Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Planning for a Team of Cooperating Mobile Robots

Tomáš Rybecký

Supervisor: RNDr. Miroslav Kulich Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Robotics
May 2018

Acknowledgements

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 25. May 2018

.....
signature

Abstract

This thesis aims at the topic of route planning for groups of cooperating agents. Several approaches were discussed, and experiments were performed to compare their attributes. For that purpose, all algorithms were implemented in C++ and provided the same set of randomly generated maps and assignments. Following that, the benefits of all presented algorithms are discussed introducing possible practical usage and future development.

Keywords: route planning, cooperative robots

Supervisor: RNDr. Miroslav Kulich Ph.D.
CIIRC,
Jugoslávských partyzánů 1580/3
Praha 6

Abstrakt

Tato bakalářská práce se zaměřuje na téma plánování trajektorií pro skupiny kooperujících agentů. Obsahuje popis několika postupů doplněných o experimenty srovnávající jejich vlastnosti. Za tím účelem byly všechny algoritmy implementovány v jazyce C++ a testovány na řadě náhodně vygenerovaných map a zadání. Následně byl rozebrán přínos všch prezentovaných algoritmů společně s jejich možným praktickým využitím i budoucím vývojem.

Klíčová slova: plánování trajektorií, kooperace robotů

Překlad názvu: Plánování pro tým kooperujících mobilních robotů

Contents

Figures

Chapter 1

Introduction

In the recent years and in the years to follow, automatization of the industry is one of the most important tasks for robotic sciences. Automated warehouses, automatic airport departure systems and other applications as well allow higher traffic combined with the better effectivity of work needed in the thickening field of e-commerce [2] [3]. The other contribution of automatization is releasing human resources from lower qualified jobs and bringing more prospective employers to fields dependent on humans.

One of the main tasks in automated warehouses or airports development is the cooperation of robots used for manipulation of goods or managing planes traffic respectively [4] [1]. Several solutions were presented during the previous decades attempting to find the near-optimal algorithm [5].

This thesis develops an approach presented by A. W. ter Mors in [1], that views the map as a three-dimensional matrix of time windows and uses a variation of a basic A^* algorithm to find a path through the empty time windows. The development aims at increasing the effectiveness of the algorithm by lowering the susceptibility to order of planned assignments. With that, an algorithm for adding one agent to a situation of already running tasks is presented.

In Chapter 2, the Context-aware route planning algorithm is presented and described. Several improvements are addressed with both their advantages and disadvantages. Chapter 3 then introduces the main idea of this thesis, clarifies the theoretical background and describes algorithms presented in this thesis.

Experiments and their aims are presented in Chapter 4 with a discussion of their results among all compared algorithms. Chapter 5, finally, contains a conclusion of the tested approaches and their possible future development.

Chapter 2

Context-aware route planning

Context-aware route planning (CARP) is an approach to solving the problem of a set of agents (vehicles in Automated Guided Vehicle Systems) each attempting to finish its task with the lowest possible time cost. This task is combined with the demand for avoiding collisions and deadlocks by constructing a set of conflict-free plans.

2.1 Model

The problem is modeled on a resource graph $G = \{V, E\}$ representing the working area. It is formed by resources, matching for example intersections in a warehouse, forming the vertices V of the graph. Each resource $r \in V$ holds information of its use during the time and has its defined duration $d(r)$, a minimal time it takes the agent to pass through the resource. Usage during time creates time windows in resources saying whether their capacity is reached or they are free in any time interval. These time windows narrow the set of physical connections to edges E standing only for the available paths. A simplification is made by assuming unit edges length.

An illustration of the graph G displayed over a simple warehouse scheme is in the Figure 2.1. The graph is shown from the point of view of a newcomer at one moment in time. Grey bordered lines are the physical lanes, while in blue, the actual edges E of graph G at the exact time are displayed with vertices V as their intersections. The lighter parts show that the vertices

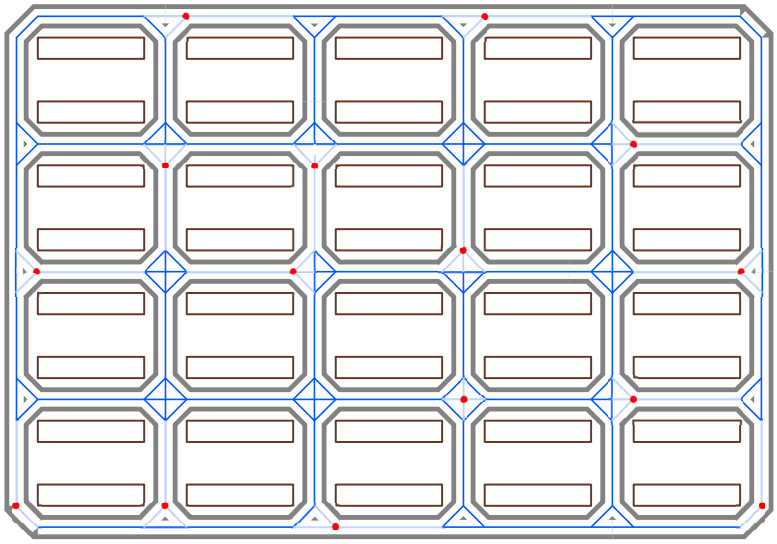


Figure 2.1: Warehouse example illustration with a resource graph.

marked red, which they lead to, have reached their capacity and cannot be accessed by anyone else. The result for the newcomer is that it cannot use these edges, because its time window in the resource marked red would collide with the previous reservation. In other words, at the time the agent would reach the resource, it would still not be empty.

The algorithm is given a set of assignments $A = \{a_1, a_2, ..., a_n\}$, where $a_j = \{r_s, r_g, t\}$. Each assignment contains a start and goal resource, r_s and r_g respectively, and the time t at which the assignment starts. The set A is assumed as finite, but in the following Chapter, an approach allowing repeated joining of new assignments is presented. The desired output is a set of plans with each agent specifying the resources it will visit and at what time it will occupy them.

The CARP algorithm, as presented in [1], gets the resource graph G and the set of assignments A as an input. After the reduction of edges to free time windows, these are used as additional constraints for a modified A^* algorithm finding a path through both the real plan and the empty windows for each of the assignments. A resource is therefore added to the plan only when all the following conditions are met:

1. current and considered resource are in successor relation
2. cost of the route is minimized
3. at the time agent enters the considered resource; it is not used (or its usage is lower than its capacity)

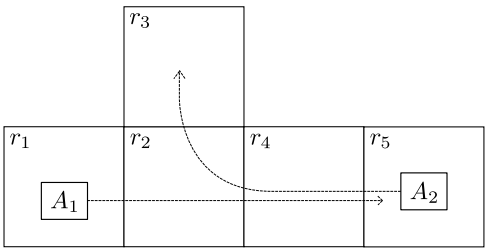


Figure 2.2: Example of a corridor problem as presented in [1].

4. the free time window in the considered resource is long enough for the agent to traverse

The last two conditions ensure the ability of the CARP to find a collision-free plan, as they only allow an agent to enter a resource if its desired time window in the resource does not intersect with time windows of previously planned agents. That can be illustrated on the following example showing that the simply cheapest path cannot always be expanded. Let us assume a situation in a corridor of unit capacity resources with one sideway illustrated in the Figure 2.2. Agent A_1 wants to go from r_1 to r_5 and the agent A_2 is heading from r_5 to r_3. A_2 enters first and plans its route without problems. Then A_1 makes its plans. However, if A_1 would expand the cheapest path by going directly to r_2, it would collide with A_2. CARP prevents that by reserving the time window of A_2 passing through r_2 and letting A_1 wait in r_1.

Another important attribute can be shown in the example. The situation was solved with the order of agents taking A_2 first, followed by A_1. However, if A_1 would make its plans and reservations first, planning for A_2 would not be possible causing a deadlock as it would have nowhere to move aside from already set route of A_1. This importance of agents' order is one of the main points of this work and will be further addressed.

As more assignments are planned, and agents add their plans, the status of resources in a graph is updated providing a new situation for the newly planned ones. To ensure the ability of the latter planed agents to find a path and not be blocked by some of the previous agents in their finishing resources, some of the possible precautions have to be considered. That is either providing these locations with sufficient capacity or supposing that after fulfilling their task, agents depart from the infrastructure like airplanes from an airport. For automated warehouses applications, it can be assumed that agents either directly receive a new task or return to their waiting stand. Both would be considered as a new assignment to be planned.

In the algorithm presented in [1], multiple capacity of one resource is

considered. However, in this thesis, only one agent per resource is assumed. In applications allowing such solutions, large intersections with non-unit agent capacity can be divided into smaller, unit capacity resources forming, e.g., a roundabout allowing more predictable and effective behavior [6].

2.2 Algorithm

The algorithm takes agents sequentially finding optimal and collision-free routes for all of them. For each agent it runs a modified A^* search, which is described in Algorithm 1.

Algorithm 1 CARP - Plan route from [1]

Input: $a = \{r_s, r_g, t\}$ – assignment; start and goal position, start time
$G = \{V, E\}$ – a resource graph; vertices (free time windows) and edges
Output: shortest-time conflict-free route plan for a
$G = \{V, E\}$ – an updated resource graph

1: **if** $\exists\ w\ [w \in V \mid t \in \tau_{entry}(w) \wedge r_s = resource(w)]$ **then**
2: $\quad add(w, open)$
3: $\quad entryTime(w) \leftarrow t$
4: **while** $open \neq \emptyset$ **do**
5: $\quad w \leftarrow argmin_{w' \in open} f(w)$
6: $\quad add(w, closed)$
7: $\quad r \leftarrow resource(w)$
8: $\quad$ **if** $r = r_g$ **then return** $followBackPointers(w)$
9: $\quad t_{exit} \leftarrow g(w) = entryTime(w) + d(resource(w))$
10: $\quad$ **for** $w' \in \{\rho(r, \tau_{exit}) \backslash closed\}$ **do**
11: $\quad\quad t_{entry} \leftarrow max(\tau_{exit}, start(w'))$
12: $\quad\quad$ **if** $t_{entry} < entryTime(w')$ **then**
13: $\quad\quad\quad backpointer(w')$
14: $\quad\quad\quad entryTime(w') \leftarrow t_{entry}$
15: $\quad\quad\quad add(w', open)$
return null

The first line of Algorithm 1 ensures that the starting resource r_s carries a free time window in the time t. If so, the time window is marked as *entry* as it is shortened by the duration of the resource to ensure the fourth condition mentioned in section 2.1. Therefore, $\tau_{entry} = [t_1, t_2 - d(r_s)]$ where t_1 and t_2 are the starting and ending times of the time window respectively and $d(r_s)$ is the duration of r_s. If such time window exists, it is added to the open list and the entry time t is recorded.

While there are available free time windows in the open list or a plan was not found yet, the following process continues. A time window with the lowest heuristic cost $f(w)$ is taken from the open list at line 5 and is added to a closed list afterward. The heuristic is the same as in the standard A^* algorithm with the function $f(w) = g(w)+h(w)$, where $g(w)$ is the cost of the partial plan to w and $h(w)$ stands for a heuristic estimate to the goal. That can either be the Euclidean distance between the resource associated to w and r_g minimizing the total plan cost in an obstacle-free graph, or optimization of another aspect, e.g., makespan.

If the resource r of this time window is the goal resource, a set of back-pointers is followed and returned as the final plan. The plan will be expanded otherwise at line 9, where the first possible exit time is found by adding the resource duration to the entry time, marking the total partial plan cost to this resource. With $\rho(r, \tau_{exit})$, where $\tau_{exit} = [t_1 + d(r), t_2]$, standing for the set of all reachable time windows from r, the loop starting at line 10 iterates over all of these that are not closed. The entry time to the new time window w' is then set as the maximum of the earliest exit time from r and the earliest entry time to w'. At line 12, the expansion is only performed if w' was not already reached with an earlier entry time assuming the time windows' entry times are initially set to infinity. Then the backpointer is set between w' and w, the entry time of w' is recorded and w' is added to open list.

If the goal resource is found, it is returned at line 8. Otherwise, the time windows in open list run out and the assignment cannot be planned, returning null at the end.

As was mentioned in the algorithm description, the fourth condition from 2.1 is met at line 1. Cost from the second condition is minimized at lines 5 and 12. Finally, the first and third conditions are ensured by selecting from reachable neighbors carrying free time windows at the desired time at line 10.

In [1], the CARP algorithm with the time complexity of one A^* equal to $O(|A||R|log(|A||R|)+|A||R|^2)$, where A is the set of agents that already have a plan and R is a set of roads and intersections in graph, was presented. The time complexity was proved correct in [4]. For this work, a C++ implementation of this algorithm from [7] by Jakub Hvězda is used.

2.3 Characteristics

The standard CARP algorithm works with a set order of agents and in this order it attempts to plan their routes with the modified A^*. The disadvantage of this approach, therefore, comes from the inability to change the order of agents. As it was shown in the example in Figure 2.2, this has a major impact on the ability of the algorithm to either find a set of routes with lower cost or even to find some plans at all.

This issue was addressed for example by the author himself in [6]. Apart from already mentioned reduction of resource capacity, two simple heuristics were presented in that paper. The *Longest first* heuristic has a significantly lower failure rate and optimizes plan's makespan. However, the total cost is much higher. The second heuristic, on the other hand, is even more effective regarding failure rate and plan cost, but the time complexity of finding a solution is unbearable [6].

Another simpler solution is trying several (10, 100, 1000) random orders and attempting to plan each of them while finding the cheapest as well. This solution has a stable time complexity, and even ten shuffles provide significantly lower fail rate[1].

For all these solutions, however, it is essential to already have all of the assignments before planning any of them to find the best order, which can be inconvenient for some applications. It is possible to run CARP with random shuffling for each added agent, but the time complexity is steeply growing[1].

[1] The experimental results of *Longest first* heuristic and the CARP with random orders are presented in Chapter 4.

Chapter 3

Proposed algorithm

It was described in the previous chapter, that all improvements of the CARP algorithm require the knowledge of all assignments at the beginning of the planning process. However, a more convenient approach for real automated warehouse use is to consider assignments coming sequentially as requests, for example for goods, do. From its specification, only basic low effective version of the CARP is capable of addressing this task. According to that, several solutions are presented to improve the ability of the basic CARP to find a cheaper or at least some plan.

3.1 Agents influence

As was presented in the CARP description and proven by its simple improvements, the key to better efficiency regarding fail rate is choosing the right order in which agents are planned. Combined with a sequential mechanism of adding assignments to the process, a tool is needed to pick a relatively small group of agents which would change their plans to allow newcomers find theirs. Replanning of a large group of agents while finding their best order (for all combinations of n agents that is $n * n!$ calls to one A^* search of the CARP) would be time costly.

3.1.1 Theory

Selection of the group of agents comes from the idea that agents' paths influence each other. If minimization of this influence is achieved by finding a proper group of the most influencing agents, the overall results of the algorithm might improve. The first task is, therefore, to quantify the influence.

The definition of influence in this thesis is based on the Euclidean distance of resources among agents' routes. The closer agents get, the more they influence each other in the way that the latter planned one has to adapt to the previous. As this happens, the order of agents gains significance as was shown in an example in the previous chapter. Once such pair or group of influencing agents is found, several variants of their order can be examined.

3.1.2 Distance matrix

To find a specific group of agents based on their routes distance, a distance matrix has to be computed. Agents' paths are represented by sets of resources with time windows stating the time and duration of resource usage. Measuring the distance of these paths, therefore, has to work not only with distance but with time as well. The process of going through plans and calculating the distance matrix is presented in Algorithm 2.

At the first line of Algorithm 2, a temporary matrix is prepared with the desired size, as it represents relations between every two plans in the set. The plans are then taken in the `for` loops while omitting a comparison of a plan with itself. The loop starting at line 7 then iterates through both of the picked plans finding all pairs of resources in the closest moments.

Finally, between lines 9 and 20, the plans are compared. First, the time window of each current step is taken. These time windows are compared and if they intersect, resources from that moment are saved at lines 13 and 14. Five possible relations of time windows can appear, and apart from identity, all are illustrated in Figure 3.1. If the time windows do not intersect, one of them is moved forward concerning the position of the other one. This movement is marked in Figure 3.1 as well. The loop iterates through the plans until it finds a pair of steps from both plans that happen at the same time.

Algorithm 2 Distance matrix

Input: $P = \{plan_i\}_{i=1}^{n}$ – a set of n planned routes
Output: M – a square matrix representing average distances between routes

```
1:  N(n)
2:  for planA ∈ P do
3:      for planB ∈ P do
4:          N[planA][planB] = ∅
5:          if planA == planB then
6:              continue
7:          for stepA, stepB ∈ planA, planB do
8:              notSet = true
9:              while notSet do
10:                 twA = planA[stepA].timeWindow
11:                 twB = planB[stepB].timeWindow
12:                 if tw1 ∩ tw2 then
13:                     resourceA = planA[stepA].resource
14:                     resourceB = planB[stepB].resource
15:                     notSet = false
16:                 else
17:                     if twA.start < twB.start then
18:                         stepA++
19:                     else
20:                         stepB++
21:             M[planA][planB] ∪ dist(resourceA, resourceB)
22: for planA ∈ P do
23:     for planB ∈ P do
24:         M[planA][planB] = Φ(N[planA][planB])
    return M
```

The Euclidean distance of the two found resources is counted, represented as $dist()$, at line 21 and added to the appropriate column in the distance matrix. At lines 22-24, the matrix is finalized to the output matrix M by the function Φ, representing, e.g., average. Another possible representation is to emphasize closer parts of paths over the further by taking the average of the square value of their distances. The third approach used in this thesis is to consider only the closest moment of the plans and therefore return the minimum of the plans' distances.

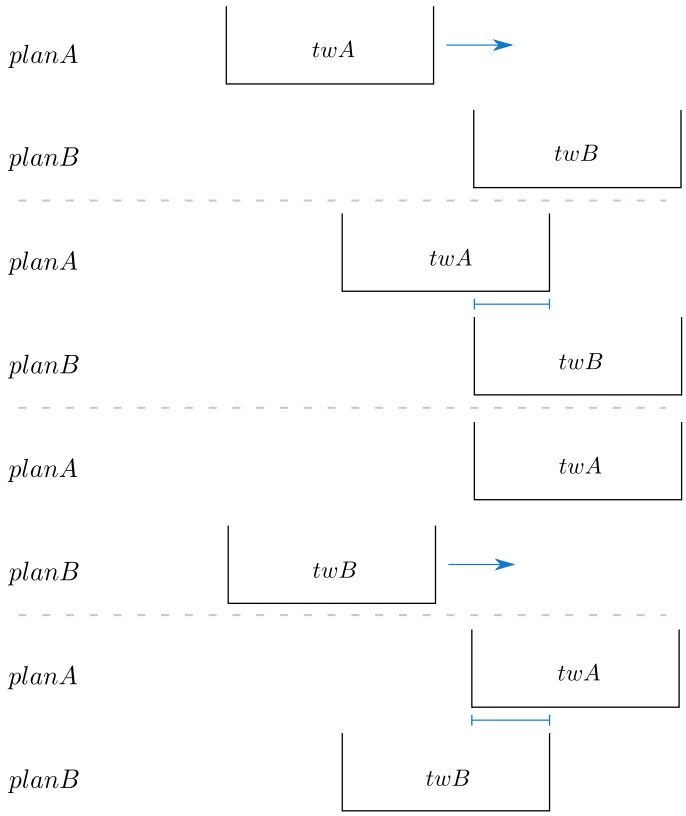


Figure 3.1: Possible time windows relations (without identity) with marked intersections or needed shift.

3.2 Algorithms

With the tools from section 3.1, the CARP algorithm can be adjusted to take each *j-th* assignment at a time and attempt to find its solution not only based on but also adjusting the previous process according to *j-th* agent's influence with the previously planned ones. The ability to work with an incomplete set of assignments is preserved, as in any moment of already ongoing task completion the path can be changed.

Two possible usages are presented. Both start their *j-th* step by finding an optimal route for the new agent with A^* algorithm adapted to the resource graph. It does not consider the occupancy of resources, but it notes down the time windows of its path to make algorithm from 3.1.2 possible.

3.2.1 Direct group

The principle of this approach is to directly find a group of l agents whose routes have the largest influence on the ideal route of the *j-th* agent. This group is taken out of the plans and attempted to be added the most effective way. *j-th* step of the adding process is presented in Algorithm 3.

Algorithm 3 Direct group

Input: $P = \{plan_i\}_{i=1}^{j-1}$ – a set of j-1 planned routes
$A = \{a_i\}_{i=1}^{j-1}$ – a set of j-1 planned assignments
$a_j = \{r_s, r_g, t\}$ – *j-th* assignment; start and goal position, start time
l – size of a group
$G = \{V, E\}$ – a resource graph; vertices (with free time windows) and edges
Output: $P = \{plan_i\}_{i=1}^{j}$ – a set of j updated routes

```
plan_j = A*(r_s, r_g, G)
P ← P ∪ plan_j
A ← A ∪ a_j
M = count distance matrix of P
C ← a_j
C ← take l closest from A based on M
newP
for a ∈ A do
    if a ∉ C then
        plan ← CARP(a, G)
        newP ← newP ∪ plan
best = ∞
for π ∈ Π(C) do
    for a ∈ π do
        plan ← CARP(a, G)
        thisP ← thisP ∪ plan
    value = cost(newP ∪ thisP)
    if value < best then
        P ← newP ∪ thisP
        best ← value
return P
```

At the first line of code, *j-th* assignment is planned and then added to the group of plans P although it may cause collisions. That is possible because P will not be returned at this state. If the algorithm would fail at any point, however, *j-th* agent would attempt to find a correct collision-free route with the CARP, and if successful, this would replace the A^* version. The assignment is added to A as well.

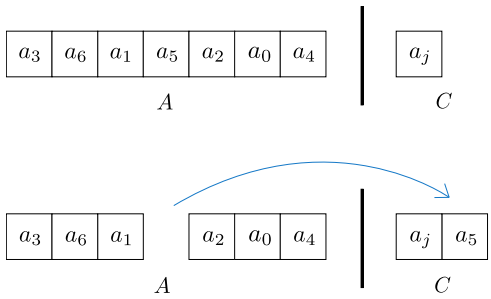


Figure 3.2: One agent taken to group.

Based on P, at line 4, a distance matrix is processed according to Algorithm 2 in 3.1.2, providing a numeric representation of influence between agent's routes. C is the group of replanned agents firstly joined by j-th. At line 6, the closest agent is added to C, based on the distance matrix. This process is repeated until the group has the desired size l or until all present (j) agents are in it. Each new agent to be chosen is compared to all members of the group, so it is the closest to the whole group.

At that moment, two groups of agents are distinguished. The order of those not taken to the cluster is preserved in A, and they go through CARP again for a new set of plans no longer considering routes of agents present in C (lines 8-11). Otherwise, new planning would expect the map resources to be occupied by agents that are no longer present and the plans would therefore not be optimal.

Finally, in the `for` loop starting at line 13, all permutations of the order of assignments in the group C are planned, each based on the context of the updated staying agents' plans in $newP$. Out of these, the one with the shortest overall Euclidean distance is chosen at line 18 and attached to the rest. If any better plan is found, it substitutes the previous best. This selection implements optimization of the time cost in the set of plans which is the target of this work.

3.2.2 Growing Group

The approach with *Growing Group* is mostly similar to the *Direct Group* version with the only difference in attempting to find plans for each state during the addition of agents to the cluster. It is shown in Algorithm 4.

In each iteration of the `for` loop starting at line 6, one agent is taken out

Algorithm 4 Growing group

Input: $P = \{plan_i\}_{i=1}^{j-1}$ – a set of j-1 planned routes
$A = \{a_i\}_{i=1}^{j-1}$ – a set of j-1 planned assignments
$a_j = \{r_s, r_g, t\}$ – j-th assignment; start and goal position
l – size of a group
$G = \{V, E\}$ – a resource graph; vertices (with free time windows) and edges
Output: $P = \{plan_i\}_{i=1}^{j}$ – a set of j updated routes

1: $plan_j = A^*(r_s, r_g, G)$
2: $P \leftarrow P \cup plan_j$
3: $A \leftarrow A \cup a_j$
4: $C \leftarrow a_j$
5: $best = \infty$
6: **for** $k \in l$ **do**
7: M = count distance matrix of P
8: $C \leftarrow$ take 1 closest from A based on M
9: $newP$
10: **for** $a \in A$ **do**
11: **if** $a \notin C$ **then**
12: $plan \leftarrow CARP(a, G)$
13: $newP \leftarrow newP \cup plan$
14: **for** $\pi \in \Pi(C)$ **do**
15: **for** $a \in \pi$ **do**
16: $plan \leftarrow CARP(a, G)$
17: $thisP \leftarrow thisP \cup plan$
18: $value = cost(newP \cup thisP)$
19: **if** $value < best$ **then**
20: $P \leftarrow newP \cup thisP$
21: $best \leftarrow value$
return P

of A based on the same distance matrix as was presented earlier. Taking an agent to the group is illustrated in Figure 3.2, also showing that the rest of A preserves its order. Then the process is the same as in the *Direct Group* adding, as agents in A are replanned and finding the best permutation of C follows. This process is then repeated until the group has size l.

The main difference is that the total Euclidean distance of the final plan is compared among all phases of the `for` loop at line 6. The cheapest one is therefore picked from $l - 1$ times larger range. The ability to use a lower size group from during the process also gives a higher probability of finding some solution, as shown in Chapter 4.

0	1	2	3	4
0	1	2	4	3
0	1	3	2	4
0	1	3	4	2
0	1	4	2	3

Figure 3.3: Preserving partial plans in permutations.

3.3 Optimalization

To reduce the number of times the CARP A^* is called, which increases the total time to find all plans, several steps can be taken. Firstly, when planning the agents that don't belong to the group C, as many plans from the past as possible are preserved. That is possible because their context stays the same which is illustrated in Figure 3.2, showing that three agents can keep their plans. The first to be replanned comes after a_5 which left A, and its route should no longer be considered.

The second simplification is quite similar. During the planning of the permutations, there is always a variable group of plans that can stay the same as in the previous permutation, as visualized in Figure 3.3. Assignments on indexes marked yellow don't need to be planned again. This requires the permutations of agents in C to be in lexicographic order. The number of times the CARP A^* search is called is reduced significantly, as for $l = 4$, instead of $4 * 4! = 96$ it is only 64 calls. For $l = 5$ that is 325 instead of 600 calls and for $l = 6$ only 1956 instead of 4320 calls are needed.

Chapter 4

Experiments

4.1 Motivation

Several characteristics of planning algorithms are significant for their evaluation. Firstly, for practical use, algorithms should generate plans in a reasonable time. From the tenet of this work, not only overall time to plan all agents is significant, but also the time to plan each agent in the group has important value. The second part of the characteristics should describe the plans themselves. That includes failure rate, the ability of an algorithm to find any plan, and several quality characteristics, such as makespan or plan cost.

4.2 Experiment setup

For the purpose of the experiments, a set of 21 maps was created. Firstly, a random complete square graph was generated with the size 20*20 vertices. The graph was then simplified to a spanning tree and, finally, approximately 50 random edges were added 20 times, to create the set of 21 maps of density ranging from 800 to 1500 edges in the graph. An example of a generated map with 1350 edges is in the Figure 4.1.

All characteristics were then measured on the planning of 500 random assignments for 100 agents on each of the 21 maps. That provides a set of 10 500 results from each algorithm. The experiments were performed on a computer equipped with Intel Xeon E5-2690.

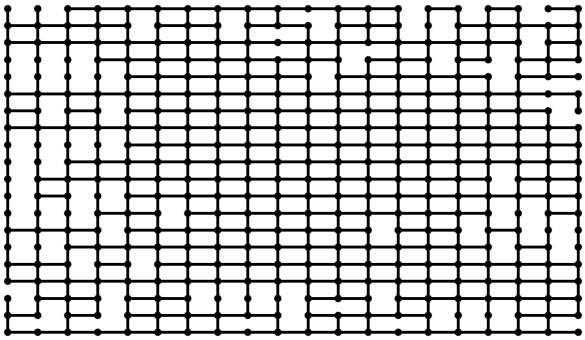


Figure 4.1: Generated map with 1350 edges.

4.3 Compared algorithms

Four versions of CARP algorithm were taken for comparison. Firstly, the original algorithm from [1] as `CARP`. The second group consists of the adjusted CARP algorithm taking 10 or 100 random orders of given assignments and attempting to find the cheapest among them (`CARP10`, `CARP100`). Finally, it is the CARP with *Longest path first* heuristic presented in [6] as `LF`.

More variations of the proposed algorithm (`DG` for *Direct group* and `GG` for *Growing group*) were tested as well. One differentiation can be made in the choice of the Φ function bringing different distance matrices as defined in section 3.1.2. Another way is considering the distance matrix not based on the actual context-aware plans, but on optimal plans from A^*. After each agent joins the process, its A^* plan is saved, and the distance matrix is always only updated with one row and one column for the j-*th* agent. That is represented by `DGA`* and `GGA`* in the following figures.

The main difference is, however, made in the setting of l, which defines the maximal size of the group C. That was chosen in the range between 4 and 6 based on initial tests to achieve sufficient quality in a sustainable time. The size of the group is marked for each algorithm, e.g., `DG_4`, standing for *Direct group* with group C consisting of four assignments.

Special versions with the size of up to 10 were tested as well. For these, the process was adjusted by directly planning the first ten assignments to

ensure a group C of ten in each adding. The other adjustment aiming at time cost reduction was taking only 150 (less than 6! in `DG_6` and `GG_6`) random permutations of the group C sorted lexicographically for partial plans memory. For the *Growing group* version, this is only used when the size of C grows over 6 members.

4.4 Measuring

The failure rate is counted as an average proportional inability to find a plan. Fail, for the proposed approaches, means, that the method itself was not able to find the *j-th* plan and it cannot be simply attached with the CARP A^* to the previous set of plans as well. For the compared CARP algorithms that is at the point when none of the random orders was possible to be planned.

Makespan determines the number of steps that are needed for all agents to reach their target. In fact, it describes how long it takes the last agent to finish its assignment. Joint plan cost shows, on the other hand, is the combined price of all routes in the final plan. Unitary length of edges is assumed. The cost of one plan is then the sum of all edges traveled plus 1 per each time agent waits in one node.

For the results of makespan and joint plan cost, only assignments, where all algorithms were successful are considered. This narrows the set of results but ensures the comparison among an appropriate group of values.

Time was measured from the point where algorithm receives the set of assignments to the moment where it plans all agents or when it fails to plan some of them and is not able to continue. The other time characteristic measures the time the algorithm takes to plan one agent on a map with 1350 edges. For the proposed algorithms, it is the time of one iteration of the loop to add *j-th* agent.

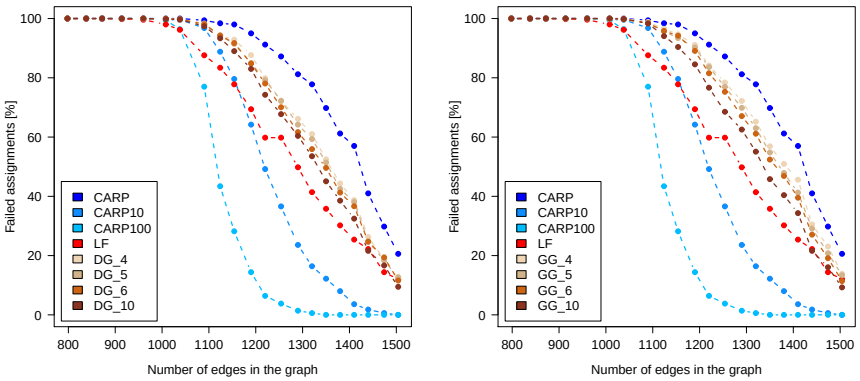
For the second time characteristic, CARP algorithms were modified to present a set of plans for each *j-th* agent as well to have a proper comparison. That means that for each agent, the algorithm creates all n random orders of agents and attempts to plan them. This not only significantly affects the duration of algorithms − with j agents and p random orders, the CARP A^* is called $j! * p$ times instead of $j * p$ times − but failure rate as well. That is because the algorithm has more chances to make a mistake, as it has to succeed in each of the j iterations and not only in one as in standard version.

4.5 Results

4.5.1 Failure rate

An improvement of about 10% - 20% can be seen in both proposed approaches regarding failure rate when compared to `CARP`. All of their versions bring a higher probability of finding a set of plans then the standard CARP algorithm. For both `DG` and `GG` none of the variants of the distance matrix differed distinctively, so the results are grouped for better readability of the Figure 4.2. The most successful combination out of all proposed versions is `GG_10` with the distance matrix based on context-aware plans and counting average path distances.

The heuristic behind `LF` shows that a simple sorting of assignments according to their ideal length pulls the failure rate down significantly, but the difference from the proposed algorithm is under 10% for most of the maps. Trying 10 or 100 random orders appears as the most efficient, but the need for the knowledge of all assignments has to be noted.



(a) : Direct group. **(b) :** Growing group.

Figure 4.2: Failure rate.

4.5.2 Joint plan cost

Optimization of a joint plan cost is the target of the proposed algorithm, and the results correspond to that. The experiments show that the size of the group has minor influence and even with four members the cost is similar

to the best application of the CARP in `CARP100`. With no surprise, the `LF` performs much worse as it aims at optimization of makespan at the expense of joint plan cost. The different results of the CARP algorithms in the Figure 4.3 are caused by the method of data processing, where only successful plans among all algorithms were taken for each Φ. That is also the cause of the large changes of the values in the first maps, as the failure rate is more significant in these maps and the number of results that passed through is lower. Sufficient credibility is, therefore, achieved in the latter maps.

With the growing number of edges, not only failure rate decreases, but it also allows the algorithms to achieve better results. The `DG` performs best with the distance matrix counted from the ideal A^* plans and a larger group C, although it matches the quality of the standard CARP in many of the maps. The choice of the distance matrix has a low influence on the performance of `GG`, which gives better results than all CARP versions in all of the maps even with the group C of 4 assignments at most. The best `GG_10` has an average of 40 actions less than the `CARP`.

4.5.3 Makespan

Out of all algorithms compared in this thesis, only the *Longest first* heuristic aims at the optimization of makespan. Therefore, for the rest of algorithms, it is mostly an informative characteristic showing the side effects of each approach. That is also the reason for `CARP10` and `CARP100` being even worse than `CARP` on some of the maps in the Figure 4.5. However, all versions of the presented algorithm perform better than `CARP` altogether, with mostly similar results in all versions.

4.5.4 Time

The last attributes to be discussed are the time characteristics, which simply reflect how many times the CARP A^* is called. For standard `CARP` with $n = 500$ agents that is 500 times. The `LF` adds just the time to plan agents with a standard A^* to sort them, so it lasts less then a double time of `CARP`, while `CARP10` and `CARP100` multiple n by 10 or 100 respectively. Regarding the proposed algorithms, the time complexity grows with the number of agents in the group. Considering the optimization steps as presented, the number of times CARP A^* is called can vary with the worst case of replanning all agents taking $(j - l)! * l!$ and $(j - l)! * (l!)^2$ for `DG` and `GG` respectively.

For the test of the time that each algorithm takes to plan one assignment when it is added to the process, it is important to note again that the CARP algorithms were modified to present a complete set of plans each time new agent assignment is added.

The proposed algorithm among all methods of counting the distance matrix performed very similarly in these characteristics, so the results are grouped again in Figures 4.7 and 4.8. In this case, that is expected, as the number of actions is generally the same and is changed only by the choice of the group C and a possible moment of failure. In the set of experiments, both have a low statistical influence.

Regarding the time to plan all assignments directly, all versions of the standard CARP overperform the proposed algorithms unsurprisingly. Their results correspond to the size of the group C and the repeated process in `GG`.

It can be seen that although groups of 6 or 10 increase the duration of planning strongly, the total time for 500 agents stays under 105 s and during the process it is under 1.3 s per assignment. Lower time cost is achieved by `DG` thanks to $(l-1)!$ times less attempts to attach the best permutation of the group C than in `GG`. Similar or even better results of `DG_10` then `DG_6` are caused by the fact that it works with only 150 permutations instead of 720 in `DG_6`. The same applies for the times from during the process in the Figure 4.8 as well. This phenomenon is suppressed in `GG_10`, which shows that for such size of the group C, the repetition of the adding process has a higher influence on the time cost than the planning of the permutations itself.

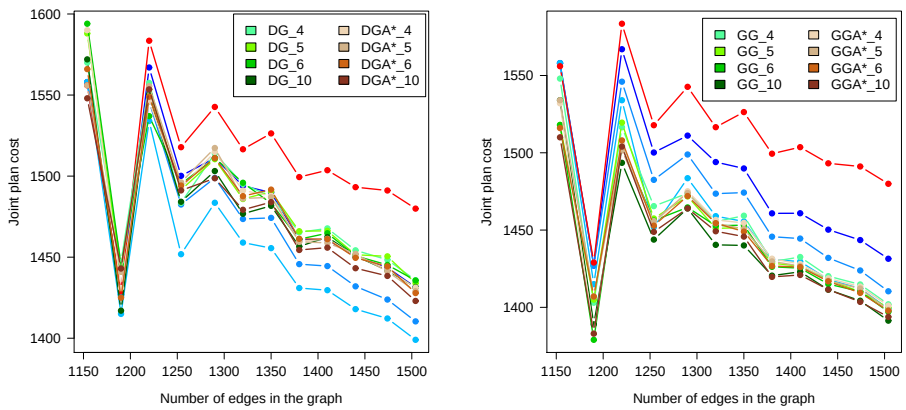
The second figure also shows that the time cost of `DG_10` and `GG_10` starts to grow later. That is caused by directly planning the first 10 agents and starting the process at assignment 10. The main outcome of this test, however, is the radically growing duration of planning in `CARP100`. That shows the main contribution of selective picking of the replanned assignments.

4.6 Summary

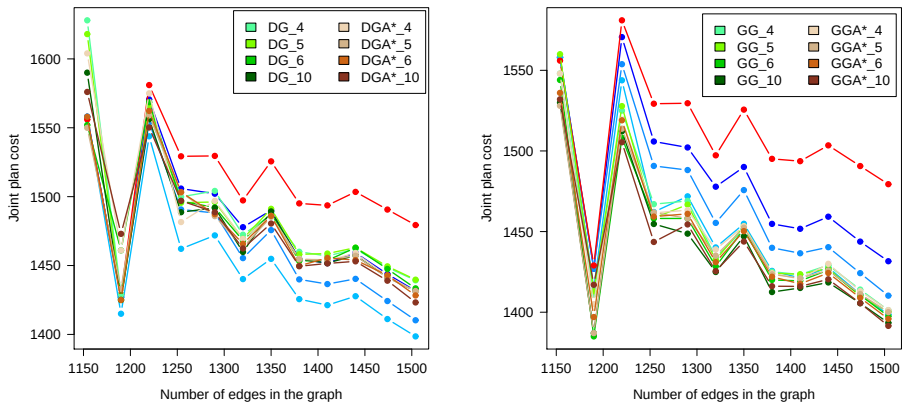
The experiments' results show that the way the distance matrix is computed has a low influence. What really affects the quality of the results is the size and method of adding to the group C. The more successful solution is giving the algorithm the ability to compare partial plans among all running sizes

of the group C, which provides the best solution out of a larger set. This, however, has an increasing time cost, so the size of the group is limited.

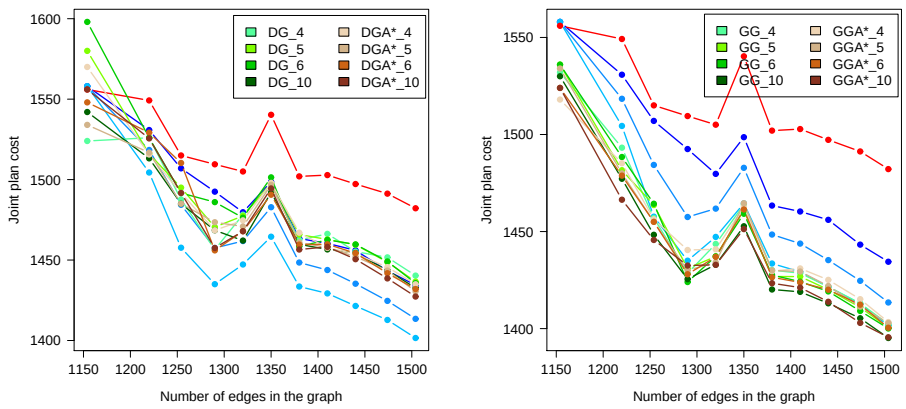
Out of all tested approaches, both `GG_6` and `GG_10` reached the best quality results with the `GG_10` having the better at the expense of a higher time cost. On the other hand, a thing worth noticing is that increasing of the size l brings only a limited improvement and the affordable time cost is upon consideration, as even `GG_4` reaches the success rate of `GG_10` in some cases and overperforms `CARP100` in plan quality. In the overall time characteristic, it is slightly better than `CARP100` and about 40 times faster than `GG_10`.



(a) : Φ as average

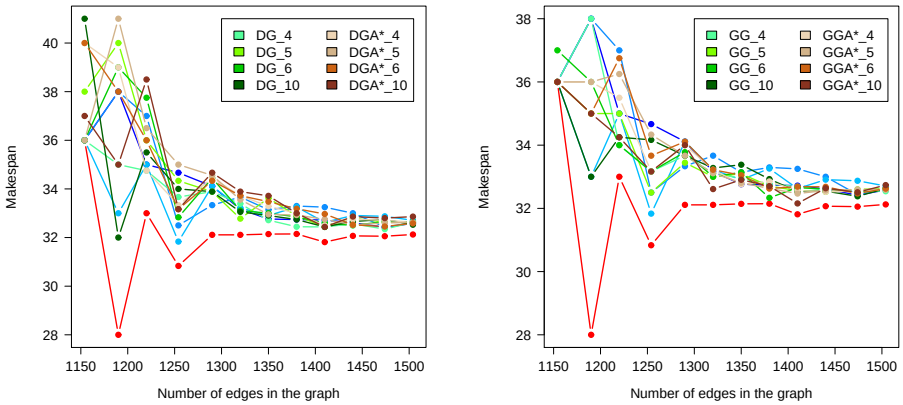


(b) : Φ as square average

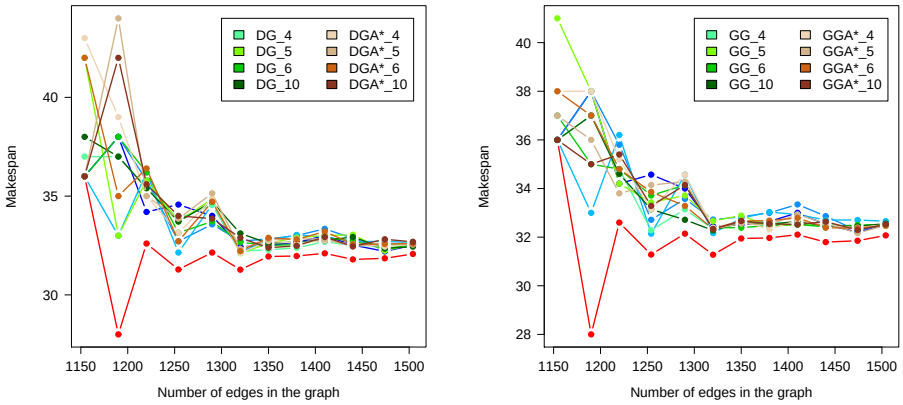


(c) : Φ as a minimum

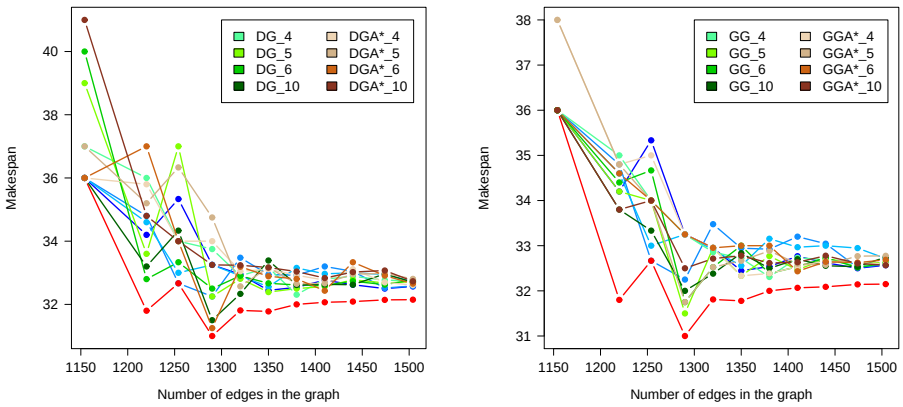
Figure 4.3: Joint cost, `DG` in the left column and `GG` in the right column.



(a) : Φ as average

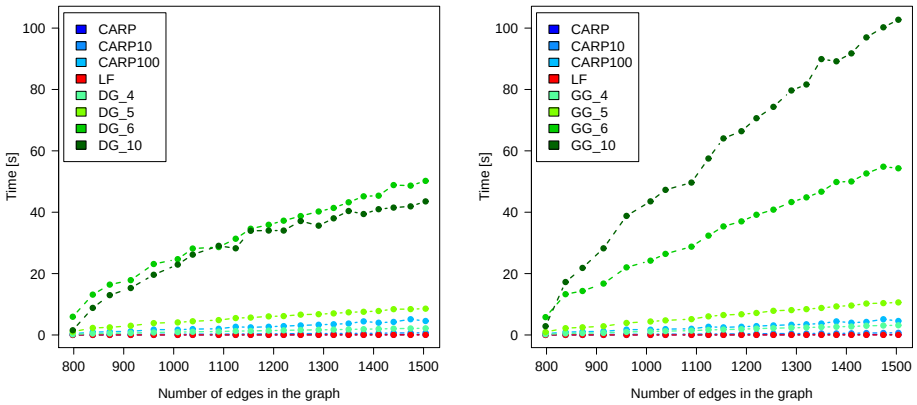


(b) : Φ as square average



(c) : Φ as a minimum

Figure 4.5: Makespan, `DG` in the left column and `GG` in the right column.



(a) : Direct group. **(b) :** Growing group.

Figure 4.7: Time.

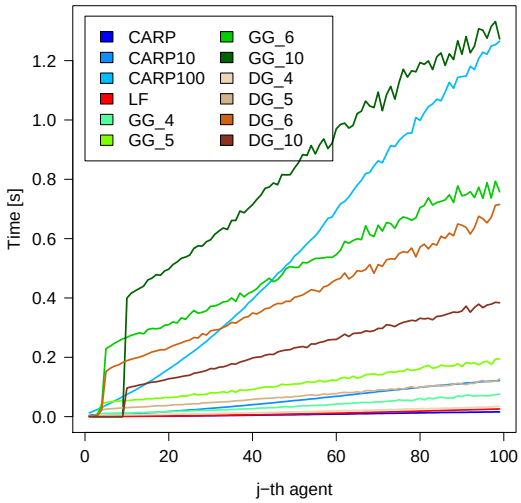


Figure 4.8: Time to plan each assignment when received.

Chapter 5

Conclusions

In this thesis, the task of route planning in automated warehouses was presented with the aim at a sequential assignment income. The Context-aware route planning algorithm was described in Chapter 2 with all its main attributes regarding the needs of the task. Based on the CARP specifications, an approach of partial changes in plans during adding of assignments was suggested with several possible modifications.

The proposed approach was then compared to the CARP and it's older improvements in a set of experiments aimed at several important characteristics of all algorithms. Out of these, the main success is in improving both fail rate and joint plan cost, which was the initial task, in comparison to the original CARP algorithm that is able of sequential adding of assignments. After modification to be capable of this, the least failing versions of the CARP with random orders are overperformed in the duration of planning for one agent, while it still trails regarding plan cost.

The results in makespan show that it can be pulled down by the optimization of joint plan cost, but for an application with such need, the proposed algorithm can be easily modified to minimize this characteristic more significantly.

The main quality of the proposed approach, in any of its versions, during the target process is the ability to preserve most of the previous setting, while being able to update it to better final quality. With the testing set of 100 agents, the used sizes of the group C work with just around 5% of the total number of agents, yet they are able to find a better solution. The

versions with C of the size ten performed at the same or better level while trying fewer permutations, than with the groups of six. This shows that the number of assignments moved has higher influence than the number of tested orders leading back to the CARP versions with a random shuffling of all assignments. On the other hand, it was shown that even when replanning only four assignments, the results can improve at a significantly lower time cost.

For practical use, a combination of discussed algorithms can be suggested. If the cheapest performing algorithm fails at any time of a process, it can be substituted by, e.g., `CARP10`, to find a solution in the situation, and then continue as before. The process of *Growing group* could also be modified to grow the C group only until a solution is found, which would save time for having a higher size limit for the cases requiring a larger correction. Adding more than one agent in each iteration in *Growing group* is another possible adjustment.

An important weakness of the performed experiments has to be pointed out as well. In an actual warehouse, agents usually reach the target rack, load goods and head to a picking station to repeat the process. This contributes to a natural flow of agents joining the queues at the picking stations, which makes the overall planning more effective [6]. In the experiments, however, all assignments were generated randomly, possibly creating senseless situations. The future work should, therefore, include a better testing approaches' development to provide real-life conditions, as well as a larger scale of testing with more assignments in larger graphs.

Further research into the aspects optimizing the joint plan cost will also be contributive. As was proved by the *Longest first* heuristic from [6], ordering assignments according to their optimal plan length optimizes makespan effectively. Such ordering heuristic leading to lowering of joint plan cost would help the sequential algorithm as well. Another approach worth developing is extending the grouping methods from this thesis to create several groups of agents moving in the same direction to simplify their movement, similarly as the third heuristic presented in [6]. Reduction of plan cost by organizing agents to such flows was observed by research in [4].

The approaches and methods presented in this thesis were used in an article appointed for the 21st IEEE International Conference on Intelligent Transportation Systems (ITSC 2018) held in Maui, Hawaii, USA on November 4-7, 2018 [8].

Appendix A

Bibliography

[1] Adriaan W. ter Mors, Cees Witteveen, Jonne Zutt, and Fernando A. Kuipers. Context-aware route planning. *Delft University of Technology, The Netherlands*, 2011.

[2] W. Matthews and S. Dawson. The shed of the future e-commerce: its impact on warehouses. *Online, available: http://www2.deloitte.com/uk/en/pages/real-estate/articles/shed-of-the-future.html*, 2014.

[3] Sebastian Trüg, Jörg Hoffmann, and Bernhard Nebel. Applying automatic planning systems to airport ground-traffic control - a feasibility study. *KI, pages 183-197*, 2004.

[4] A.W. ter Mors. The world according to marp: multi-agent route planning. *Delft University of Technology, PhD thesis*, 3 2010.

[5] L. E. Parker. Path planning and motion coordination in multiple mobile robot teams. *Encyclopedia of Complexity and System Science*, 2009.

[6] Adriaan W. ter Mors. Evaluating heuristics for prioritizing context-aware route planning agents. *Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, The Netherlands*, 4 2011.

[7] J. Hvezda. Comparison of path planning methods for a multi-robot team. *CTU in Prague, Master thesis*, 2017.

[8] J. Hvezda, T. Rybecky, M. Kulich, and L. Preucil. Context-aware route planning for automated warehouses. *21st IEEE International Conference on Intelligent Transportation Systems (ITSC 2018), in review*, 2018.

Appendix B

Enclosed CD contents

The root directory of the enclosed CD contains the following items

- thesis.pdf: This thesis
- figures: A directory containing all presented figures
- source: A C++ project containing the implementation of all mentioned algorithms
- tex: A LaTeXproject of this thesis
- stats: Source codes in R for the statistics processing
- readme.txt: Instructions to running the C++ project and the R statistics, list of CD contents

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Rybecký Tomáš** Personal ID number: **457220**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

Branch of study: **Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Planning for a Team of Cooperating Mobile Robots

Bachelor's thesis title in Czech:

Plánování pro tým kooperujících mobilních robotů

Guidelines:

1. Get acquainted with current approaches to collision-free path planning for a team of cooperating robots.
2. Design a method for selection of a fixed-size subset of most dependent trajectories.
3. Design a method which sequentially plans a trajectory for particular robots in a team. To do that, use a method developed in step 2.
4. Design and create a set of testing scenarios.
5. Experimentally evaluate properties of the implemented algorithm. Describe and discuss obtained results.

Bibliography / sources:

[1] A. W. ter Mors, Conflict-free route planning in dynamic environments, 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, San Francisco, CA, 2011, pp. 2166-2171.
[2] K. Solovey, O. Salzman, O. and D. Halperin, Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning. Algorithmic Foundations of Robotics XI , 2013, pp. 591-607.
[3] J. Hvězda, Comparison of path planning methods for a multi-robot team, CTU in Prague, Master thesis, 2017

Name and workplace of bachelor's thesis supervisor:

RNDr. Miroslav Kulich, Ph.D., Intelligent and Mobile Robotics, CIIRC

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **11.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

RNDr. Miroslav Kulich, Ph.D. Supervisor's signature	doc. Ing. Tomáš Svoboda, Ph.D. Head of department's signature	prof. Ing. Pavel Ripka, CSc. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt	Student's signature