOF: In every iteration of the outer while loop (Line 3), one free time window is removed from the open list Q. Because there are $|F|$ free time window in total, the while loop is executed at most $|F|$ times. If we implement the open list Q as a priority queue, removing the smallest element from the list takes $O(\log(|F|))$ time. Lines 1–7 therefore contribute $O(|F| \log(|F|))$ to the complexity of the algorithm.

Rather than looking at lines 8–12 in the context of the outer while loop, we observe that over the whole run of the algorithm, these lines will be executed at most $O(|S| \cdot |\mathcal{A}|)$ times in total: there are at most $|S|$ connections between resources, and for each connection, there are at most $O(|\mathcal{A}|)$ time windows on each of the resources because of Assumption 1. From Proposition 3, we know that there are at most $O(|\mathcal{A}| + |\mathcal{A}| - 1) = O(|\mathcal{A}|)$ time windows that can reach each other. Finally, we note that any free time window is considered at most once for expansion, and so each of the reachable free time windows is also considered at most once. In case there are no special constraints to check, the procedure constraints_ok simply returns true, requiring $O(1)$ time.

Hence, Algorithm 1 has a run-time complexity of $O(|F| \log(|F|) + |S| \cdot |\mathcal{A}|)$.

Special constraints

The procedure constraints_ok can check whether any special, domain-specific constraints need to be satisfied. In this paper we will discuss a constraint (which was first considered by Hatzack and Nebel (Hatzack & Nebel 2001)) that is necessary in some domain to rule out head-on conflicts. Consider the following example situation: we have two resources r_a and r_b of unit capacity, and $S = \{(r_a, r_b), (r_b, r_a)\}$, i.e., travel is possible in both direction. Furthermore, we have agents A_1 and A_2, with the following respective plans: $P_1 = ((r_a, [0, 5]), (r_b, [5, 10]))$ and $P_2 = ((r_b, [0, 5]), (r_a, [5, 10]))$. Note that the union of these plans does not at any time exceed the capacity of the resources, and these plans therefore seem conflict-free.

The problem with these plans is that at time 5, the two agents 'exchange' resources. Unless there is enough space to manoeuver at the intersection of resources r_a and r_b, this attempted resource exchange will result in a head-on conflict. The simultaneous exchange check can be performed by a lookup into a hash table associated with the current resource, in which the keys are times of resource entry and exit, and the values are the number of exchanges at those time points. Updating of the hash table occurs when an agent reserves his plan; the lookup itself requires $O(1)$ time, so taking this constraint into account does not change the run-time complexity of the planning algorithm.

Algorithm variants

There are a number of ways in which Algorithm 1 can be adapted, and we have implemented the following:

Informed A*-search : Our A*-search variant uses a *consistent* heuristic distance function (Dechter & Pearl 1985) h that tries to direct the search earlier towards the goal, resulting in fewer open list operations. The heuristic h we have chosen calculates the length of a shortest path from the current resource to the goal resource assuming no reservations on the resources.

Algorithm 1 has to be adapted as follows: Instead of removing a free time window in Line 12, we should put the window on a *closed list*, just before this window gets expanded. Also, after Line 7, we insert a test to see if the window determined by (r_i, t_i) is on the closed list; if so, we directly continue to the next iteration; otherwise, we put it on the closed list, and proceed with Line 8. Finally, when we are considering successors for expansion, we should check whether a successor already exists on the open list; if it does, we replace this open list entry only if its heuristic value is smaller than that of the open list entry. Replacing an entry on the open list takes $\log(|F|)$ time, so the complexity of the algorithm then becomes $O(|\mathcal{A}||R|^2 \log(|\mathcal{A}||R|))$.

Acyclic plans : In Algorithm 1, we allow agent plans that visit one resource multiple times. To understand why an agent might do that, we can think of the agent as stepping aside to let another agent pass. In many application domains, like airport taxi planning, such actions would be considered too wasteful. In acyclic planning, the size of the algorithm's open list might be reduced if we do not allow cycles. To adapt Algorithm 1 we check after Line 4 whether the current resource has already been visited in the current partial plan (by traversing the parent pointers of the open list elements). With acyclic planning, we no longer have the guarantee that we find the optimal plan.

Experiments

In this section, we will compare our one-phase free path approach to conflict-free routing to a two-phase approach where the *sequence* of resources is fixed, but the time at which we make use of them is not. In particular we compare the performance of the different variants of Algorithm 1 as described above with an implementation of the algorithm by Hatzack and Nebel (Hatzack & Nebel 2001) (H&N) as a representative of the two-phase approach. Secondly, we will look at the effects of the sequence in which the agents plan on the total throughput of the infrastructure. All experiments were run on the infrastructure of Amsterdam Airport Schiphol, where we consider the problem of aircraft taxiing.

We would also have liked to compare our one-phase algorithm to Kim and Tanchoco's (Kim & Tanchoco 1991) one-phase algorithm, but it turned out there was no implementation of their algorithm available, and no previous experimental results have been published that we could find.

One-phase versus two-phase routing

To make a fair comparison between the two approaches, each algorithm was used to calculate a route for the same start-destination pair, given the same set of reservations on the infrastructure. For each set of reservations, we calculated the average time to find a conflict-free path for 20 randomly chosen start-destination pairs. To get an impression of how plan quality and CPU-time depend on the number of reservations, we started with an empty set of reservations. For each set of reservations, we used the last conflict-free plan found to obtain new reservations, added them to the existing set of reservations, and used the new set to calculate again the time for route finding. We repeated this procedure for 3000 iterations. We ran this experiment twice: the first time, we used the plans generated by the A*-search with distance heuristic to make reservations, the second time we used the plans obtained by H&N's algorithm.[5] At all times, we started the algorithms with start time $t_s = 0$.

We conclude from Figure 3 that H&N's two-phase approach is so fast, our one-phase algorithm looks slow by comparison. A closer look reveals that our algorithms are still pretty fast, as a solution is found on average within two tenths of a second. Also, the 95% confidence intervals are reasonably small, so this performance is reasonably stable. With regard to the different variants of our algorithms, we see that the no-cycles variant is significantly faster than the other two, despite the fact that checking for a-cyclicity is not a very cheap operation. Note that the A*-search with distance heuristic requires about the same amount of CPU-time as the standard version of Algorithm 1, which utilizes no heuristic. Apparently, the cost of the closed list operations more or less cancels out the benefits of having an informed search.

Looking at the cost of the generated plans (Figure 4), the plans generated by the no-cycles variant are not significantly more expensive than those generated by Algorithm 1 and

[5]For H&N's algorithm, we set the initial fixed sequence of resources to be the shortest path from start to destination.

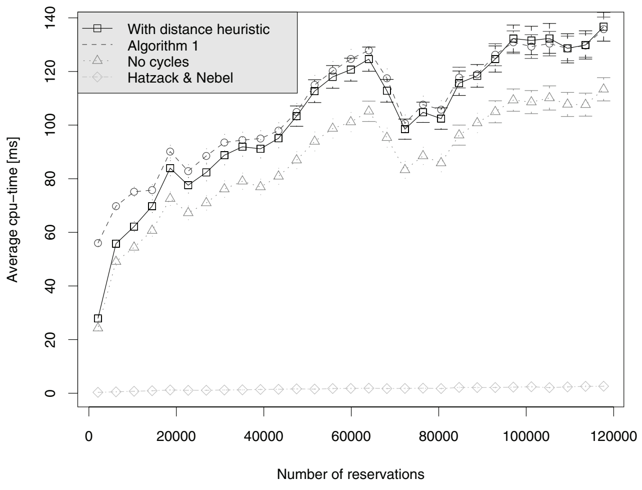

Figure 3: Average CPU-time for increasing amount of resource reservations (of plans generated by A*-search algorithm).

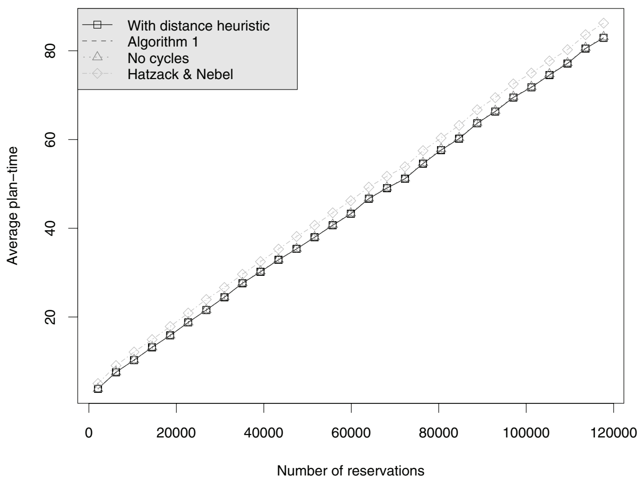

Figure 4: The average end-times of plans for increasing amount of resource reservations (of plans generated by A*-search algorithm).

informed A*-search, both of which are optimal. Note that the plans made by H&N are slightly longer (in time).

In Figures 5 and 6, the results are given using plans made by H&N to make new reservations. Although the aforementioned is still lightningly fast, the plans made by H&N are significantly more expensive. The reason is that many shortest paths will make use of the same resources, so after a while a number of bottleneck resources will emerge, dramatically deteriorating the performance of H&N's algorithm. The free path planners still manage to plan around the bottleneck resources to a large extent, but the search process is slowed down considerably, with an average CPU-time of half a second per shortest path call, and frequent outliers of one or even two seconds for a single shortest path call. The informed A*-search suffers especially, presumably because the distance heuristic, which is based on the shortest path

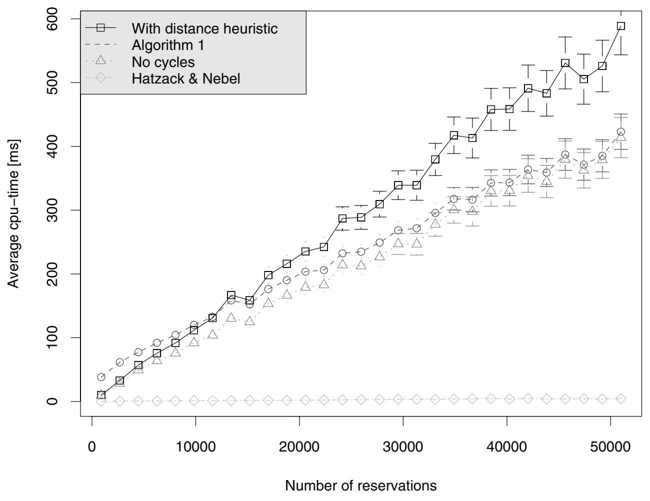

Figure 5: Average CPU-time for increasing amount of resource reservations (of plans generated by Hatzack and Nebel's algorithm).

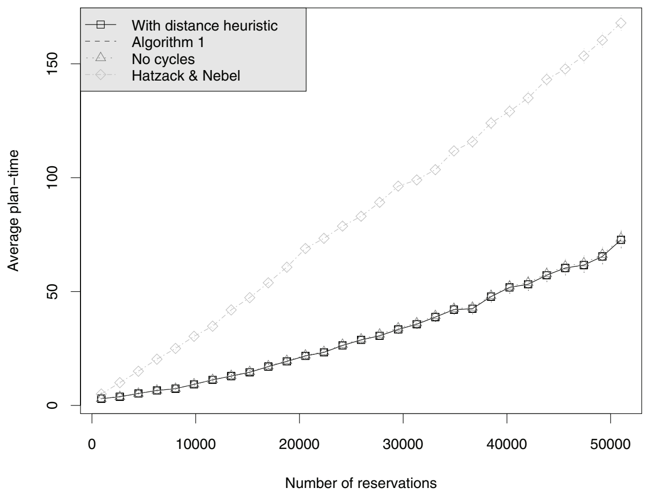

Figure 6: The average end-times of plans for increasing amount of resource reservations (of plans generated by Hatzack and Nebel's algorithm).

without reservations, directs the search right into the congested area of the infrastructure.

One phase vs. two-phase: conclusions Clearly, the advantage of two-phase routing over one-phase routing is its speed. Conflict resolution of one plan requires a few hundredths of a second, compared to maybe a few tenths of a second for a complete route planning call. The plan quality of a two-phase algorithm will depend on the sequence of resources that is the result of the first phase. In the above experiments, we simply chose the shortest path when no reservations are taken into account; and this turned out to be a good choice, *as long as there is an even spread* of reservations over resources, and over time. This spread of reservations is typically achieved if a free path method is

n	Alg.1 [s]	H & N [s]	rec. calls
1	0	0	10
2	0	0	26
4	0	0	110
8	0	10	1,682
16	0	2.600	426,026
20	0	41.460	6,815,798
24	0	696.280	109,051,970
10000	2.260	-	-
20000	12.140	-	-

Table 1: For different problem sizes n, the time required to find a plan by Algorithm 1, by H&N's algorithm, and the number of recursive calls used by the latter.

used to generate plans that are reserved on the infrastructure. However, using the plans generated by H&N's conflict resolution algorithm to make reservations can lead to gross inefficiency.

Finally, we suggest some circumstances favour a two-phase planning approach more than others. In H&N's paper, they consider an airport taxiing problem in which the taxi routes are fixed. Moreover, in their scenario it is reasonable to assume that there is some spread of the agents over time (in the above experiments, all agents were ready to go from time 0). In general, one should determine whether the flexibility afforded by a free path approach is worth the additional CPU-cycles.

Conflict resolution: hard instances Ironically, although their algorithm was shown to be the fastest, the worst-case time complexity of H&N's algorithm is not polynomially bounded. Their algorithm makes use of backtracking, and since they do not make use of the idea that a free time window needs to be expanded at most once, it is possible to construct examples in which the algorithm keeps backtracking through the same paths of time windows.

To create such an instance where their algorithm needs $2^n + 1$ updates, no more than the following $5n$ reservations are needed in a route $r_1, \ldots, r_{3n}$ of $3n$ resources (each having a traversal time equal to 1). Resources

- r_{3i-2} are reserved during $[5i-3, 5i-2)$ for $1 \leq i \leq n$,
- r_{3i} are reserved during $[5i-3, 5i)$ for $1 \leq i \leq n$,
- r_i are reserved during $[5n, 5n+1)$ for $1 \leq i \leq 3n$.

Table 1 clearly shows that if such a structure of reservations occurs H&N's algorithm will not be able to solve the instance within acceptable time.

Planning in sequence From Figure 4 it is already clear why agents would prefer to plan before others make any reservations: the cost of the average plan increases linearly with the number of reservations in the system. Figure 7 shows that if an agent is 400th in line to make a plan, then his plan cost will approximately be twice the cost of the minimum-cost plan, which is the shortest path when reservations are not taken into account.

This does not mean that the order in which airplanes plan makes a difference to the total *system*. Here we show the

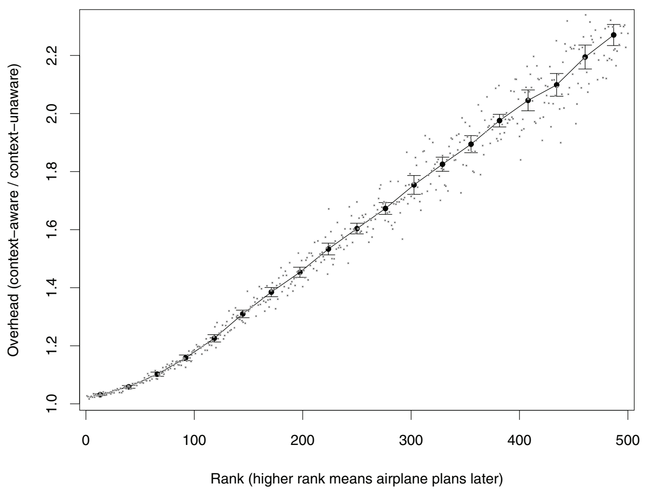

Figure 7: The average overhead (plan cost divided by minimum plan cost) increases when an airplane plans later.

performance of the total system for a group of 500 airplanes routing from and to random resources in the Schiphol infrastructure. The same tasks are repeated 100 times with different random permutations in which the airplanes create their plans and make the reservations.

The relatively small 95% confidence intervals shown in Figure 7 indicate that for the Schiphol airport network the order in which airplanes reserve their plans is not significant; the overhead for being n^{th} in line does not vary greatly.

Conclusions and Future work

In this paper we have presented an optimal single-agent algorithm for the context-aware routing problem, with a worst-case time complexity of $O(|F|\log(|F|) + |\mathcal{A}| \cdot |S|)$, where F is the set of free time windows, and S is the number of connections between resources in the infrastructure (the number of 'edges'). On a realistic airport infrastructure consisting of over 1000 resources, the algorithm finds the optimal plan within a few tenths of a second, even after 3000 agents have reserved their plans on the infrastructure resources (an average day at Schiphol airport has around 1600 flight movements).

To apply the single-agent algorithm in a multi-agent system, the order in which agents are allowed to make their plans will determine the utility of the individual agents, and might therefore be the subject of negotiation between the agents. Fortunately, the order of in which the agents make their plans does not affect the throughput of the infrastructure to a great extent.

In the near future, we will look at the problem of *multi-stage* context-aware routing, in which an agent is looking for the quickest way to visit a (fixed) sequence of resources. In the airport taxiing domain, this can be relevant when an aircraft needs to go by the *deicing station* on its way from the gate to the runway. In a warehouse AGV domain, multi-stage routing can be useful when multiple tasks have to be processed by an AGV, and also in case a visit to the battery charging system must be inserted into its plan.

In the longer run, we intend to explore the possibilities of increasing the number of agents that can plan a route simultaneously, and to resolve any conflicts between the newly made plans in an after-planning negotiation phase.

Acknowledgments

This research has been supported by the TNO-TRAIL project (16) on *Fault detection and recovery in multi-modal transportation networks with autonomous mobile actors*, by the Dutch Ministry of Economic affairs under the SENTER TSIT program on *Cybernetic Incident Management* (TSIT2021), and by NWO (Netherlands Organization for Scientific Research), Grant No. CSI4006.

References

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Commun. ACM* 26(11):832–843.

Beaulieu, M., and Gamache, M. 2006. An enumeration algorithm for solving the fleet management problem in underground mines. *Comput. Oper. Res.* 33(6):1606–1624.

Broadbent, A. J.; Besant, C. B.; Premi, S. K.; and Walker, S. P. 1985. Free ranging AGV systems: promises, problems, and pathways. In *2nd International Conference on Automated Materials Handling*, 221–237.

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality af A*. *J. ACM* 32(3):505–536.

Fujii, S.; Sandoh, H.; and Hozaki, R. 1989. A routing control method of automated guided vehicles by the shortest path with time-windows. In *Production Research: Approaching the 21st Century*, 489–495.

Hatzack, W., and Nebel, B. 2001. The operational traffic problem: Computational complexity and solutions. In Cesta, A., ed., *Proceedings of the 6th European Conference on Planning (ECP'01)*.

Huang, J.; Palekar, U.; and Kapoor, S. 1993. A labelling algorithm for the navigation of automated guides vehicles. *Journal of Engineering for Industry*.

Kim, C. W., and Tanchoco, J. 1991. Conflict-free shortest-time bidirectional AGV routeing. *International Journal of Production Research* 29(1):2377–2391.

Möhring, R. H.; Köler, E.; Gawrilow, E.; and Stenzel, B. 2004. *Conflict-free Real-time AGV Routing*, volume 2004 of *Operations Research Proceedings*. Springer Berlin Heidelberg. 18–24.

Shoham, Y., and Tennenholtz, M. 1995. On social laws for artificial agent societies: Off-line design. 73(1–2):231–252.

Taghaboni-Dutta, F., and Tanchoco, J. 1995. Comparison of dynamic routing techniques for automated guided vehicle system. *International Journal of Production Research* 33(10):2653–2669.