

# Context-Aware Multi-Stage Routing\*

Adriaan ter Mors  
Almende BV  
Rotterdam, The Netherlands  
Adriaan@almende.org

Jeroen van Belle  
Delft University of Technology  
Delft, The Netherlands  
domi1979@hotmail.com

Cees Witteveen  
Delft University of Technology  
Delft, The Netherlands  
C.witteveen@tudelft.nl

## ABSTRACT

In context-aware route planning, a set of agents has to plan routes on a common infrastructure and each agent has to plan a conflict-free route from a source to a destination without invalidating plans made by other agents. The existence of such a conflict-free set of plans can be ensured if each agent is allowed to reserve time slots on the infrastructure resources it intends to use.

In the multi-stage variant of the context-aware routing problem, each agent has a sequence of destination locations it must visit. A naive approach to solve the multi-stage variant is to make context-aware route plans between every two subsequent locations in the sequence, and then to concatenate these plans together. It can easily be shown, however, that this concatenation approach cannot guarantee that a multi-stage plan (if it exists) can always be found, and even if it is found, then it need not be optimal. Therefore, we present a new polynomial-time algorithm for the multi-stage routing problem that always returns the optimal (shortest-time) route for a single agent, given a set of reservations made by previous agents, thus providing a set of Pareto-optimal route plans.

Obviously, the need for such a dedicated multi-stage routing algorithm depends on the frequency with which the concatenation approach fails to find a plan, or finds a rather inefficient one. Our experiments show that, given a set of reservations from 200 agents, the concatenation approach fails to find a solution in more than 50% of the cases, for random visiting sequences of six locations or more. However, if the concatenation approach does find a solution, its plan quality is often close to that of an optimal solution.

## Categories and Subject Descriptors

F.2.2 [Nonnumerical Algorithms and Problems]: Routing and Layout; I.2.11 [Distributed Artificial Intelligence]: Multiagent Systems

\*This research is funded by the Dutch Ministry of Economic Affairs, project number CSI4006.

**Cite as:** Context-Aware Multi-Stage Routing, Adriaan ter Mors, Jeroen van Belle, Cees Witteveen, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 49 – 56  
Copyright © 2009, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org), All rights reserved.

## General Terms

Algorithms

## Keywords

Multi-stage routing, agents

## 1. INTRODUCTION

In context-aware routing [6, 14], there is a set of agents each planning a conflict-free route from a start location to a destination location on a common infrastructure. Each agent must take into account its context, which consists of the plans of other agents that also want to use the infrastructure. As such, context-aware routing is an attempt to resolve possible conflicts *before* the execution of the agent plans, instead of constructing the plans individually, and using a coordination mechanism, such as social rules (cf. [11]), in the plan execution phase. An advantage of the context-aware approach is the avoidance of deadlocks or (unexpected) delays during the execution of the plan and, therefore, an increased predictability in the plan execution phase. Examples of application domains for context-aware routing include route planning for Automated Guided Vehicles (AGVs) in manufacturing, and at container terminals (e.g., in Rotterdam or Singapore), and taxi-route planning at airports.

Context-aware routing problems are solved in a distributed way by allowing agents to reserve time slots on infrastructure resources. Allowing an agent to find an optimal plan that does not invalidate the plans of other agents ensures that the set of agent plans is Pareto optimal. One of the first context-aware algorithms is from Kim and Tanchoco [6]. They presented a single-agent algorithm that finds the optimal (shortest-time) route plan, *given* the set of reservations from previous agents. The complexity of their algorithm was reported to be  $O(|\mathcal{A}|^4|R|^2)$  (where  $\mathcal{A}$  is the set of agents, and  $R$  is the set of infrastructure resources), which was considered to be computationally too expensive by many (cf. [12, 15]). Hatzack and Nebel [5] presented an algorithm that places reservations along the shortest (-distance) path. Their algorithm has a complexity of  $O(|\mathcal{A}||R|)$  but it does not guarantee the optimal solution, and its application can lead to overuse of key resources. A recent paper from Ter Mors et al. [14] presented an optimal algorithm that runs in  $O(|\mathcal{A}||R| \log(|\mathcal{A}||R|) + |\mathcal{A}||R|^2)$  time. The robustness of context-aware routing is evaluated in [9] and [13].

In the literature, alternative approaches exist to context-aware routing. Here, we will discuss only approaches that assume a graph-based infrastructure, which means that agents

can choose different roads to travel on, but they cannot travel in arbitrary directions (cf. [10]). An approach that simplifies the routing problem assigns each agent to a non-overlapping section of the infrastructure (this is sometimes called a tandem configuration) [8]. Within its own area, an agent can simply follow the shortest path, because it will never encounter another agent. The tandem configuration approach has only a limited applicability, however.

A second approach prevents deadlocks at run-time by only allowing an agent to enter the next infrastructure resource if it is safe to do so. Typically, a Petri net [7] is used to model agent entry into resources, and only transitions are allowed that leave the Petri net in a live (i.e., non-deadlocked) state. This approach does not require much additional reasoning in dynamic environments, but it can also be inefficient, and furthermore travel times are unpredictable.

In multi-stage routing, an agent must visit a fixed *sequence* of locations, instead of just a single destination location. The multi-stage routing problem can occur in all of the aforementioned context-aware routing domains, since agents frequently have more than one (routing) task to perform. At airports, for example, wintry conditions sometimes require snow and ice to be removed from wings and fuselage. This means that an aircraft cannot taxi directly from the gate to the runway, because it must first make a stop at a de-icing station, which may be located elsewhere at the airport.

In manufacturing, an AGV may have a sequence of transportation orders to perform, and it must also make the occasional trip to the battery charging station in between orders. Even if an AGV has only a single transportation task, it cannot simply stop moving after delivering its final cargo, because it might get in the way of other agents. Hence, multi-stage routing is also relevant for the idle vehicle positioning problem (cf. [2]).

As far as we know, the multi-stage routing problem hasn't been studied previously in the literature of multi-agent routing. The reason might be that often, the shortest path along a sequence of resources is simply the concatenation of shortest paths between successive resources. However, in context-aware routing with reservations, this concatenation approach might return a non-optimal route, or even no route at all (even though a route does exist), which we will demonstrate in section 3.

Note that a related, but more general, problem that has been studied extensively is the Traveling Salesperson Problem (TSP), in which there is a *set* (i.e., unordered) of locations that must be visited with minimum total cost. However, the generality afforded by the TSP is not required in our intended applications; for instance, in the airport de-icing scenario, an agent need not consider route plans where the aircraft takes off prior to de-icing.

**Organization.** In section 2, we start by discussing the basics of the context-aware routing problem. In section 2.1, we explain the reservation-based approach to routing, which is based on the definition of *free time windows* on resources: time intervals in which the capacity of a resource is not used up by previous reservations. In section 3, we will demonstrate why the concatenation approach to multi-stage routing is neither optimal nor complete, and we present a multi-stage routing algorithm that is both complete and optimal. In section 4, we investigate empirically the extent to which the sub-optimality of the concatenation approach manifests

itself, compared to our optimal multi-stage routing algorithm.

## 2. FRAMEWORK

Context-aware routing is the problem of finding conflict-free start-destination routes for a set of agents that must share a common infrastructure consisting of finite capacity resources. That is, the planned routes of the agents must ensure that there are never more agents in a resource (e.g., a road segment) than its capacity allows. In this section, we will first present a framework to model this problem, followed by a framework that models the solution method that is based on placing reservations on resources.

An *infrastructure* is an undirected graph  $G = (V, E)$ , where  $V$  is a set of vertices representing *intersections*, and  $E \subseteq V \times V$  is a set of edges representing *lanes*. We define a set  $\mathcal{A}$  of agents that can traverse the infrastructure. For each agent  $A_i \in \mathcal{A}$  there is a pair  $(s_i, d_i)$  of locations where  $s_i$  is the agent's start location and  $d_i$  its destination location. In our route planning algorithms, we will treat both lanes and intersections as *resources* that must be traversed by the agents, in non-zero time. Hence, we define the set  $R$  of resources by  $R = V \cup E$ . The function  $C : R \rightarrow \mathbb{N}^+$  associates with every resource  $r \in R$  a capacity  $C(r)$  that specifies the maximum number of agents that can simultaneously occupy a resource. For an intersection resource  $r$  we always have  $C(r) = 1$ . We also define a function  $D : R \rightarrow \mathbb{N}^+$  that gives the minimum travel time of a resource. Note that we specify the set of possible time points by  $\mathbb{N}$ .

From the infrastructure graph  $G$ , we derive a resource graph  $R_G = (R, E_R)$  where for each edge  $e = \{v_1, v_2\} \in E$ , the *resource successor* relation  $E_R$  contains the pairs  $(v_1, e)$ ,  $(e, v_2)$ ,  $(v_2, e)$ , and  $(e, v_1)$ .

Given the resource graph  $R_G$ , we consider the routing problem as a *planning* problem: in order to determine a set of conflict-free plans, agents have to specify exactly for each time point which resource they will occupy.

**DEFINITION 2.1 (AGENT PLAN).** *Given a source, destination pair  $(s, d)$  and a resource graph  $R_G$ , an agent plan is a sequence  $\pi = ((r_1, \tau_1), \dots, (r_n, \tau_n))$  of  $n$   $\langle$  resource, interval  $\rangle$  pairs such that  $r_1 = s$  and  $r_n = d$  and  $\forall j \in \{1, \dots, n-1\}$ :*

1. interval  $\tau_j$  meets interval  $\tau_{j+1}$ ,
2.  $|\tau_j| \geq D(r_j)$ ,
3.  $(r_j, r_{j+1}) \in E_R$

The first constraint in the above definition makes use of Allen's interval algebra [1], and states that the exit time of the  $j^{\text{th}}$  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 infrastructure.

In context-aware routing, agents need to find plans that do not interfere with each other. Only those combinations of agent plans that do not violate any resource capacity constraints should be considered.

**DEFINITION 2.2 (RESOURCE LOAD).** *Given a set  $\Pi$  of agent plans, the resource load  $\lambda$  is a function  $\lambda : R \times \mathbb{N} \rightarrow \mathbb{N}$  that gives the number of agents occupying a resource  $r_i$  at each time point  $t$ :  $\lambda(r_i, t) = |\{(r_i, \tau) \in \pi \mid \pi \in \Pi \wedge t \in \tau\}|$ .*

The single objective in routing that we will consider in this paper is to minimize *completion* time. Hence, we define the cost of an agent plan as the end time of the plan:

**DEFINITION 2.3 (PLAN COST).** *Given an agent plan  $\pi = (\langle r_1, [t_{s_1}, t_{e_1}] \rangle, \dots, \langle r_m, [t_{s_m}, t_{e_m}] \rangle)$ , the cost of the plan is defined as  $c(\pi) = t_{e_m}$ , the end time of the plan. The cost  $c(\Pi)$  of all plans is defined as  $c(\Pi) = \max_{\pi \in \Pi} (c(\pi))$ .*

## 2.1 Free time window graph

We will take a sequential approach to routing, where agents iteratively compute a route plan, and place reservations on the resources for the intended periods of occupation. During planning, an agent is only allowed to use resources in time intervals that do not conflict with the set of existing reservations. We define these allowed intervals as *free time windows*:

**DEFINITION 2.4 (FREE TIME WINDOW).** *Given a function  $\lambda$  for the resource load, a free time window on resource  $r_i$  is a maximal interval  $f_{i,v}$  such that:*

1.  $\forall t \in f_{i,v} : \lambda(r_i, t) < C(r_i)$ ,
2.  $|f_{i,v}| \geq D(r_i)$ .

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  $r_i$  can be represented as a vector  $(f_{i,1}, \dots, f_{i,m})$  of disjoint intervals such that for all  $j \in [1, \dots, m-1]$ ,  $f_{i,j}$  precedes  $f_{i,j+1}$ .

Within a free time window, an agent must enter a resource, traverse it, and exit the resource. Because of non-zero travel times  $D(r_i)$ , an agent cannot enter a resource  $r_i$  right at the end of the free time window, and it cannot exit the window at the start of it. We therefore define for every free time window  $f_{i,v} = [t_s, t_e]$  an *entry window*  $\tau_{\text{entry}}(f_{i,v}) = [t_s, t_e - D(r_i)]$  and an *exit window*  $\tau_{\text{exit}}(f_{i,v}) = [t_s + D(r_i), t_e]$ .

If an agent occupies a resource  $r_i$  during a free time window  $f_{i,v}$ , then it can only travel to a neighbouring resource  $r_j$  if there is a suitable free time window  $f_{j,w}$  that is *reachable* from  $f_{i,v}$ :

**DEFINITION 2.5 (FREE TIME WINDOW REACHABILITY).** *Given a free time window  $f_{i,v}$  on resource  $r_i$ , and a free time window  $f_{j,w}$  on resource  $r_j$ , free time window  $f_{j,w}$  is reachable from  $f_{i,v}$ , denoted by  $(f_{i,v}, f_{j,w}) \in E_F$ , if:*

1.  $(r_i, r_j) \in E_R$ ,
2.  $\tau_{\text{exit}}(f_{i,v}) \cap \tau_{\text{entry}}(f_{j,w}) \neq \emptyset$ .

The reason for condition 2 in definition 2.5 above follows from definition 2.1 of an agent plan: the exit time out of the  $j^{\text{th}}$  resource in the plan must equal the entry time into resource number  $j+1$  in the plan.

The set of free time windows  $F$  together with the reachability relation  $E_F$  form a graph structure. Ter Mors et al. [14] presented an algorithm uses this *Free Time Window Graph FTWG*  $= (F, E_F)$  to find the shortest-time route plan for a single agent, while respecting the reservations of previous agents. The algorithm works as follows: we maintain a list of partial plans, and in each iteration we remove the cheapest partial plan from the list. This partial plan ends in some free time window  $f$ , and we expand it to all reachable free time windows that have not been expanded to before. We have included a specification of this algorithm in the appendix. The algorithm is guaranteed to return the optimal solution, and it has a run-time complexity of  $O(|E_F| + |F| \log(|F|))$ .

## 3. MULTI-STAGE ROUTING

In previous work on context-aware routing (e.g. [5, 6]), agents have always had a start location and a single destination location. A straightforward generalization of this problem is to assume that an agent  $A_i$  should visit a *sequence*  $(v_{i,1}, v_{i,2}, \dots, v_{i,m})$  of locations (resources). A straightforward approach to multi-stage routing is to partition the problem into a sequence of ‘single-stage’ routing problems, and then to *concatenate* the resulting plans. We will begin this section with an example that shows why this approach is incomplete, i.e., it does not always find a solution in route planning with free time windows when there exist one. Then, we will discuss two complete algorithmic approaches to the multi-stage routing problem: (i) an optimal algorithm that fully explores the free time window graph, and (ii) an optimal best-first search algorithm, which is similar to  $A^*$  [4]. First, however, we give a specification of the concatenation algorithm.

---

### Algorithm 1 Multi-Stage Concatenation

---

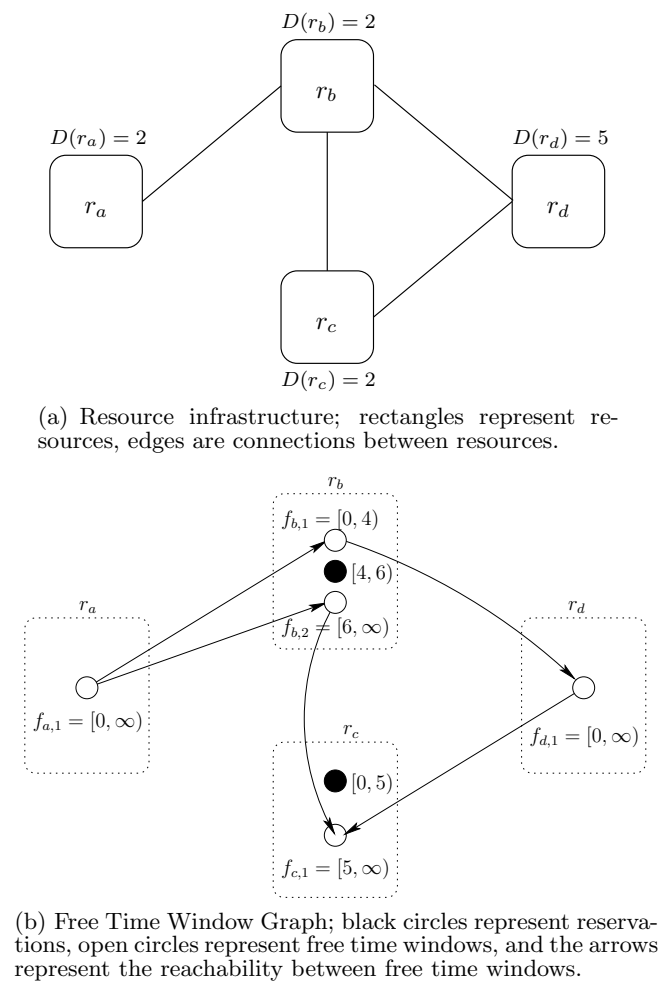
**Require:** visiting sequence  $\sigma = (r^1, \dots, r^m)$ , start time  $t_s$ ;  
 free time window graph  $FTWG = (F, E_F)$ .

- 1:  $\pi \leftarrow \langle r^1, [t_s, t_s + D(r^1)] \rangle$
- 2:  $i \leftarrow 1$
- 3: **while**  $\pi \neq \text{nil} \wedge i < m$  **do**
- 4:      $t \leftarrow c(\pi) - D(r^i)$
- 5:      $\pi_{(i,i+1)} \leftarrow \text{planRoute}(r^i, r^{i+1}, t, FTWG)$
- 6:     **if**  $\pi_{(i,i+1)} = \text{nil}$  **then**
- 7:          $\pi \leftarrow \text{nil}$
- 8:     **else**
- 9:          $\pi \leftarrow \pi \frown \pi_{(i,i+1)}$
- 10:      $i \leftarrow i + 1$
- 11: **return**  $\pi$

---

The concatenation approach to multi-stage routing is incomplete for the following reason: if we are looking for a route from resource  $r_a$  to resource  $r_c$ , then it does *not* hold that the shortest route to an intermediate resource  $r_b$  (i.e., resource  $r_b$  is on all shortest routes from  $r_a$  to  $r_c$ ) can be expanded to the shortest route to  $r_c$ <sup>1</sup>. Instead, it may be necessary to take a slower route to  $r_b$  in order to find the shortest route to  $r_c$ . This is illustrated in the following example.

<sup>1</sup>By contrast, in classical shortest path planning, it does hold that the shortest path to an intermediate node can be expanded to the shortest path to the destination node.



**Figure 1: Resource graph and free time window graph.**

**EXAMPLE 1.** In figure 1, there are four resources  $r_a$ ,  $r_b$ ,  $r_c$ , and  $r_d$ , and there are some reservations on  $r_b$  and  $r_c$ . The travel times of  $r_a$ ,  $r_b$ , and  $r_c$  are 2, and the travel time of  $r_d$  is 5. Suppose that an agent wants to go from  $r_a$  to  $r_c$ . In  $r_a$ , there is only a single free time window, and from that window it can reach both free time windows on  $r_b$ . Obviously, the shortest route to  $r_b$  makes use of the earlier time window. However, since the free time window  $f_{b,1} = [0, 4)$  ends before the start of the only free time window  $f_{c,1} = [5, \infty)$  of resource  $r_c$ , the shortest path to  $r_b$  can only be expanded to  $r_d$ . Traversing  $r_d$  requires 5 time units, so  $r_c$  is entered at time 9. However, the shortest route to  $r_c$  enters  $r_b$  at time 6, the start of the second free time window  $f_{b,2}$ , and then goes directly to resource  $r_c$ , which it enters at time 8.

From this example we conclude that the concatenation approach to multi-stage context-aware routing is sub-optimal: for the visiting sequence  $(r_a, r_b, r_c)$ , the concatenation approach would return the plan

$$\langle r_a, [0, 2) \rangle, \langle r_b, [2, 4) \rangle, \langle r_d, [4, 9) \rangle, \langle r_c, [9, 11) \rangle$$

which is clearly not optimal.

To demonstrate that the concatenation approach is also incomplete, only a small modification to the example is required: if we remove resource  $r_d$  altogether, then the shortest route to  $r_b$ , ending in  $f_{b,1}$ , cannot be expanded at all, so the concatenation approach will not find a plan for the visiting sequence  $(r_a, r_b, r_c)$ .

### 3.1 Multi-stage algorithms

Although the shortest route to an intermediate resource cannot always be expanded, it does hold that if an intermediate *free time window* is on the shortest route to the destination, then the shortest route to that free time window can always be expanded to the shortest route to the destination. The problem is that we do not know *which* free time window on an intermediate resource is on the shortest route to the destination. Of course, if we want to find the optimal multi-stage route, we can simply try them all.

We can perform an exhaustive search of the free time window graph using the following breadth-first search algorithm. From the start location, and the free time window associated with the start time, we make a route plan to the next stage. This route plan will reach the second stage in some free time window  $f$ . We mark  $f$  as visited, and again try to find a route plan from the start location to the second stage, which will make use of a later free time window, if any still exists. We continue until all possible plans from the start location to the second stage have been made.

Next, for every partial plan to stage 2, we try to find a plan to each free time window on stage 3. This results in zero or more partial plans to *each* free time window on stage 3. For all free time windows on stage 3, we take the best (shortest-time) partial plans, and use these to make partial plans to stage 4, and we continue until we have a set of plans to the destination location, and we return the best of these.

The second algorithm we present tries to improve on the breadth-first search algorithm by finding the optimal solution using fewer calls to the context-aware routing algorithm `planRoute` (see algorithm 3). This Multi-Stage Routing algorithm (see Algorithm 2) differs from the aforementioned breadth-first search algorithm in two respects:

1. A partial plan is only expanded with one new plan in each iteration, instead of making plans to all reachable free time windows on the next stage. To guarantee completeness, a partial plan that has only been partially expanded must be put back into the queue of partial plans.
2. In each iteration, we expand the most promising partial plan (in this regard, algorithm 2 is similar to A\* and algorithm 3). A plan  $\pi$  is the most promising iff  $y(\pi) = g(\pi) + h(\pi)$  is minimal over all partial plans in the queue, where  $g(\pi)$  is the cost of partial plan  $\pi$  (i.e.,  $g(\pi) = c(\pi)$ ), and  $h(\pi)$  is a heuristic function estimating the cost of completing  $\pi$  to the destination.

For the heuristic function  $h(\pi)$ , we choose the cost minimal path — without reservations — from the current location to the destination location. Note that this heuristic function never overestimates the true cost of reaching the destination (i.e., it is *admissible*).

In line 2, we make a copy of the free time windows associated with every stage in the visiting sequence  $\sigma$ . These copies are required to keep track of the partial plans that

---

**Algorithm 2** Multi-Stage Routing

**Require:** visiting sequence  $\sigma = (r^1, \dots, r^m)$ , start time  $t_s$ ;  
 free time window graph  $FTWG = (F, E_F)$ .  
**Ensure:** shortest-time plan  $\pi$  from  $\langle r^1, t_s \rangle$  to  $r^m$  s.t.  $\sigma$  is a  
 sub-sequence of resources ( $\pi$ ).

```

1: for all  $r^i \in \sigma$  do
2:    $F_{\text{copy}}^i \leftarrow F_{r^i}$ 
3:  $r_s \leftarrow \sigma(1)$ 
4: if  $\exists v [f_{s,v} \in F_s \mid t_s \in \tau_{\text{entry}}(f_{s,v})]$  then
5:    $Q \leftarrow \{\langle f_{s,v}, r_s, t_s \rangle\}$ 
6: while  $Q \neq \emptyset$  do
7:    $\pi = (\langle f, r^i, t \rangle, T) \leftarrow \operatorname{argmin}_{\pi_x \in Q} y(\pi_x)$ 
8:   if  $r^i = r^m$  then
9:     return  $\pi$ 
10:   $\pi^{i+1} \leftarrow \operatorname{planRoute}(r^i, r^{i+1}, t, (F \oplus F_{\text{copy}}^{i+1}, E_F))$ 
11:   $\pi' = (\langle f', r^{i+1}, t' \rangle, T') \leftarrow \pi \cap \pi^{i+1}$ 
12:  if  $\pi' \neq \text{nil}$  then
13:    if  $\exists \pi'' = (\langle f', r^{i+1}, - \rangle, -) \in Q$  then
14:       $Q \leftarrow Q \setminus \{\pi''\}$ 
15:       $\operatorname{entryTime}(F_{\text{copy}}^{i+1}, f') \leftarrow t'$ 
16:       $f'' \leftarrow \operatorname{nextFreeTimeWindow}(f')$ 
17:       $y(\pi) \leftarrow g(r^{i+1}, \operatorname{start}(f'')) + h(r^{i+1})$ 
18:       $Q \leftarrow Q \cup \{\pi, \pi'\}$ 
19: return nil
    
```

---

have been created for each stage. Note that if  $\sigma$  contains duplicate resources, then we also create duplicate copies of the free time windows.

In line 4, we find the free time window on the start resource such that the entry window contains the start time  $t_s$ . Line 5 adds the first element to the open list  $Q$ . An element of the open list is a partial plan to a certain stage  $r^i$ . We represent a partial plan as a tail end of the plan  $T$ , and the ‘head’ of the plan consisting of a tuple  $\langle f, t, r^i \rangle$ , where

- $f$  is a free time window;
- $t$  is the time at which the resource is entered (such that  $t \in \tau_{\text{entry}}(f)$ );
- $r^i$  is the resource of stage  $i$ .

In line 7, we remove the minimum-value element from  $Q$ . In this case, the elements in  $Q$  are ordered in increasing value of  $g(r, t) + h(r)$ .<sup>2</sup> If the minimum-value element has reached the destination stage, then we can return this plan. If the final stage has not been reached, then the minimum-value element will be expanded to the next stage. Expansion of element  $\pi$  in line 10 amounts to finding the shortest context-aware plan from the current stage (resource  $r^i$ ) to the next stage, denoted as  $r^{i+1}$  (the index  $i$  refers to the position in the vector  $\sigma$ ). The final argument,  $(F \oplus F_{\text{copy}}^{i+1}, E_F)$ , requires some explanation.

In the set  $F_{\text{copy}}^{i+1}$  we mark the entry times of partial plans into the stage  $r^{i+1}$ . Given a free time window  $f$ , the effect of a marking with time stamp  $t$  is that the (single-stage) route planning algorithm will not make use of this free time window, unless it can reach  $f$  at a time earlier than  $t$  (see

<sup>2</sup>In general, the functions  $g$  and  $h$  are defined on plans; here  $g(r, t)$  is shorthand for the cost of the plan that enters the resource  $r$  at time  $t$ , whereas  $h(r)$  is the estimated cost of completing a plan that ends in resource  $r$ .

line 10 of algorithm 3). The expression  $F' = F \oplus F_{\text{copy}}^{i+1}$  indicates that a set  $F'$  of free time windows is created in which the markings of  $F_{\text{copy}}^{i+1}$  are added to the corresponding free time windows in  $F$  (note that the set  $F$  contains no other markings). The markings should ensure that a plan is found that:

1. makes use of a ‘non-discovered’ free time window on  $r^{i+1}$ , or
2. makes use of a previously-visited free time window, but arrives at an earlier time within this free time window.

Note that when making a plan from stage  $r^i$  to stage  $r^{i+1}$ , the single-stage routing algorithm only takes into account the markings of stage  $r^{i+1}$ , not of any other stage  $r^j$ , even if stages  $i+1$  and  $j$  correspond to the same resource.

Having found a plan  $\pi^{i+1}$  from stage  $r^i$  to  $r^{i+1}$ , we concatenate  $\pi^{i+1}$  with the plan  $\pi$  from  $r_s$  to  $r^i$ . In line 15, we mark the free time window  $f'$  (reached by the new plan  $\pi'$ ) by the entry time  $t'$ .

Finally, we have to insert the plan  $\pi$  back into the open list  $Q$ , because we have not fully expanded this plan yet. However, before we put the plan back, we update its ‘ $y$ -value’ to avoid expanding the same plan in the next iteration. The next time we expand  $\pi$ , we will find a plan that is more expensive than  $\pi'$ , because no expansion of plan  $\pi$  can make use of free time window  $f'$  anymore. In line 17, we determine the new  $y$ -value based on the start time of the next free time window  $f''$  on  $r^{i+1}$ . In line 18, we add both the new plan  $\pi'$  and the updated plan  $\pi$  to the open list.

### 3.1.1 Correctness of algorithm 2

We must prove that if a solution to the multi-stage routing problem exists, then algorithm 2 finds the optimal solution.

**PROPOSITION 1.** *Algorithm 2 returns the optimal solution.*

**PROOF.** Algorithm 2 is optimal for the same reason that a standard A\* algorithm is: in each iteration the most promising plan is expanded, and it is not possible that expansion of a plan  $\pi$  results in a plan  $\pi'$  such that  $y(\pi') < y(\pi)$ . What remains to be proven is the correctness of:

1. removal of partial plans from the queue in line 14,
2. putting partial plans back into the queue in lines 17 and 18.

**Ad 1:** In line 13, we check whether a plan exists in  $Q$  that has the same combination of stage and free time window as the plan that was found by algorithm 3 in line 10. If such a plan  $\pi''$  exists, then it *must* be that  $c(\pi'') > c(\pi')$ : the contrary would imply that the entry time of  $\pi''$  into  $r^{i+1}$  (and therefore into  $f'$ ) were smaller than the entry time of  $\pi'$  into  $r^{i+1}$ . But algorithm 3 can only find the plan  $\pi'$  that makes use of  $f'$  if the entry time of  $\pi'$  into  $f'$  is smaller than any previously recorded entry time (line 10 of algorithm 3). Hence, the entry time of  $\pi''$  into  $f'$  must be larger, and we can safely discard  $\pi''$ , since the set of plans that can be expanded from  $\pi''$  is a subset of the plans that can be expanded from  $\pi'$ .

**Ad 2:** In line 18, we place an expanded plan  $\pi$  back into  $Q$ , to ensure completeness. However, we first update the value of  $y(\pi)$  to reflect that the next time  $\pi$  will be

expanded, it will result in a more expensive plan. To guarantee optimality, it must hold that the new value of  $y(\pi)$  does not *overestimate* the cost of completing  $\pi$  to the final stage. There are two reasons why this holds: (i) the next time  $\pi$  will be expanded, it cannot find a plan into a time window earlier than  $f'$ , or earlier in  $f'$  than  $t'$ , because  $\pi'$  is the optimal plan from  $\pi$  to the next stage  $\tau^{t+1}$ ; (ii) the next best plan cannot reach the next stage earlier than the start of the next free time window  $f''$ .

We conclude that algorithm 3 is correct, and because of the ‘A\* property’, it is optimal.  $\square$

### 3.1.2 Algorithm complexity

The computational complexity of algorithm 2 is determined by the number of calls that are made to algorithm 3.

**PROPOSITION 2.** *Algorithm 2 has a run-time complexity of  $O(|F|^2 \cdot (|E_F| + |F| \log(|F|)))$ .*

**PROOF.** The main while loop that spans lines 6 to 18 is executed at most  $O(|F|)$  times. Each free time window from every stage (except the last one) can be expanded, in the worst case, to every free time window on the next stage, which results in  $O(|F|)$  calls to algorithm 3, the single-stage routing algorithm.

Hence, there are  $O(|F|^2)$  calls to algorithm 3, which has a complexity of  $O(|E_F| + |F| \log(|F|))$ .  $\square$

## 4. EXPERIMENTS

In section 3, we showed that there exist instances where the plan-concatenation approach to multi-stage routing fails to find an optimal solution, or even any solution at all. In this section, we will investigate empirically (i) how often the concatenation approach fails to find a plan, (ii) how often it finds a sub-optimal plan, and (iii) if concatenation finds a sub-optimal plan, how much more expensive it is than an optimal plan. We will also investigate the CPU time required by the optimal algorithm (algorithm 2), and how the CPU time depends on the length of the visiting sequence.

### 4.1 Test procedure

We tested the two algorithms on two different infrastructures. The first infrastructure is a random graph with 280 resources. For each agent a randomly generated visiting sequence  $\sigma$  was determined. The visiting sequence size  $|\sigma|$  was set to 4, 6 and 8.

To investigate how multi-stage routing performs in a real-life application, we looked at the de-icing problem at an airport (see introduction). Our infrastructure is a model of Amsterdam Airport Schiphol, which consists of 1219 resources. For each aircraft (agent), the visiting sequence consisted of (i) one of six available runways, (ii) one of 209 gates, (iii) one of two de-icing stations, located at the center of the airport, and one of five remaining runways (i.e., different from the arrival runway).

We used a total of 900 agents to perform the experiments. The first agent to make a plan has no reservations to take into account, whereas agent 900 has to respect the reservations of the previous 899 agents (we chose a random order in which to let the agents plan). For each agent, we ran both the Multi-Stage Concatenation (MSC — algorithm 1) and the Multi-Stage Routing (MSR — algorithm 2) algorithm, and we reserved the plan made by MSR. All of the following experiments have been repeated 100 times with different (randomly chosen) visiting sequences.

## 4.2 Results

Figure 2 shows the percentage of plans produced by MSC that are optimal (line with circles), sub-optimal (triangles), and the percentage of null plans (i.e., the percentage of runs that MSC finds no plan — line with squares). Note that to obtain the percentage of runs in which MSR outperforms MSC, we need to add the percentage of sub-optimal plans to the percentage of null plans.

The first conclusion we can draw from figure 2 is that the percentage of null plans not only increases with the number of reservations in the system, but also with the length of the visiting sequence. Although it is true that 200 agents produce more reservations when they have a plan along 8 locations rather than 4, we also see that 80% or more null plans is never reached when  $|\sigma| \leq 6$ , even for 900 agents, whereas it is already reached for around 200 agents when  $|\sigma| = 8$ . Also, for all sizes of visiting sequence we see that the percentage of null plans increases quickly at first, but then it starts to level out. The exception is the Schiphol infrastructure, where the levelling of the percentage of null plans was not (yet?) observed for 900 agents. The behaviour of the null-plan lines in figure 2 leads us to believe that there is a small probability of failure at each intermediate resource, and that this probability increases with the number of reservations in the system, up to a point where no more reservations can be made for a certain time period.

For all sizes of visiting sequence, and for both Schiphol and the random graph, we see that the percentage of sub-optimal plans stays relatively constant. For  $|\sigma| \geq 6$ , the percentage is higher when the percentage of null plans is still small. The actual differences in plan cost are small, however, as shown in table 1. The differences are small indeed for the Schiphol network, and we suspect there are two reasons for this. First of all, the distances between stages on the Schiphol network are larger, so the loss of quality is divided by a greater total plan cost. Second, and perhaps more importantly, the final stage in any plan (a departure runway) is shared by many agents, and will therefore become a bottleneck resource. Hence, any time lost after the penultimate stage (the de-icing station) can be made up because the agent has to wait for entry into the runway. A final conclusion is that the concatenation approach suffers more from incompleteness than from sub-optimality.

**Table 1: Plan cost of sub-optimal MSC plans.**

network type	$ \sigma $	plan cost compared to optimum
random graph	4	102.73%
random graph	6	102.21%
random graph	8	102.09%
Schiphol	4	100.15%

Figure 3 shows the average execution times of the MSR algorithm along with 95% confidence intervals, for visiting sequence sizes 4, 6 and 8. The confidence intervals grow both with the number of agents, but also with the size of the visiting sequence. This means that for larger  $|\sigma|$  there is more variation in execution times. The average CPU-times for the MSC algorithm haven’t been plotted, because the required CPU-time is only a fraction of the time needed by the MSR algorithm (i.e., smaller than 10 ms for  $|\sigma| = 8$ ). This is not surprising, as only  $|\sigma| - 1$  calls to the single-stage

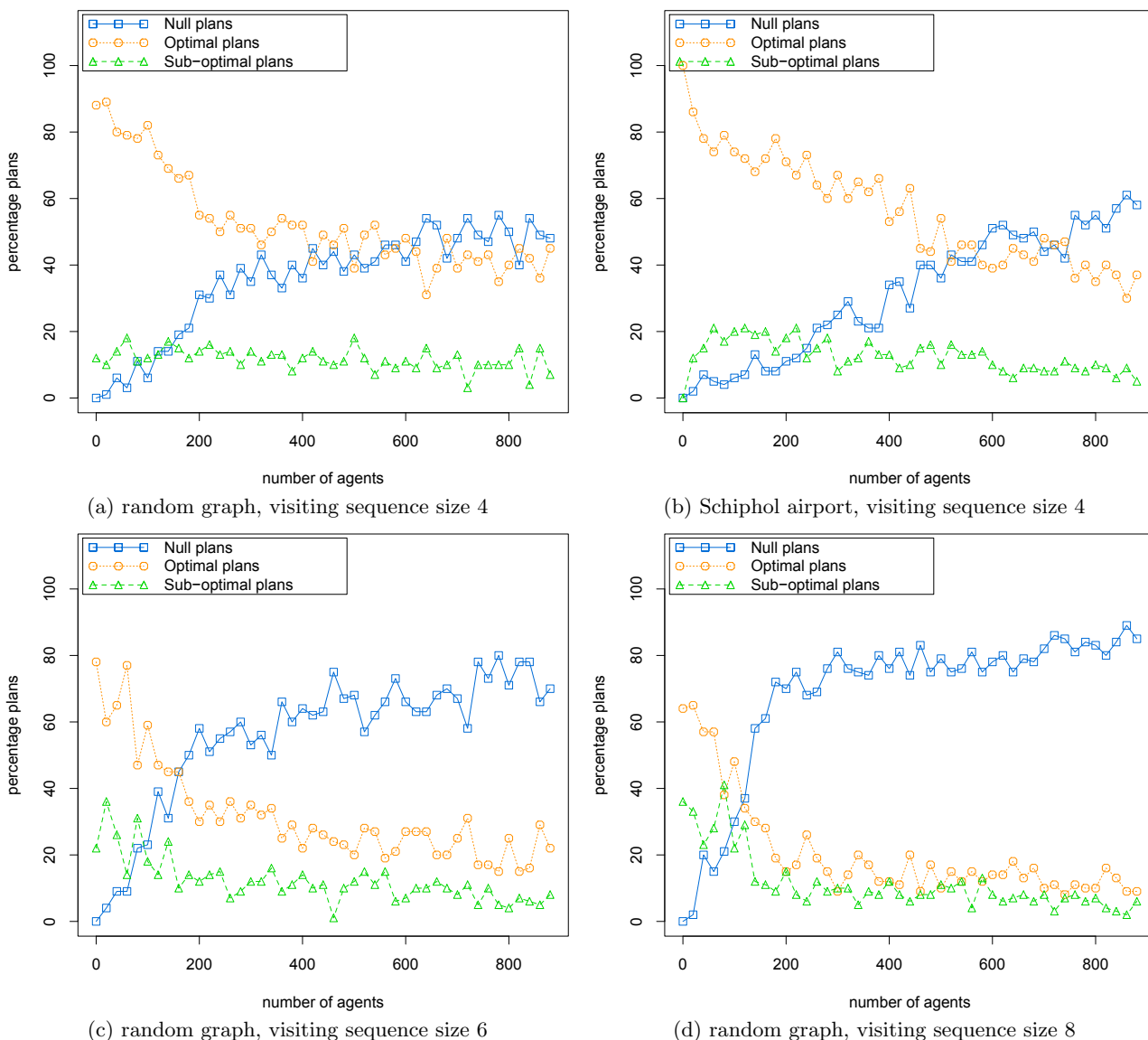


Figure 2: Comparison of plan costs between MSC and MSR.

routing algorithm are required. In figure 3 we can see that the relation between the CPU time and the number of agents is more or less linear for the random graph. Hence, the worst-case complexity of algorithm 2, which would require  $O(|F|^2)$  calls algorithm 3 (for a total complexity that is at least cubic), is not observed.

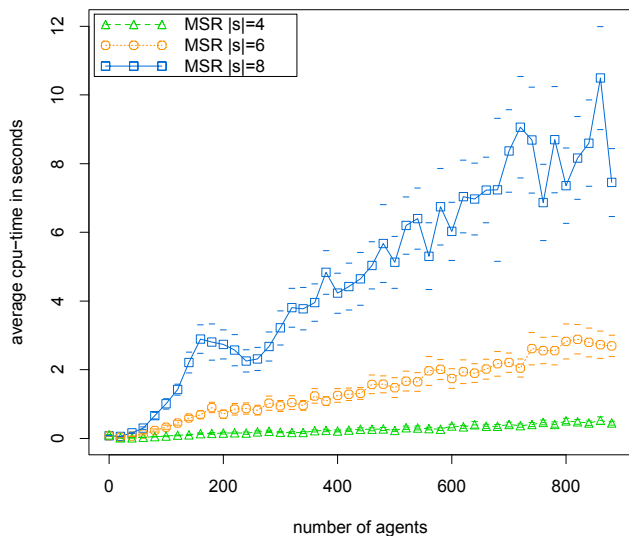
## 5. CONCLUSIONS AND FUTURE WORK

In this paper we presented an optimal algorithm for the multi-stage context-aware routing problem. The straightforward concatenation approach requires less CPU time, but it often fails to find a plan, and more often as the size of the visiting sequence increases. However, when the concatenation approach does yield a plan, its cost is usually close to that of an optimal solution.

For future work, we should try to find algorithms that bridge the computational gap between the concatenation ap-

proach and our optimal algorithm. Such algorithms can be adaptations of algorithm 2 that use different cost functions and different (possibly non-admissible) heuristic functions (cf. [3]). Given the high quality of the plans that *are* returned by the concatenation approach, it may be worthwhile to look for an algorithm that is complete but not optimal.

Taking a broader perspective, we can also investigate the role of context-aware routing in multi-agent systems. So far, existing research has always assumed that placing a reservation on a resource is allowed as long as there is capacity available. However, we can also imagine that the infrastructure resources are governed by agents, and that these agents have preferences about which agent uses a resource, and when. A second assumption is that once a reservation has been made, other agents unquestioningly plan around this reservation. A more interesting scenario might be that agents should negotiate over resource reservations.



**Figure 3: Average CPU-time for MSR algorithm for increasing number of agents on random graph.**

## 6. REFERENCES

- [1] J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.
- [2] G. Bruno, G. Ghiani, and G. Improtà. Dynamic positioning of idle automated guided vehicles. *Journal of Intelligent Manufacturing*, 11(2):209–215, April 2000.
- [3] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A\*. *Journal of the ACM*, 32:505–536, 1985.
- [4] P. E. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC4*, 4(2):100–107, 1968.
- [5] W. Hatzack and B. Nebel. The operational traffic problem: Computational complexity and solutions. In A. Cesta, editor, *Proceedings of the 6<sup>th</sup> European Conference on Planning (ECP’01)*, pages 49–60, 2001.
- [6] C. W. Kim and J. Tanchoco. Conflict-free shortest-time bidirectional AGV routing. *International Journal of Production Research*, 29(1):2377–2391, 1991.
- [7] Y. Kim, T. Suzuki, and T. Narikiyo. Fms scheduling based on timed petri net model and reactive graph search. *Applied mathematical modelling*, 31(6):955–970, 2007.
- [8] J. Lin and P.-K. Dgen. An algorithm for routeing control of a tandem automated guided vehicle system. *International Journal of Production Research*, 32(12):2735–2750, December 1994.
- [9] S. Maza and P. Castagna. A performance-based structural policy for conflict-free routing of bi-directional automated guided vehicles. *Computers in Industry*, 56(7):719–733, 2005.
- [10] J. H. Reif. Complexity of the mover’s problem and generalizations. In *20<sup>th</sup> Annual IEEE Symposium on Foundations of Computer Science*, pages 421–427, San Juan, Puerto Rico, October 1979.
- [11] Y. Shoham and M. Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73(1–2):231–252, 1995.
- [12] F. Taghaboni-Dutta and J. Tanchoco. Comparison of dynamic routing techniques for automated guided vehicle system. *International Journal of Production Research*, 33(10):2653–2669, 1995.
- [13] A. ter Mors, X. Mao, J. Zutt, C. Witteveen, and N. Roos. Robust reservation-based multi-agent routing. In *Proceedings of the 18th European Conference on Artificial Intelligence*. IOS Press, July 2008.
- [14] A. W. ter Mors, J. Zutt, and C. Witteveen. Context-aware logistic routing and scheduling. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 328–335, 2007.
- [15] I. F. Vis. Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170(3):677–709, May 2006.

## APPENDIX

### Single-stage routing algorithm

Algorithm 3 below is similar to the A\* search algorithm [4]. There is an open list  $Q$  (typically implemented as a priority queue) of partial plans, and in each iteration the minimum-cost plan is removed from  $Q$  and expanded to all neighbours. A partial plan ends in some resource  $r_i$  that has been reached at a time  $t$ , where  $t \in \tau_{\text{entry}}(f)$  for some free time window  $f$ . The ‘neighbours’ of a partial plan are determined by the free time window reachability relation  $E_F$ , together with the specific entry time  $t$ . We write  $f' \in \rho(r_i, t)$  if  $(f, f') \in E_F$  and  $\exists t' \in \tau_{\text{entry}}(f')[t + D(r_i) \leq t']$ .

In algorithm 3 we only expand a free time window (line 9) that hasn’t been expanded before at an earlier time, which we verify in line 10. It has been proved in [14] that each free time window can be expanded only once.

---

#### Algorithm 3 planRoute

---

**Require:** start resource  $r_s$ , destination resource  $r_d$ , start time  $t_s$ ; free time window graph  $FTWG = (F, \rho)$

**Ensure:** shortest-time, reservation-respecting route plan from  $(r_s, t_s)$  to  $r_d$ .

- 1: **if**  $\exists v [f_{s,v} \in F_s \mid t_s \in \tau_{\text{entry}}(f_{s,v})]$  **then**
  - 2:      $Q \leftarrow \{(r_s, t_s)\}$
  - 3: **while**  $Q \neq \emptyset$  **do**
  - 4:      $\langle r_i, t_i \rangle \leftarrow \text{argmin}_{(r,t) \in Q} g(r, t)$
  - 5:      $Q \leftarrow Q \setminus \{(r_i, t_i)\}$
  - 6:     **if**  $r_i = r_d$  **then**
  - 7:         **return**  $\langle r_i, t_i \rangle$
  - 8:      $t_{\text{exit}} \leftarrow g(r_i, t_i)$
  - 9:     **for all**  $f_{j,v} \in \rho(r_i, t_{\text{exit}})$  **do**
  - 10:         **if**  $t_{\text{exit}} < \text{entryTime}(f_{j,v})$  **then**
  - 11:              $t_{\text{entry}} = \max(t_{\text{exit}}, \sigma_{j,v})$
  - 12:              $\text{backpointer}(r_j, t_{\text{entry}}) \leftarrow \langle r_i, t_i \rangle$
  - 13:              $Q \leftarrow Q \cup \{(r_j, t_{\text{entry}})\}$
  - 14:              $\text{entryTime}(f_{j,v}) \leftarrow t_{\text{entry}}$
  - 15: **return nil**
-