Plan Repair in Conflict-Free Routing

Adriaan ter Mors[1] and Cees Witteveen[2]

[1] Almende BV, Rotterdam, The Netherlands, adriaan@almende.org
[2] Delft University of Technology, The Netherlands, c.witteveen@tudelft.nl

Abstract. In conflict-free routing a set of agents have to traverse a common infrastructure without interfering with each other. Maza and Castagna [1] showed how route plans can be repaired by maintaining the priority of agents on infrastructure resources. They also developed an algorithm that allows agents to change priorities to avoid long waits. We extend the work of Maza and Castagna by (*i*) specifying an algorithm that allows more priority changes, and by (*ii*) defining a graph structure that can predict exactly which priority changes will lead to a deadlock, and which will not.
Our experiments show that our algorithm performs around twice as many priority changes as Maza and Castagna's algorithm. Also, the average delay produced by our algorithm is lower in our experiments, although there is no clear relation between the number of priority changes and the delay: priority changes seem the most effective when the system is not fully congested, and when there are few incidents that cause delay.

1 Introduction

Let $\mathcal{A}$ be a set of vehicles or *agents* each with a start location and a goal location. The problem of conflict-free routing is to find the shortest-time route to the goal location for each of the agents, in such a way that agents do not interfere with each other (e.g., they do no collide). This problem has applications in the routing of Automated Guided Vehicles (AGVs) in warehouses or at container terminals (such as Rotterdam and Singapore), but also in the routing of aircraft on taxiways.

In the literature many *planning approaches* to this problem exist, that find an optimal (shortest-time) plan for a *single* agent, given the plans of previous agents, which may not be invalidated. Early algorithms include those from Fujii et al. [2], and from Kim and Tanchoco [3], who proved that their algorithm returned the optimal solution in $O(|\mathcal{A}|^4|R|^2)$ time, where $\mathcal{A}$ is the set of agents, and R is the set of infrastructure resources. More recent work includes that of Hatzack and Nebel [4], who presented a sub-optimal $O(|\mathcal{A}||R|)$ algorithm, and an algorithm from Ter Mors et al. [5] that finds the optimal solution in $O(|\mathcal{A}||R|\log(|\mathcal{A}||R|) + |\mathcal{A}||R|^2)$ time.

In of all of the aforementioned application domains, deadlock situations can occur if only one agent is delayed. One approach is to detect and resolve deadlocks (cf. [6]) as they occur, but that leaves the question what to do with the

plans that have been made: it is unclear whether these are still of good quality, or whether they will soon lead to another deadlock. Many other approaches are based on deadlock avoidance, with little or no planning involved. In these approaches, agents usually take the shortest *path*, and before an agent enters the next road or intersection, a check is performed to see if it safe to do so. Wu and Zhou [7], for example, build a Petri net model of the transportation system, and only allow an agent to enter the next segment if it leaves the Petri net in a live state. Reveliotis [8] presents a similar approach in which he extends Dijkstra's Banker's algorithm. The disadvantage of these deadlock avoidance approaches is that they do not try to find optimal agent routes.

The approach we discuss in this paper was first explored by Maza and Castagna [1]. They assume a planning algorithm such as [3], and they note that the set of agent plans specifies for each infrastructure resource the order in which the agents should enter the resource. They prove that if during execution we simply maintain this *priority*, then no deadlocks can occur. A disadvantage is that it requires agents to wait for delayed agents. The authors came up with an algorithm that sometimes allows non-delayed agents to increase their priority. However, the constraints under which this is possible are quite restrictive.

We extend the approach of Maza and Castagna in two ways: first, we present an algorithm that allows more priority changes; second, we define a graph structure that can predict exactly which priority changes will lead to a deadlock, and which will not.

2 Framework

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*. For each agent $A_i \in \mathcal{A}$ there is a pair (s_i, d_i) of locations were s_i is the agent's start location and d_i its destination location. 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 $\mathrm{C} : R \to \mathbb{N}^+$ associates with every resource $r_i \in R$ a capacity $\mathrm{C}(r_i)$ that specifies the maximum number of agents that can simultaneously occupy a resource. For an intersection resource r_i we always have $\mathrm{C}(r_i) = 1$. We also define a function $\mathrm{D} : R \to \mathbb{N}^+$ that gives the minimum travel time of a resource. 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).

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

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

The first constraint in the above definition states that the exit time of the jth 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 conflict-free 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 (Resource Load). *Given a set Π of agent plans, the* resource load *λ 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: $\lambda(r_i, t) = |\{\langle r_i, \tau\rangle \in \pi \mid \pi \in \Pi \wedge t \in \tau\}|$.*

In this paper we consider two additional constraints. The first is that agents are not allowed to overtake each other on a (lane) resource. The second constraint is that a resource can be used in only a single direction at one time. This implies that once an agent has entered a (lane) resource from one end (i.e., an intersection), no other agent can enter the resource from the other end until the first agent has exited. In other words, if two agents occupy the same resource at the same time, they must have entered from the same intersection. Given plan step σ, we write $ep(\sigma)$ to denote the entry point of σ, which equals the previous resource in the plan.

3 Plan execution

We employ a simple model for agent driving: each agent has a maximum speed, and it can drive at any speed up to that maximum speed. We assume no acceleration or deceleration are necessary. Given its plan of pairs of resources and time intervals, an agent will try to traverse each resource in the specified time interval. If an agent arrives at a resource early, it will simply wait before entering the resource; if an agent is late, it will simply continue to drive along the planned sequence of resources, and it will try to make up some time. Making up time might be possible if the planned speed of traversing a resource is less than the agent's maximum speed.

With the above model of plan execution, the delay of one agent can cause a deadlock situation, as the following example illustrates.

Example 1. Consider a small airport with aircraft agents A_1 and A_2, in figure 1. A_1 starts at hangar h_2 and wants to go to h_4, while A_2 wants to go from hangar h_3 to hangar h_1. The agents make the following plans, to the effect that A_1 is allowed to pass along the taxiway straight $t_2 - t_4 - t_5$ first:

$$\pi_{A_1} = \langle h_2, 0\rangle, \langle t_3, 2\rangle, \langle t_2, 4\rangle, \langle t_4, 5\rangle, \langle t_5, 10\rangle, \langle t_7, 11\rangle, \langle h_4, 13\rangle$$
$$\pi_{A_2} = \langle h_3, 0\rangle, \langle t_6, 2\rangle, \langle t_5, 11\rangle, \langle t_4, 12\rangle, \langle t_2, 17\rangle, \langle t_1, 18\rangle, \langle h_1, 20\rangle$$

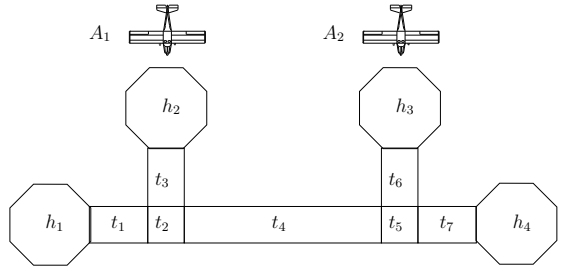

Fig. 1. Two aircraft agents A_1 and A_2 with hangars h_4 and h_1 as respective destinations.

Hence, A_2 plans to stay on taxiway t_6 in the interval $[2, 11)$, just long enough to let A_1 pass. Suppose that during plan execution, A_1 stalls his engine, and he departs with a delay of 5. Then, he will be on taxiway t_4 in the interval $[10, 15)$. Clearly, this interferes with the plan of A_2, who will be on t_4 from 12 until 17. Agent A_1 and A_2 will therefore meet each other on taxiway t_4, and unless either of them can drive backwards, they will end up in a deadlock situation.

What happened in example 1 was that the *priority* that the agents had agreed upon during planning was violated during plan execution. After the planning phase, we know for each resource in which order it will be visited by the agents. If we maintain this order during plan execution, then no deadlocks can occur [9]. This leads us to formulate the following conditions for entry into a resource:

Definition 3 (Resource Entry Permission). *An agent A_i is allowed to start its next plan step $\sigma_i = \langle r_j, \tau = [t_s, t_e) \rangle$ by entering resource r_k at time t if:*

1. *σ_i is plan step number n on r_k, and $n - 1$ plan steps have been started on this resource so far,*
2. *the number of agents on r_k at time t is at least one less than $\mathrm{C}(r_k)$,*
3. *if an agent A_j occupies r_k at time t, with plan step σ_j , then $ep(\sigma_i) = ep(\sigma_j)$,*
4. *$t \geq t_s$.*

Changing the priority (i.e., the plan step number on a resource) of the agents' plan steps during plan execution can often be beneficial, for instance if A_1 in example 1 suffers a very long delay, then A_2 should go first. The question is whether we can change the priority of the agents without introducing deadlocks. One simple mechanism is from Maza and Castagna [1]: *Algorithm Maza-2:* Grant one agent the highest priority from the current resource — where the agent is waiting for some delayed agent — until the resource in its plan where the agent already has the highest priority (i.e., this agent is the next to enter), if and only if the resources, on which the agent will inherit the highest priority, are currently empty.

4 Planstep-Priority Graph

In this section we will construct the *Planstep-Priority Graph (PPG)* from the (plan) steps of the plans of the agents. We will show that a deadlock will occur if and only if this graph contains a cycle. In the following we will write $p(r_k, \sigma_{i,x})$ to denote the priority of plan step $\sigma_{i,x}$ on resource r_k.

Definition 4 (Enabling Plan Step). *Plan step $\sigma_{i,x+1} \in \pi_i$ is the* enabling plan step *of $\sigma_{j,y} \in \pi_j$ if $\sigma_{i,x}$ (i.e., the previous step in π_i) and $\sigma_{j,y}$ are on the same resource r_k, and:*

1. $p(r_k, \sigma_{j,y}) = p(r_k, \sigma_{i,x}) + 1 \ \wedge ep(\sigma_{i,x}) \neq ep(\sigma_{j,y})$*; or*
2. $p(r_k, \sigma_{j,y}) = p(r_k, \sigma_{i,x}) + \mathrm{C}(r_k) \wedge$
   $\forall h, w : p(r_k, \sigma_{j,y}) \leq p(r_k, \sigma_{h,w}) \leq p(r_k, \sigma_{i,x}) : ep(\sigma_{i,x}) = ep(\sigma_{h,w})$.

When the enabling plan step starts, at least two entry conditions from definition 3 become satisfied: first of all, there are now fewer agents occupying the resource than the capacity, and second, all agents traversing the resource in the opposite direction have exited.

Definition 5 (Planstep-Priority Graph). *Given a set of agent plans $\Pi = \{\pi_1, \ldots, \pi_n\}$, $\pi_i = (\sigma_{i,1}, \ldots, \sigma_{i,m})$, the* Planstep-Priority Graph *(PPG) is a directed graph $G = (V, E)$, where the set of vertices is given by:*

$$V = \bigcup_{i=1}^{|\mathcal{A}|} \bigcup_{j=1}^{|\pi_i|} \sigma_{i,j} \tag{1}$$

That is, there is a vertex for every plan step in every plan. The set of edges is made up of three parts $E = E_1 \cup E_2 \cup E_3$:

$$E_1 = \{(\sigma_{i,j}, \sigma_{i,j+1})\} \tag{2}$$

$$E_2 = \{(\sigma_{i,x}, \sigma_{j,y}), (\sigma_{i,x+1}, \sigma_{j,y+1}) \,|\, p(r_k, \sigma_{i,x}) = p(r_k, \sigma_{j,y}) + 1\} \tag{3}$$

$$E_3 = \{(\sigma_{i,x}, \sigma_{j,y}) |\, \sigma_{i,x} \textit{ is enabling plan step of } \sigma_{j,y}\} \tag{4}$$

The edges in the Planstep-Priority Graph reflect the order in which plan steps should be executed; if $(\sigma_{i,x}, \sigma_{j,y}) \in E$, then plan step $\sigma_{i,x}$ must *start* before $\sigma_{j,y}$ can start. Edges in E_1 express that an agent should perform the steps in its plan sequentially. The set E_2 specifies that if the priority of $\sigma_{i,x}$ (on some resource r_k) is one higher (i.e., the priority number is one *lower*) than the priority of $\sigma_{j,y}$, then $\sigma_{i,x}$ must not only start before $\sigma_{j,y}$, it must also *finish* before $\sigma_{j,y}$ (since overtaking is not allowed). This implies that the next plan step $\sigma_{i,x+1}$ of π_i must start before the next step $\sigma_{j,y+1}$ in π_j. Finally, the set E_3 states that a plan step cannot start until its enabling plan step has started.

Proposition 1. *A deadlock will occur if and only if there is a cycle in the Planstep-Priority Graph.*

Proof (sketch). Case I: deadlock → cycle in PPG. A deadlock occurs if no agent can make any progress any more. In our model, this can occur if an agent is waiting to enter a resource, or if an agent is 'in the middle' of a resource, but stuck behind another agent. In both of these cases, we can find a path of edges in E to connect a waiting agent to the culprit. Because a deadlock involves a cyclic wait, the paths in E between waiting agents form a cycle.

Waiting to enter a resource: Definition 3 gives three reasons why an agent should wait for entry; two of them indicate that an agent's enabling plan step has not started yet (in which case we can find an edge in E_3). The third reason is that it's not yet the agent's turn to enter a resource; in this case, we can construct a path between the waiting agents using edges from E_1 and E_2.

Stuck behind another agent: Using edges in E_2, we can find a path to the first agent to exit the resource, and this agent is waiting to enter its next resource, so we return to the previous case.

Case II: cycle in PPG → deadlock. The edges in PPG specify a precedence order between plan steps; if this order is cyclic, then no plan step in a cycle can ever start, and hence a deadlock will occur the moment an agent tries to start a plan step that is part of a cycle.

4.1 Algorithm: increasing agent priorities

Arriving at a resource, an agent A_i may find that there are one or more agents with a higher priority that have not entered the resource yet. To continue driving, agent A_i should increase its priority to the maximum of all these delayed agents. If we only change priorities on the current resource, however, then it is easy to create a deadlock situation. Suppose that one of the delayed agents, agent A_j, follows the same route as (what remains of the plan of) A_i, but in the opposite direction. If agent A_i is allowed to increase its priority, then sooner or later agent A_i and agent A_j will meet, when neither can make progress anymore.

For each of the delayed agents, A_i must check whether they also have a higher priority on subsequent resources in A_i's plan, and, if yes, then A_i can only increase its priority if these agents are not already occupying one of these resources.

From lines 7 and 9 we can see that the set A of delayed agents can change as A_i scans the resources that remain in its plan. A new agent A_j can be added to A if for some future resource r_k the priority of A_j is higher than that of A_i, but lower than that of some agent already in A. Of course, an agent can also disappear from the set A as soon as it does not share any resources anymore with agent A_i.

In most cases, the procedure described above results in a deadlock-free change of priorities. In a few cases, however (most notably when a delayed agent's path crosses that of A_i more than once), it is still possible that a deadlock situation is created. Therefore, we simply check after increasing an agent's priority whether the Planstep-Priority Graph is still acyclic.

Algorithm 1 Increase Agent Priority

Require: agent A_i, plan π, Planstep-Priority Graph PPG
Ensure: give agent A_i the highest priority on the first resource of π, in case this does not create a deadlock

```
r_curr ← π(0).resource
A ← {A_j ∈ agents(r_curr) | p(r_curr, A_j) < p(r_curr, A_i)}
M ← ∅
k ← 0
deadlock ← false
while A ≠ ∅ ∧ ¬deadlock do
    r_curr ← π(k).resource
    A_min ← min_{A_j ∈ A} p(r_curr, A_j)
    A ← {A_j ∈ agents(r_curr) | p(r_curr, A_min) ≤ p(r_curr, A_j) < p(r_curr, A_i)}
    for all A_j ∈ A do
        if locatedAt(A_j, r_curr) then
            deadlock ← true
            continue
    M ← M ∪ ⟨r_curr, A⟩
    k ← k + 1
if ¬deadlock then
    for all ⟨r_k, A⟩ ∈ M do
        increasePriority(A_i, A, r_k, PPG)
    if cycle in PPG then
        rollbackPriorityChanges(M, PPG)
```

5 Experiments

In this section, we will evaluate the performance of our IAP algorithm in comparison with algorithms 1 and 2 from Maza and Castagna (we will refer to these algorithms as Maza-1 — which is to perform no priority changes — and Maza-2 respectively). The first question we aim to answer is: *which algorithm allows the most priority changes?* The second question is: *which algorithm results in plan execution with the smallest delay?* Intuitively, we would expect that the algorithm that performs the most changes will result in the smallest delay, because with every change an agent with little or no delay is given priority over an agent with more delay.

To introduce delay into the system, we introduce random incidents that temporarily immobilize an agent. The *incident rate* is a value between 0 and 1 that indicates the probability that an agent will suffer an incident during the execution of one plan step. Hence, if the incident rate equals 0.1, and an agent's plan contains 50 plan steps, then the agent can expect to have five incidents on the way. The *incident duration* is always the same for one experiment run.

5.1 Setup

We experimented with a set of 100 random infrastructures, each consisting of 120 intersections and 200 lanes. To evaluate which algorithm results in the smallest

delay, we measure the (relative) *mechanism delay*: the time an agent has to wait for one or more conditions of definition 3 to become satisfied (or if an agent is waiting behind another agent that is accumulating mechanism delay), as a percentage of its plan length.

In our experiments we varied three parameters: the number of agents, from 100 to 500, with an increment of 10; the incident rate, with values 0.02, 0.06, and 0.1; and the incident duration, with values of 30 and 60 seconds. In total, this amounts to around 800 experiments. To compare the different algorithms, we ran each of them on the same problem instances. One problem instance consists of: (*i*) one of the 100 infrastructures, chosen by: experiment-number modulo 100; (*ii*) for each agent a randomly chosen start and destination location; (*iii*) a set of agent plans, obtained by letting each agent plan its optimal route, given the reservations of the agents that have planned before it; (*iv*) for each step of each agent plan, we use the incident rate parameter to determine whether the agent will suffer an incident during the execution of this plan step.

5.2 Results

In figure 2, we compare the number of priority changes performed by algorithms IAP and Maza-2, for three different parameter settings: an incident rate of 0.02, an incident rate of 0.06, and an incident rate of 0.1; the incident duration was set to 60 seconds (other settings have been omitted from the figure, but do not change the general picture). The conclusions that can be drawn from figure 2 are straightforward: (*i*) IAP performs more changes than Maza-2, sometimes up to twice as many; (*ii*) the number of changes performed by both of the algorithms increases linearly with the number of agents.

Figure 3 shows the relative mechanism delay (i.e., the mechanism delay divided by the plan length), averaged over the six different incident settings (incident rates of 0.02, 0.06, and 0.1, and incident durations of 30 and 60 seconds). We can see from figure 3 that IAP results in lower mechanism delay than algorithms Maza-1 and Maza-2. However, the differences are perhaps not as large as we may have anticipated after seeing figure 2: IAP often performs twice as many priority changes as Maza-2, but the difference in performance ranges from 0% to 10% of relative mechanism delay. Moreover, algorithm Maza-1, which does not perform any priority changes at all, is actually quite competitive for larger numbers of agents in the system.

Table 1 shows the relative mechanism delay for each of the six incident settings. For each combination of incident setting and number of agents, the algorithm that produced the lowest delay is highlighted in bold. A quick glance at the table reveals that for settings with a low incident load, IAP (and to a lesser extent, Maza-2) performs very well. For the highest incident load (rate = 0.1 and duration = 60s), however, Maza-2 and Maza-1 perform better, with the latter best of all for more than 300 agents. From this table we can conclude that for higher incident loads and more congestion, there is less benefit from making priority changes.

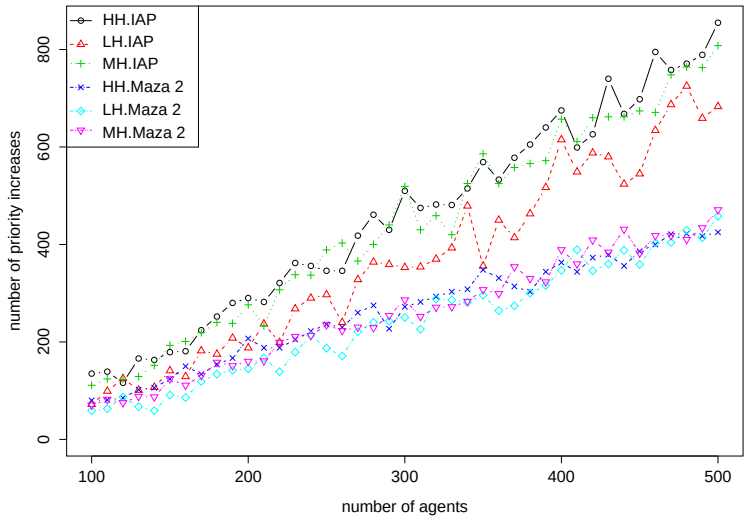

Fig. 2. Number of priority changes made by IAP and algorithm Maza-2 for incident settings: HH = (0.1, 60s) ; MH = (0.06, 60s) ; LH = (0.02, 60s).

Table 1. Relative mechanism delay for six different incident settings.

$\|\mathcal{A}\|$	0.02, 30s			0.02, 60s			0.06, 30s			0.06, 60s			0.1, 30s			0.1, 60s		
	M1	M2	IAP	M1	M2	IAP	M1	M2	IAP	M1	M2	IAP	M1	M2	IAP	M1	M2	IAP
100	14	3	**2**	36	8	**4**	30	**20**	25	53	35	**28**	28	16	**9**	44	**25**	27
150	15	10	**7**	33	21	**10**	27	10	**6**	36	**22**	26	24	13	**10**	40	**31**	32
200	14	11	**5**	38	11	**8**	17	12	**9**	44	35	**30**	22	26	**14**	34	**25**	26
250	16	10	**6**	30	27	**18**	17	11	**6**	36	32	**24**	18	13	**5**	33	28	**28**
300	11	5	**3**	30	18	**5**	19	13	**13**	37	36	**36**	21	**15**	15	**32**	35	45
350	11	6	**4**	30	16	**9**	16	18	**13**	31	31	**30**	17	**14**	14	**27**	31	28
400	9	4	**3**	32	24	**16**	15	11	**6**	29	26	**21**	**16**	27	20	**25**	29	35
450	10	5	**3**	26	16	**11**	16	11	**8**	26	27	**22**	18	13	**11**	**25**	39	43
500	12	5	**3**	28	19	**10**	14	15	**8**	25	**20**	21	17	14	**8**	**22**	24	33

5.3 Discussion

The rationale behind making priority changes in the manner of algorithms Maza-2 and IAP is that a 'timely' agent can continue execution of its plan unhindered, whereas the delayed agent might not even notice the loss of priority. Of course, this is only a rule of thumb, and, as we have seen in figure 3 and table 1, it does not always hold, and sometimes priority changes can have averse effects.

The added value of IAP over Maza-2 is therefore better illustrated by figure 2 than by figure 3. In figure 2 we see that IAP finds more possibilities for changing priorities than Maza-2. Currently, we do not have a clear idea which priority changes are beneficial, and which are detrimental. However, if we develop additional heuristics that tell us when priority changes are good, then we can use IAP as a basis for a more effective plan repair algorithm.

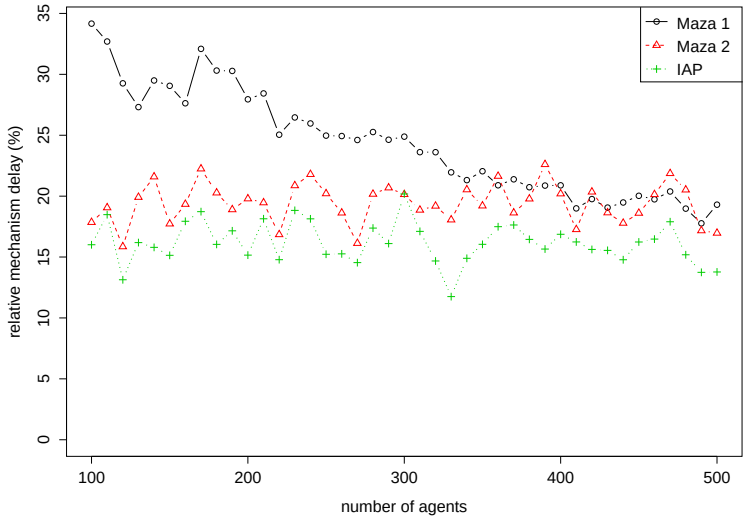

Fig. 3. Relative mechanism delay, averaged over different incident settings.

References

1. Maza, S., Castagna, P.: A performance-based structural policy for conflict-free routing of bi-directional automated guided vehicles. Computers in Industry **56**(7) (2005) 719–733
2. Fujii, S., Sandoh, H., Hozaki, R.: A routing control method of automated guided vehicles by the shortest path with time-windows. In: Production Research: Approaching the 21st Century. (1989) 489–495
3. Kim, C.W., Tanchoco, J.: Conflict-free shortest-time bidirectional AGV routeing. International Journal of Production Research **29**(1) (1991) 2377–2391
4. Hatzack, W., Nebel, B.: The operational traffic problem: Computational complexity and solutions. In Cesta, A., ed.: Proceedings of the 6th European Conference on Planning (ECP'01). (2001) 49–60
5. ter Mors, A.W., Zutt, J., Witteveen, C.: Context-aware logistic routing and scheduling. In: Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling. (2007) 328–335
6. Lehmann, M., Grunow, M., Günther, H.O.: Deadlock handling for real-time control of AGVs at automated container terminals. OR Spectrum **28**(4) (October 2006) 631–657
7. Wu, N., Zhou, M.: Resource-oriented petri nets in deadlock avoidance of agv systems. In: Proceedings of the 2001 IEEE International Conference on Robotics and Automation (ICRA), Seoul, Korea (May 2001) 64–69
8. Reveliotis, S.A.: Conflict resolution in AGV systems. IIE Transactions **32**(7) (July 2000) 647–659
9. Maza, S., Castagna, P.: Conflict-free AGV routing in bi-directional network. In: Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation. Volume 2., Antibes-Juan les Pins, France (October 2001) 761–764