# Suboptimal and conflict-free control of a fleet of AGVs to serve online requests

8th European Congress of Mathematics

---

**Markó Horváth** · Péter Györgyi · Tamás Kis · Márton Drótos
Institute for Computer Science and Control

Online, 20-26 June 2021

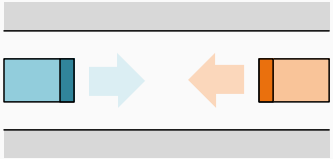
# Introduction

# Introduction

- Autonomously guided vehicles (AGVs) are widely used in modern manufacturing systems and they are the source of several research problems as well
- Avoiding physical collisions is not too difficult today by appropriate sensors and hardware
- Excluding the possibility of different conflict situations among vehicles is a challenging task
- Our goal is to ensure conflict-free operation of vehicles while serving transportation requests arriving over time

# What is deadlock?

"Each vehicle of a group is waiting for another member to take an action"



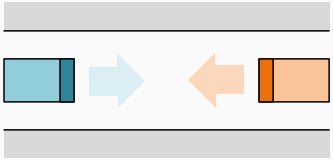
**(a)** Deadlock in a narrow corridor

# What is deadlock?

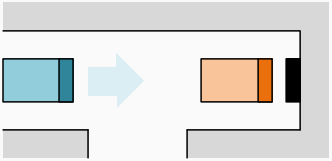
"Each vehicle of a group is waiting for another member to take an action"

+

Idle vehicles...



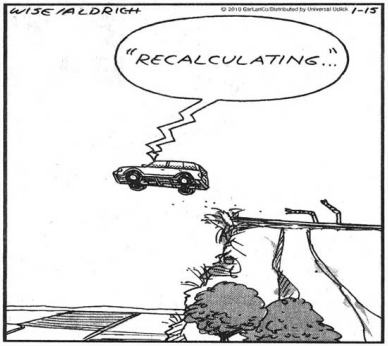
**(a)** Deadlock in a narrow corridor



**(b)** Deadlock caused by an idle vehicle

# What to do with deadlock?

- **Resolve existing situation**
  - recalculating
  - last resort: manual intervention




Real Life Adventures by Gary Wise and Lance Aldrich

# What to do with deadlock?

- **Resolve existing situation**
  - recalculating
  - last resort: manual intervention
- **Resolve future situation** (recognize deadlock in advance)
  - recalculating
  - segment reservation, ...

# What to do with deadlock?

- **Resolve existing situation**
  - recalculating
  - last resort: manual intervention
- **Resolve future situation** (recognize deadlock in advance)
  - recalculating
  - segment reservation, ...
- **Avoid deadlock by restrictions**
  - simple network (e.g., grid, one-way edges, ...)
  - splitting workspace into zones; few number of vehicles
  - ...

# What to do with deadlock?

- **Resolve existing situation**
  - recalculating
  - last resort: manual intervention
- **Resolve future situation** (recognize deadlock in advance)
  - recalculating
  - segment reservation, ...
- **Avoid deadlock by restrictions**
  - simple network (e.g., grid, one-way edges, ...)
  - splitting workspace into zones; few number of vehicles
  - ...
- **Avoid deadlock inherently by planning**

# Pick-up and delivery problem with online requests

- Given a fleet of AGVs
- Given a mixed graph $G = (N, A)$ which describes the layout
  - nodes in $N$ represent the docking stations, the intersections of the lanes, and the parking places
  - undirected arcs in $A$ correspond to lanes with bidirectional traffic, while directed arcs model one-way lanes
- Requests arrive in an online fashion
  - each request prescribes the transportation of a single item (part or pallet) from one station to another, that is, from its pick-up station to its delivery station
  - each request has a due date
- Goal: satisfy the requests minimizing the total tardiness

# An example of conflict-free control

**Method of Malopolski[1]:**

- Each vehicle has a dedicated depot node which cannot block the route of any other vehicle
  - vehicles always start and end their routes at their dedicated depot nodes
- If a request is assigned to a vehicle, a route of three paths is generated:

  $$\text{depot node} \rightarrow \text{pick-up node} \rightarrow \text{delivery node} \rightarrow \text{depot node}$$

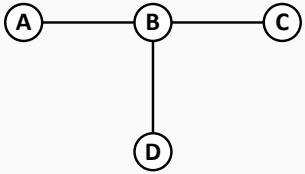
  - the vehicle is pushed back to the traversing orders of the visited nodes
- Precedence constraints: a vehicle can go through a node only if it is the first vehicle in the traversing order of the node

[1] Małopolski, W. (2018). A sustainable and conflict-free operation of AGVs in a square topology. Computers & Industrial Engineering, 126, 472-481.

# A schedule-based approach to avoid deadlock

# Schedule of the vehicles

- Avoid deadlock inherently by maintaining a feasible schedule
  - *resources:* nodes and edges (with capacities)
  - *jobs:* vehicle operations ('no-wait' criteria)
    (WoN: Wait-on-Node; EtN: Edge-to-Node; NtE: Node-through-Edge)

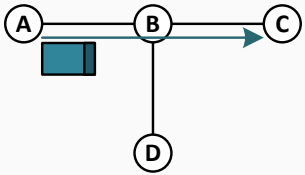



| | |
|---|---|
| A | |
| AB | |
| B | |
| BC | |
| C | |
| BD | |
| D | |

**Figure:** A network and the corresponding resources

# Schedule of the vehicles

- Avoid deadlock inherently by maintaining a feasible schedule
  - *resources:* nodes and edges (with capacities)
  - *jobs:* vehicle operations ('no-wait' criteria)
    (WoN: Wait-on-Node; EtN: Edge-to-Node; NtE: Node-through-Edge)



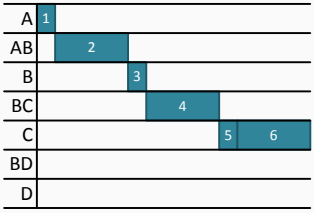
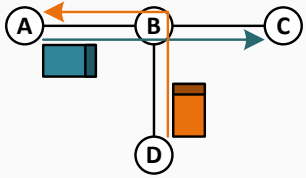




**Figure:** WoN(A) → NtE(AB) → EtN(B) → NtE(BC) → EtN(C) → WoN(C)

# Schedule of the vehicles

- Avoid deadlock inherently by maintaining a feasible schedule
  - *resources:* nodes and edges (with capacities)
  - *jobs:* vehicle operations ('no-wait' criteria)
    (WoN: Wait-on-Node; EtN: Edge-to-Node; NtE: Node-through-Edge)



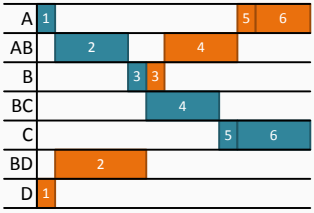




**Figure:** WoN(D) → NtE(BD) → EtN(B) → NtE(AB) → EtN(A) → WoN(A)

# Schedule of the vehicles

- Avoid deadlock inherently by maintaining a feasible schedule
  - *resources:* nodes and edges (with capacities)
  - *jobs:* vehicle operations ('no-wait' criteria)
    (WoN: Wait-on-Node; EtN: Edge-to-Node; NtE: Node-through-Edge)

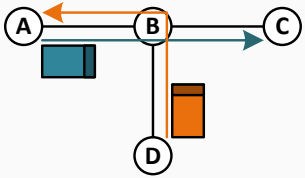
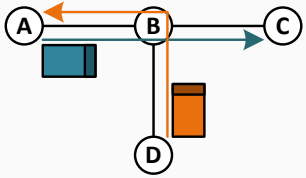


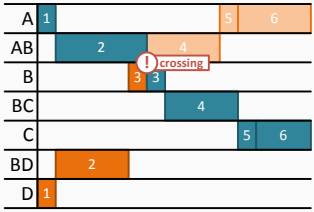


**Figure:** Schedule is infeasible due to 'crossing'

# Schedule of the vehicles

- Avoid deadlock inherently by maintaining a feasible schedule
  - *resources:* nodes and edges (with capacities)
  - *jobs:* vehicle operations ('no-wait' criteria)
    (WoN: Wait-on-Node; EtN: Edge-to-Node; NtE: Node-through-Edge)




- **What matters: sequence**
  - lower bounds on the processing time of operations
  - if there are no crossings (i.e., schedule is feasible) then start and completion times can be calculated from the sequence of operations

# Schedule of the vehicles

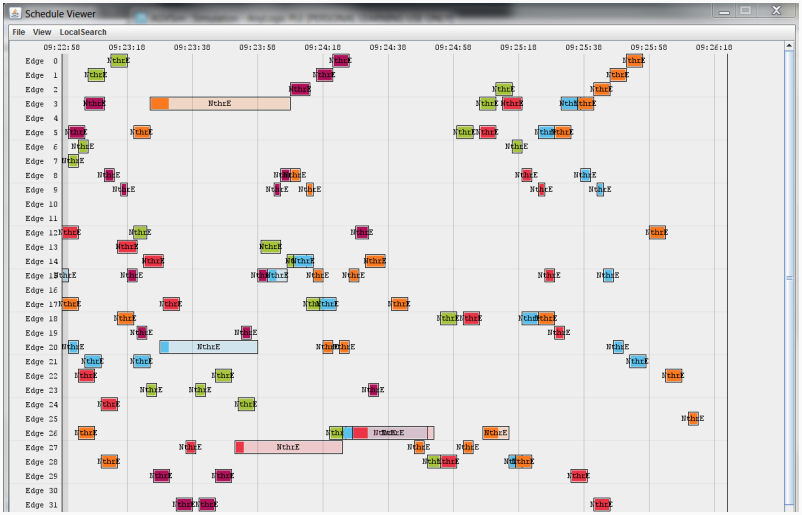



**Figure:** Screenshot of our Schedule Viewer

# Creating and maintaining a feasible schedule

**How to create a feasible schedule?**

- In the beginning each vehicle stands in a (maybe fictive) node

# Creating and maintaining a feasible schedule

**How to create a feasible schedule?**

- In the beginning each vehicle stands in a (maybe fictive) node

**How to maintain a feasible schedule?**

- We insert routes (i.e., a sequence of operations) into the schedule one-by-one
- **How to determine pull-of routes?**

# Creating and maintaining a feasible schedule

**How to create a feasible schedule?**

- In the beginning each vehicle stands in a (maybe fictive) node

**How to maintain a feasible schedule?**

- We insert routes (i.e., a sequence of operations) into the schedule one-by-one
- **How to determine pull-of routes?**

**How to insert a sequence of operations into the schedule?**

- Mixed integer linear program
  - constraints: feasible insertion of operations
  - constraints: determine start and completion times
  - objective: minimize the maximum tardiness
- Improve schedule: Local search, loop elimination

# How to determine pull-off routes for the blocking vehicles?

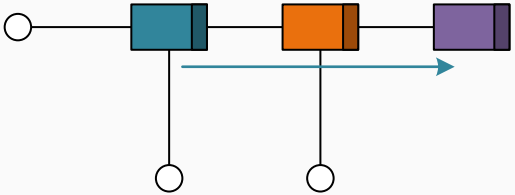


**Figure:** Orange and Purple vehicles block Blue vehicle: do pull-off routes!

# How to determine pull-off routes for the blocking vehicles?

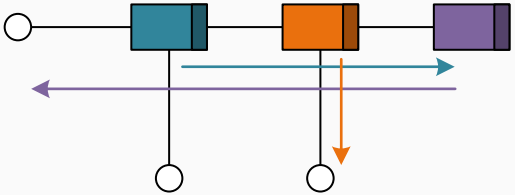


**Figure:** The way is free for Orange vehicle

# How to determine pull-off routes for the blocking vehicles?

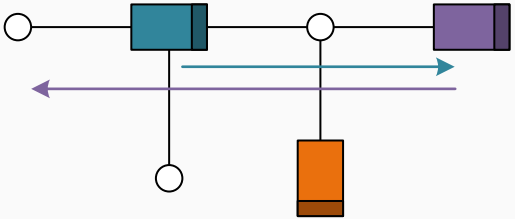


**Figure:** Blue vehicle blocks Purple vehicle: do pull-off route!

# How to determine pull-off routes for the blocking vehicles?

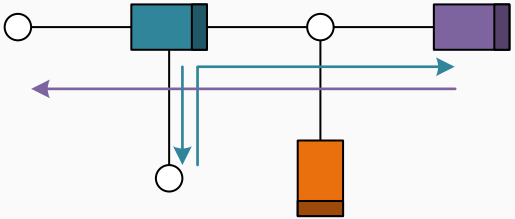


**Figure:** The way is free for Blue vehicle (pull-off route)

# How to determine pull-off routes for the blocking vehicles?

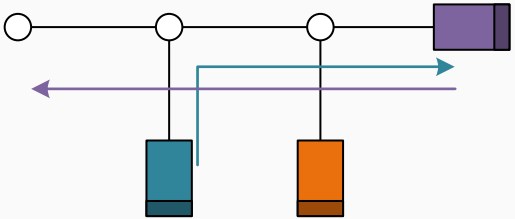


**Figure:** The way is free for Purple vehicle

# How to determine pull-off routes for the blocking vehicles?

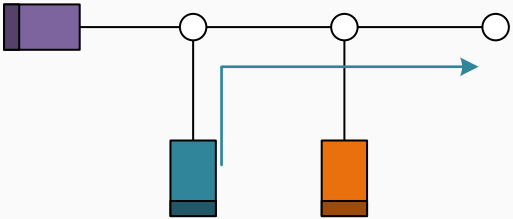


**Figure:** The way is free for Blue vehicle (original route)

# How to determine pull-off routes for the blocking vehicles?

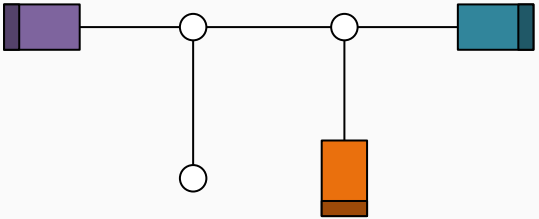


**Figure:** Blue vehicle reached its destination

# How to determine pull-off routes for the blocking vehicles?


**Figure:** 15-puzzle

# Sketch of solution process

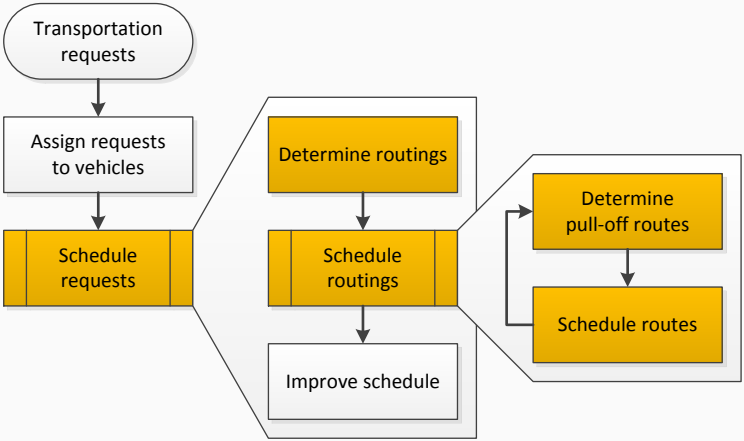



**Figure:** Very sketchy flowchart of our solution process

# Computational experiments

- Compare different methods
  - (Our) Our plain solution approach
  - (Our + LS) Our plain solution approach + Local search
  - (Our + Impr) Our plain solution approach + Local search + Loop elimination
  - (Malopolski) Malopolski's method
  - (Malopolski + Impr) Malopolski's method + Local search + Loop elimination
- 3 different layouts

# Computational experiments

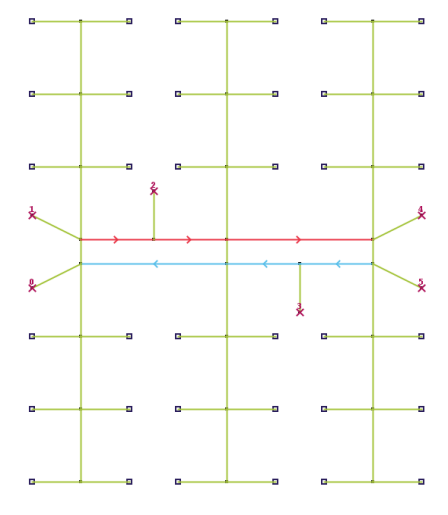
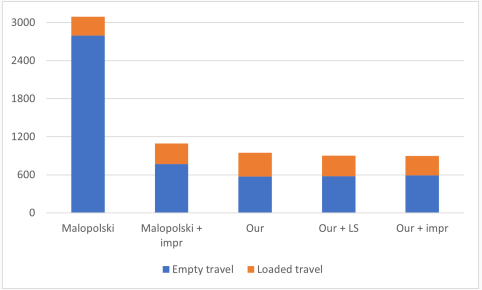
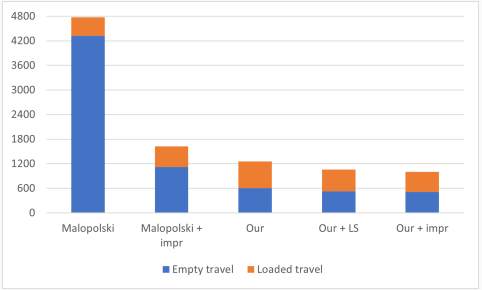



**Figure:** An example layout with 6 vehicles

# Computational experiments



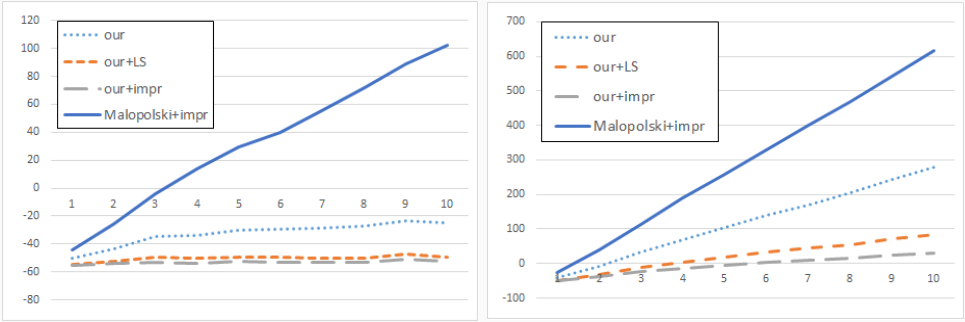

**(a)** Average results with 79 requests




**(b)** Average results with 119 requests

**Figure:** Total empty and loaded travel times

# Computational experiments




**(a)** Average results with 79 requests

**(b)** Average results with 119 requests

**Figure:** Average lateness

# Summary of our solution approach

- The created plans are conflict-free (guaranteed by their structure)
- Our method does not require the identification or classification of possible conflict situations
- We can handle idle blocking vehicles and vehicles with common target locations, without creating any further conflicts
- Our approach supports the optimization of plans, and we describe a method based on local search
- We have only very mild assumptions on the layout
- Our approach requires only low computational time (thus it avoids the main drawback of centralized methods), and it scales up well with the number of AGVs, and the size of the network
- It outperforms the method of Malopolski (2018) with respect to average tardiness of the requests

# Thank you for your attention!

✉ marko.horvath@sztaki.hu

Drótos, M., Györgyi, P., Horváth, M., & Kis, T. (2021). *Suboptimal and conflict-free control of a fleet of AGVs to serve online requests*. Computers & Industrial Engineering, 152, 106999.

This work has been supported by the National Research, Development and Innovation Office (NKFIH) grant no. SNN 129178, and ED_18-2-2018-0006.