Agent Based Modelling and Simulation

Assignment

J. van Beek
J.M. Hinsenveld

November 20, 2022

Technische Universiteit Delft

TUDelft Delft University of Technology

Challenge the future

AGENT BASED MODELLING AND SIMULATION

ASSIGNMENT

Agent Based Modelling and Simulation (AE4422-20)

Master of Science (MSc)
Faculty of Aerospace Engineering

Technische Universiteit Delft
Delft, The Netherlands

Authors:

J. van Beek	4544439
J.M. Hinsenveld	4368908

CONTENTS

1 Introduction

The application of Agent based modelling as a method for path finding is promising. Applications such as vehicle routing, and aircraft taxiway networks could benefit from the fast, reliable results without compromising safety if modelled correctly.

The goal of this report is to develop and evaluate three models commonly used in optimization. The goals of these planners is to optimize the paths of agents travelling on a grid from predefined start to a predefined goal, without any collisions between the agents. Predefined objects block agents from travelling onto certain locations. The first model is prioritized planning, in which agents have priority over other agents. The second model is Conflict Based Search (CBS), in which agents create a path, and conflicts are identified by an orchestrating agent. The last model is distributed planning, which lets the agents find a path, and negotiate with other agents. These models are elaborated upon in chapter 2, chapter 3 and chapter 4 respectively. Next, these three models are analyzed, and compared on speed, accuracy, robustness, scalability, and safety and some recommendations on the distributed planner are given in chapter 5.

2
Prioritized Planning

In the prioritized planning algorithm, all agents are given a value of priority. The first agent is free to use the shortest path to its destination within the feasible region. The second agent is constrained from moving to a an edge or node currently occupied by the first agent, such that there are no collisions. A third agent gets the same constraints as the second agent, as well as new constraints from the second agent path.
In the algorithm, all agent paths are made by an orchestrating agent. This agent gives constraints to lower priority commodities, and creates the paths accordingly.
The solution of prioritized planning could be optimal, but with more commodities, and starting and end points close to each other, this becomes more and more unlikely. In some cases there is no solution possible, since the lower priority agent gets too many constraints, and might be blocked by a higher priority agent which has already reached its destination. This could be fixed by implementing a function which makes sure all commodities have a feasible path to reach their destination, for example by increasing priority of a lower priority agent.

2.1. Model Specification

2.1.1. Specification of the Environment

The environment is made of grey and white areas. The white area is accessible to the agents, and the grey area is not. There are three different maps, as shown in fig. 5.4. The environment is Deterministic, Static, and fully Accessible.

2.1.2. Specification of Agent Types and Characteristics

All agents in the prioritized model are the same, except their priority and color. They are proactive, and create their own paths based on the shortest path, and constraints from the higher priority agent. The agent characteristics are the following:

1. At timestep 0, all agents are given a start location on the map, and have a goal location.
2. An agent is allowed to remain in its position, or move one step in any direction per timestep.
3. For all timesteps: if an agent is not yet at its goal, it will move towards the goal using the shortest feasible path.
4. All agents are constrained from moving to an edge or node which is occupied by a higher priority agent.
5. All agents will not move to a grey area of the environment

2.1.3. Cognitive and Behavioral Properties

The first agent creates the shortest path to its destination. It observes the environment, and takes action by finding a path. All following agents do the same, except with more constraints from the higher priority agent. It observes the location of the higher agents, and takes action by not occupying the same nodes or edges at the same time. There is no further interaction between agents.

2.2. SEMI-FORMAL REPRESENTATION OF SINGLE AGENT PLANNER

In this section, the alterations of the single agent planner algorithm will be shown in a semi-formal representation, with respect to the originally available file. Some original functions of the algorithm are shown for clarity.

2.2.1. A STAR

build constraint table with **build_constraint_table(constraints, agent)**

while open list is not 0,
current node is popped from open list

if current location is goal,
and if time larger than largest time in constraints:
return current path
else if current location is goal,
if no more constraints at goal in future, return current path
else continue search

for each direction
move current agent
if move constraint or if move onto obstacle or out of map:
next direction
else:
make child

if child visited before, compare heuristics, push lowest cost node
else:
push child node

2.2.2. MOVE

difference: added (0,0) to possible directions

2.2.3. BUILD CONSTRAINT TABLE

(constraints, agent)
for constraint in constraints
if constraint belongs to agent,
make constraint table, indexed by timestep
append constraint to list in constraint table at appropriate index(timestep)

return constraint table

2.2.4. IS CONSTRAINED

(current location, next location, next time, constraint table)

goal constraints and timesteps when reached by agent stored at constraint table index 0 retreived
loop over goal constraints
if goal constraint is the next location,
return true

loop over constraints

```
if constraint is vertex constraint
if constrained location is next location
return true

if constraint is edge constraint
if the move is constrained
return true
```

2.2.5. SEMI-FORMAL REPRESENTATION OF PRIORITIZED

In this subsection, only the added code to the prioritized agent planner will be given in semi-formal representation.

```
for agent in agents
for location in path
if agent is not itself
add location to constraints
if location is not start:
add edge to constraints

add goal to all constraints of all other agents
```

3 Conflict Based Search

In Conflict Based Search (CBS), the shortest path of all commodities is calculated using the A^* algorithm. Only if there are conflicts, constraints are added, such that the total path length is the shortest. The algorithm stops if there are no more conflicts, and the optimal solution is found.

3.1. Model specification

3.1.1. Specification of the Environment

The environment is made of grey and white areas. The white area is accessible to the agents, and the grey area is not. There are three different maps, as shown in fig. 5.4. The environment is Deterministic, Static, and fully Accessible.

3.1.2. Specification of Agent Types and Characteristics

There is one orchestrating agent, which is proactive, and assigns constraints to the other agents. All other agents in the CBS model are the same except their color. These agents are reactive, and create a new path after receiving constraints from the orchestrating agent. The agent characteristics are the following:

1. At timestep 0, all agents are given a start location on the map, and have a goal location.
2. An agent is allowed to remain in its position, or move one step in any direction per timestep.
3. For all timesteps: if an agent is not yet at its goal, it will move towards the goal using the shortest feasible path.
4. If the two agent paths cross on an edge or node on the same timestep, a new path will be created
5. All agents are constrained from moving to an edge or node which is occupied by a higher priority agent.
6. All agents will not move to a grey area of the environment

Cognitive and behavioral properties The orchestrating agent assigns constraints to the other agents if there is a conflict. So the agent observes the paths of all other agents and takes action by assigning the proper constraints. All other agents receive a constraints from the orchestrating agent, and create a new path. They observe the map, constraints, and their location, and take action by finding the shortest path. There is no further interaction between moving agents, but only with the orchestrating agent.

3.2. Semi-Formal Representation of Conflict Based Search

3.2.1. Find Solution

in Class CBSSolver

Function find solution
Initiate the root node.
find and add a path with A* for each agent in self.num of agents.
find and add the cost, collisions to root node (using **detect collisions**)
pushnode root

subfunction: use node with lowest cost and find constraints
Create new node.
If open list is nonzero:
Start iteration, with a maximum of 400000. **Pop lowest cost node**, becomes current node.
Return current node if amount of collisions = 0.

else:
Find constraints with **standard splitting** of current collisions.

subfunction: make two children and iterate further
For constraint in constraints:
create new node
copy constraints, agent and path of current node
if agent start = agent goal, and constraint location is goal, do not add constraint
else, add the new constraint
find new path with A* with new constraint
add new path, collisions (**detect collisions(new path)**) and cost to new node
push new node to open list

if iteration is out of bounds, then return current node

3.2.2. STANDARD SPLITTING(COLLISION)
if length location of collision is 1, implies a vertex constraint,
location of constraint 2 is location of constraint 1
if length location of collision is 2, implies an edge constraint,
location of constraint 2 is reverse of location of constraint
return the two constraints

3.2.3. DETECT COLLISIONS(PATHS)
for each path i in paths:
for each path j in paths later than current path i:
if path is not itself:
find location and time using **detect collision(path i, path j)**
return a list of all collisions in paths

3.2.4. DETECT COLLISION(PATH 1, PATH 2)
for time within 0 and shortest path:

if **get location(path, time)** of path 1 is equal to path 2,
return location (vertex), time

else if previous location of agent 1 is current location of agent 2, and other way around,
return location (edge), time

else: check for a goal constraint
loop over the longer path from arrival time of shorter path.
if location of longer path at time k is equal to goal of shorter path,
return location, time k

else if no collisions: return None, None

4
Distributed Planning

In Distributed Planning, the agents themselves create the shortest path. Most likely there are conflicts, which are solved by the agents, without the assistance of an orchestrating agent. An agent receives the paths of all agents within range, and checks whether there are conflicts. If there are conflicts, both agents create a new path. Which of the two agents use the new path depends on the total costs, which should be lowest.

4.1. Model Design

4.1.1. Path Planning

For the distributed planning model, the path planning is done in the same manner as the previously explained methods, by the A^* algorithm. Each agent plans its shortest path, and stores this.

4.1.2. Coordination

In order to resolve conflicts between agents, some design choices were made. When two agents are in range of each other, and there is a conflict, both agents propose a new path, with the conflict as a constraint. The decision for the chosen paths is based on one-to-one negotiation, in which the combination of paths with the lowest cost is chosen. This reflects the CBS algorithm, but then on a much smaller scale, allowing for larger scenarios to be simulated with acceptable computational costs. An issue with this approach could be that by choosing two paths in timestep t, other conflicts are created in a later stage, for which the conflict is still out of range. This could result in a higher total cost, and an increase in computation time, since some agents have to take a longer path due to choices in the past. The choice for one-to-one negotiation was made to increase simplicity, and minimize computation time, since there is only two paths are considered in the same time.

4.2. Model specification

4.2.1. Specification of the Environment

The environment is made of grey and white areas. The white area is accessible to the agents, and the grey area is not. There are three different maps, as shown in fig. 5.4. The environment is Deterministic, Static, and fully Accessible.

4.2.2. Specification of Agent Types and Characteristics

All agents are of the same type. They are proactive, create their own paths, check the surrounding, and resolve conflicts. The agent characteristics are the following:

1. At timestep 0, all agents are given a start location on the map, and have a goal location.
2. An agent is allowed to remain in its position, or move one step in any direction per timestep.
3. For all timesteps: if an agent is not yet at its goal, it will move towards the goal using the shortest feasible path.
4. An agent receives the location of surrounding agents which are up to 9 squares away.
5. If two agents occupy the same node or edge on the same timestep, there is a conflict.

6. If there is a conflict, both agents will create a new path, and the new paths with the lowest total cost are used.
7. All agents will not move to a grey area of the environment

Cognitive and behavioral properties The agents create the shortest path, see other agent paths, and change the path if there is a conflict. The agents observe and take action if required.

4.2.3. SPECIFICATION OF INTERACTION
Agents interact with each other. If a conflict is detected, both agents cooperate to find the two paths with the lowest cost. The interactions are sequential, and are initiated by the agent which is first in the loop.

4.3. SEMI-FORMAL REPRESENTATION OF THE DISTRIBUTED PLANNER
In this section, the Distributed planner is presented in a semi-formal manner. It will be split up in the two python files, and their respective functions. Distributed planner serves as an iterator, storing the results for each aircraft. It also calls the aircraft function, for each agent, and provides the agent with information on all other agents.

4.3.1. DISTRIBUTED PLANNER
subfunction: initialization Store time data
create agent object for amount of agents
find path for each agent

update the maximum time

subfunction: find new paths for each agent, and keep iterating while time is lower than maximum time
while iterations are within bounds and there are collisions
for each agent:
find its radar
find a new result without collisions using **Radar** from aircraft.py
if there still are collisions due to new solution
iterate again

else:
go to next timestep

4.3.2. AIRCRAFT
object is an agent

subfunction: find out if all constraints are still required for each constraint of this agent
if the constraint is in the next two timesteps
find and copy the other agent (agent 2) in the collision

copying the constraints of the original agent
remove the current (singular) constraint

find a new path for the original agent
is it a valid path, replace the new agent's path
if no collision occurs between the agent with the new path, and the second agent, the constraint is be permanently removed

subfunction: find out which agents collide, and create constraint for each agent given in the radar:
if the distance, using **calcdistance**, between the agent object, and the other agent is too large, continue to next path
else:
find the collisions between the two agents within radar range

if there are collisions, and the iteration remains within bounds:
create a constraint for both agents

subfunction: find new paths for both agents find a new path for the object agent, starting from its current location, or its goal, depending on the time, to the goal, with the new constraint. The solution is checked to be nonzero

do the same for the second agent, and also check if there is a solution

subfunction: which solution to keep find a new radar with both new paths, and find collisions using **detect collision**

if there is no collision, there is no **detect stalemate**, and the solution with two new paths is shorter than keeping one of the original paths, the solution is kept

else if using the new path of the object agent is shorter than the new path of the second agent, and there is no **detect stalemate**, keep the new path of the object agent

else if the second constraint is nonzero, and there is no **detect stalemate**, changing the path of the second agent must be more optimal, thus this solution is kept

else if the first constraint is nonzero, the object agents' new path is kept

update the radar, constraints and the collisions

detect stalemate
find out if two agents are waiting on each other and both not moving if both agents did not move past timestep, return True, else return False

returnradar
for each agent, find the path between current time and time plus the radar
return a list of all agents' radar paths

5
Model Performance and Comparison

This chapter describes the analysis of the performance of the models, and a comparison between all models.

5.1. Performance Indicators

To assess the performance of all models, some performance indicators were defined.

- **Computation time:** The time the model takes give a solution.
- **Accuracy:** The model should give a solution which should be optimal, or very close.
- **Safety:** The model should not have any collisions.
- **Robustness:** The ability of the model to succeed with different maps, inputs, and outputs.
- **Scalability:** The ability of the model to succeed with a larger number of agents, more in/outputs, and more interactions.

5.2. Comparison of Model Performance

In this section, the models will be compared with the above described indicators. The numerical results for computation costs and path length were generated using similar circumstances on the computer used (e.g. power plugged in, comparable amount of other workload), while directly compared results per map and planner were always done in direct succession to each other. This is done to influence results the least and be able to compare the numerical values. Additionally, the maps used are exactly the same for each planner, for the same goals.
All three methods were deemed "safe", as they were all programmed such that no collisions can occur if there is a viable solution.
Note that the before mentioned radar of the distributed planner is fixed at a length of 9. This radar length worked well in the test cases, and a good middle ground between looking far ahead and not to far for the main layouts.

5.2.1. Specification of the Environment

Three test cases have been used in comparing the three different planning algorithms. First of all, there is a large sample size, random small grid environment, the "test" files already given in the assignment, and the three main layouts.
To compare the performance on smaller grids, the standard test form of an 8x8 grid was chosen. To complicate the model, random starts, goals and obstacles were placed on the map, with the starts and goals being connected. The number of agents were varied slightly, with 6-8 agents, making the map relatively complicated for such a small map. a sample size of 200 maps was used in comparing the results, each with 6 obstacles.
The test files which were provided, are very similar to these randomly generated maps, while the main layouts are much larger. They are shown in fig. 5.4, and will be referenced as layout 1, 2 and 3 respectively in numerical results.

5.2.2. Performance Comparison on Small Randomized Layouts

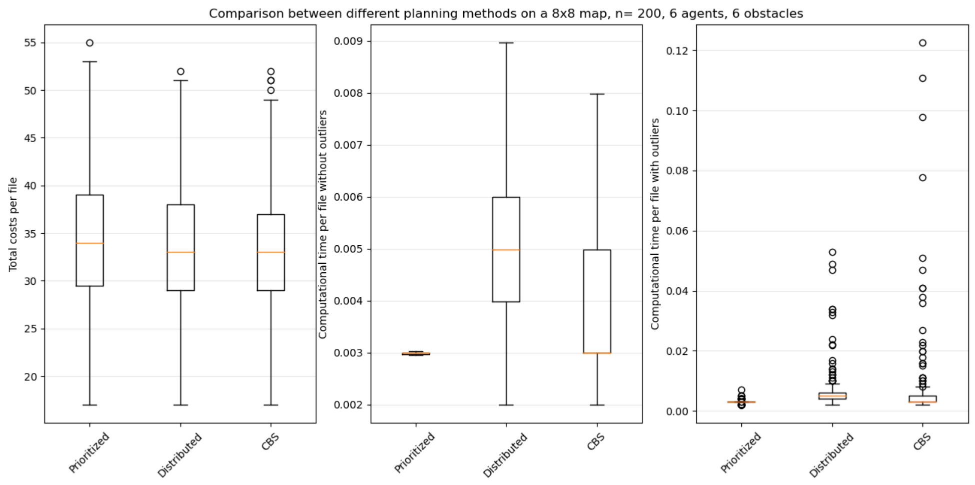

Figure 5.1

Table 5.1: A comparison of average results of the three planners

	Prioritized	Distributed	CBS
Average path length [-]	34.32	33.55	32.97
Total computational time [s]	0.5954	1.360	1.511
Average time per map [s]	0.002992	0.006835	0.007598

In fig. 5.1, the box plots for the three planners in the following scenario are given: 200 random maps (the same maps for each planner), each with 6 agents and 6 obstacles. It can be noted that the prioritized planner often has the lowest computational time, with especially the implemented distributed planner having a large variance. However, the CBS algorithm has the highest outliers when looking at computational time. The box-plot for the total cost, or path length, shows that the prioritized is less optimized than the other two planners, with CBS having the lowest cost. This is expected, as CBS always gives the optimal result. Furthermore, all code has been written with an amount of print statements reduced to a minimum, and none in loops, as these drastically increase computation times.

Table 5.1 gives an overview of the average path length, total computational time, and average time per map, for the scenario elaborated upon above. This data will eventually lead to the same conclusions as the box plots; CBS is optimal, but relatively slow, the distributed planner is somewhat quicker on average, but does not find an optimal result, while the prioritized planner is computationally less expensive, but lacks accuracy.

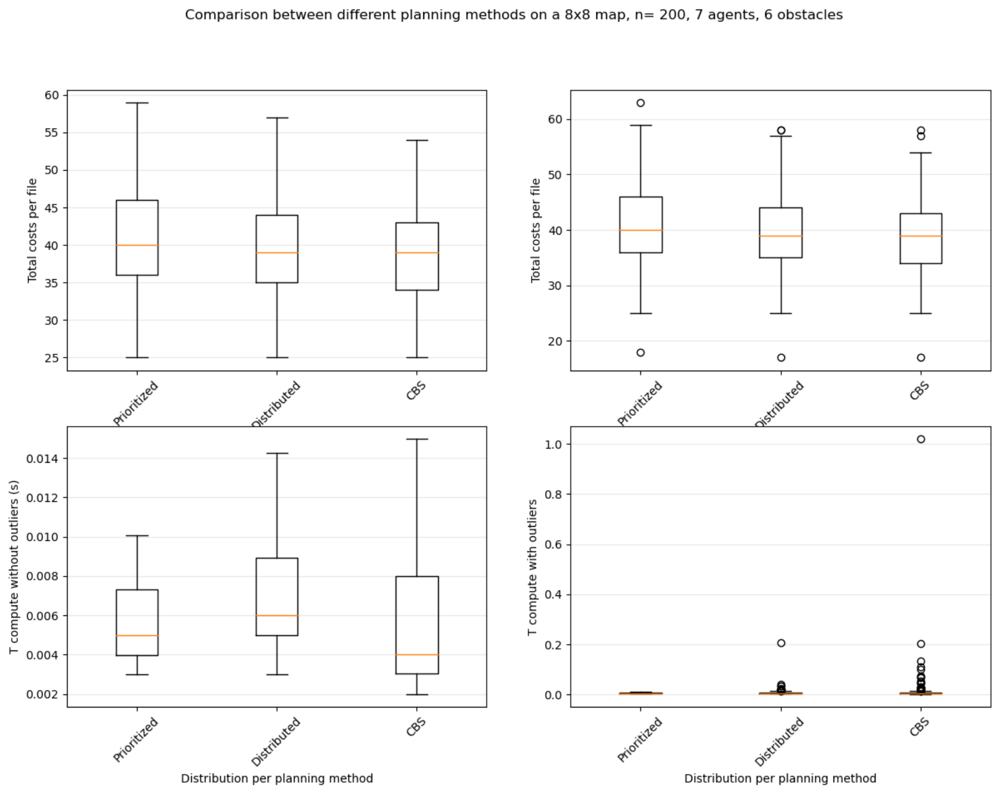

Figure 5.2

Table 5.2

	Prioritized	Distributed	CBS
Average path length [-]	40.82	39.86	38.76
Total computational time [s]	1.106	1.745	3.141
Average time per map [s]	0.005615	0.008858	0.01594

Table 5.2 and fig. 5.2 show very similar results as the previous simulation with 6 agents. In this case 7 agents were used. CBS clearly has more difficulty with obtaining the optimal result, leading in quite a larger computational time, also relative to the other two planners. The outliers especially increase, while the mean computation time stays low. This is likely due to the small size of the maps, with a few maps being more complicated and thus more difficult for this algorithm.

If another agent is added, the trends that occur with respect to computation time and only increase, while robustness drastically decreases. Especially the prioritized planner had a lot of difficulties with providing a solution to more complicated scenarios. The numerical results are not reliable to base any conclusions on, as the scenarios that were run were less complicated. Therefore, the box plot is only shown in the appendix. The likely culprit of the robustness issues is, that an agent with a higher priority blocks the path of an agent with a lower priority, causing no solution to be found. In these tests the robustness of CBS and the distributed was satisfactory, and did not present any problems running these tests.

5.2.3. PERFORMANCE COMPARISON ON TEST LAYOUTS

In fig. 5.3 the box plot for a simulation on all test files is given, again for the three evaluated planners. These results are relatively similar to the results in the 200-batch randomized files. Again, robustness issues were present with the prioritized planner, not being able to run test 25. This file was omitted from the results. The total path cost for the three planners is 1922 for Prioritized, 1866 for distributed and 1810 for CBS, the optimum. The robustness of CBS and the distributed was again not a problem.

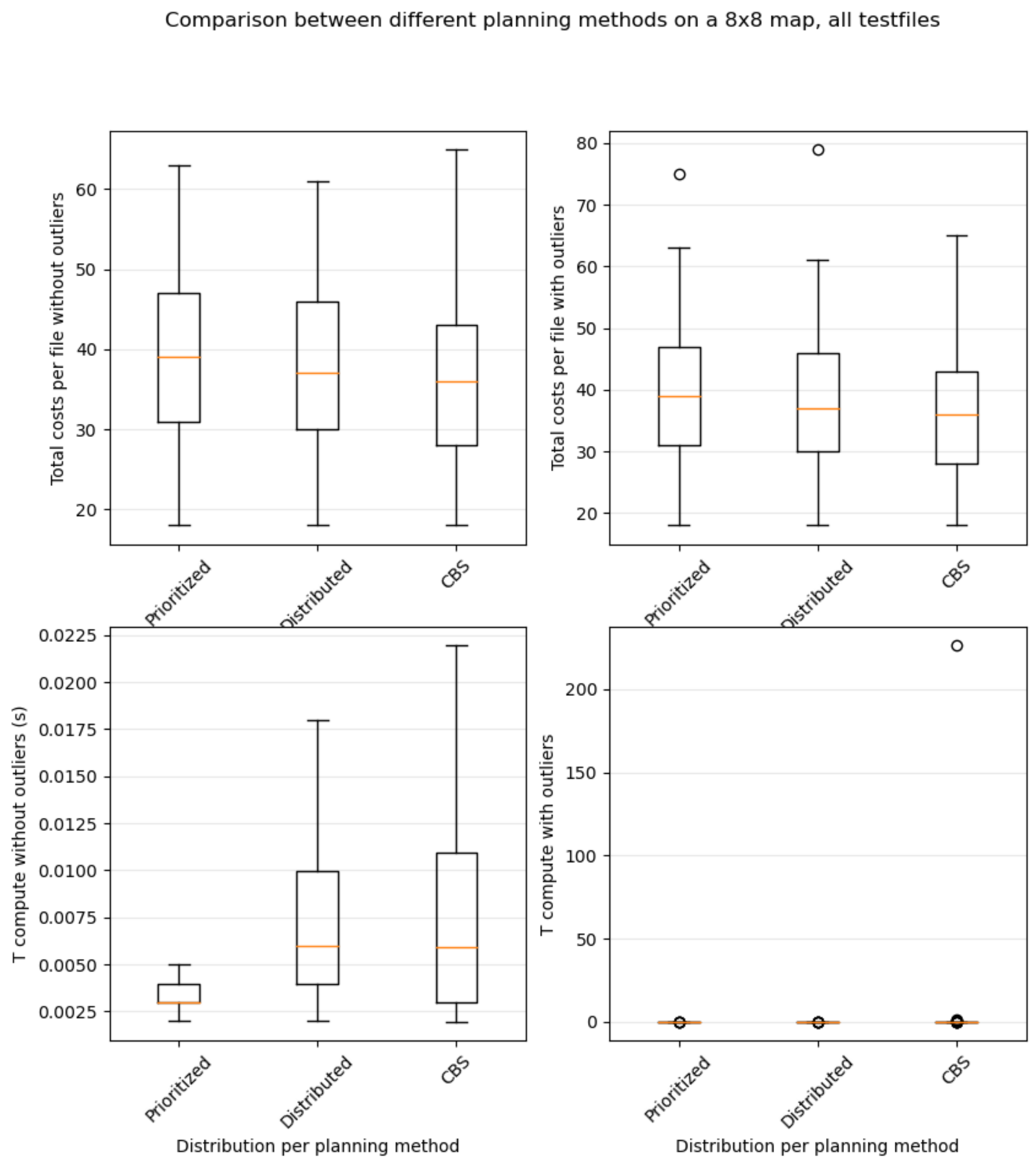

Figure 5.3

5.3. PERFORMANCE COMPARISON ON STANDARD LAYOUTS

To assess the performance of the models, three standard maps were used. The results of the first and last map are explained in this section. Results from the second map gave in-between results, and are shown in appendix A. The three standard maps are shown in fig. 5.4.

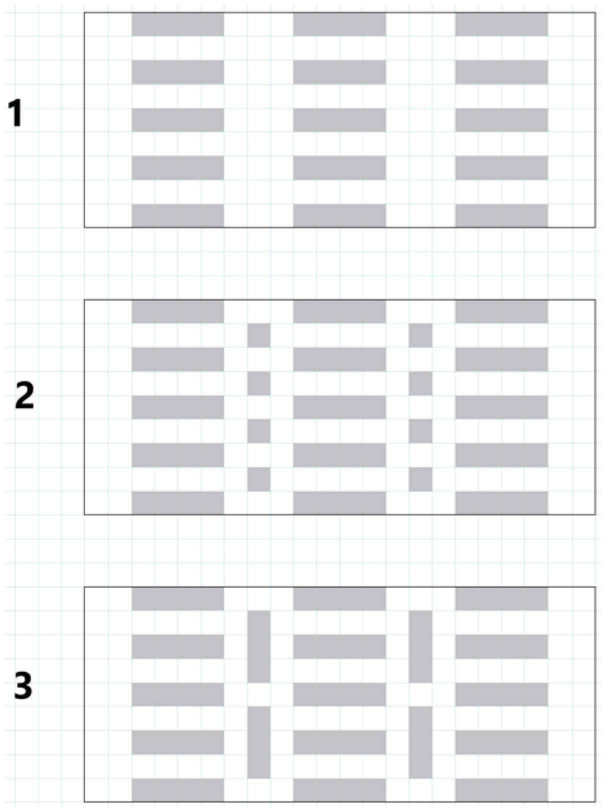

Figure 5.4: The main layouts used by the agents.

5.3.1. COMPARISON OF TOTAL COST

In this section, the performance of the models using the layouts presented in fig. 5.4 is analyzed. Since CBS is designed to find the optimal solution, the accuracy of the other models was measured with respect to the CBS results. Figure 5.5 shows the increase in total cost of the agent movements with respect to the CBS model results, with increasing number of agents. The Distributed model performed better in this simulation, with a lower total cost than the Prioritized model. The sharp increase implies that for numbers of agents larger than 12, this might not be true anymore. This was not possible to simulate, due to the required computation time for the CBS algorithm. This is recommended for future research.

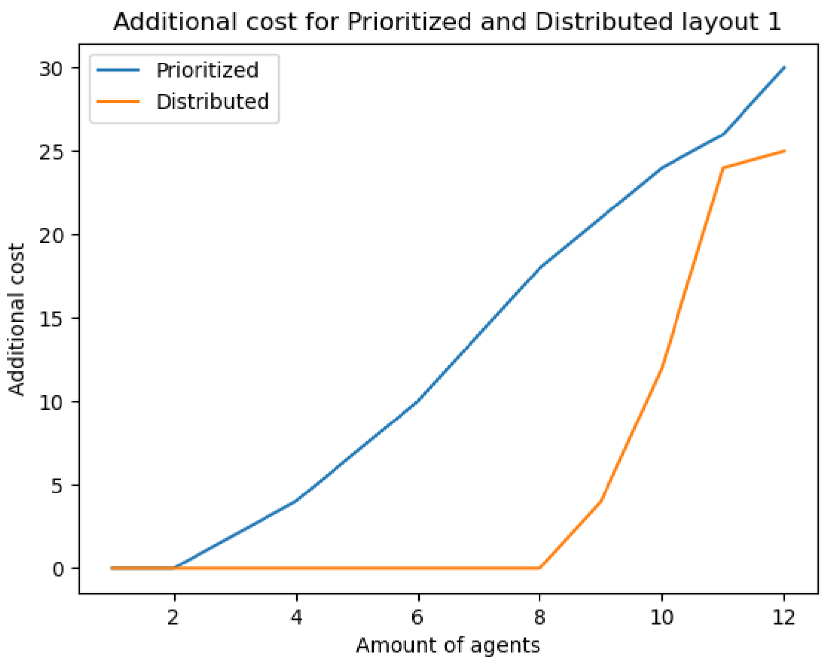

Figure 5.5: Additional cost of the simulation of the prioritized and distributed models with respect to CBS using layout 1.

Figure 5.6 shows the additional cost of the two models used with layout 3. The results show similarities. The the Distributed model performs a better in these tests with a lower total cost. Increasing the number of obstacles increases the error for the prioritized model slightly. The distributed model has a discrepancy with a smaller number of agents, but performs about the same with 9 agents.

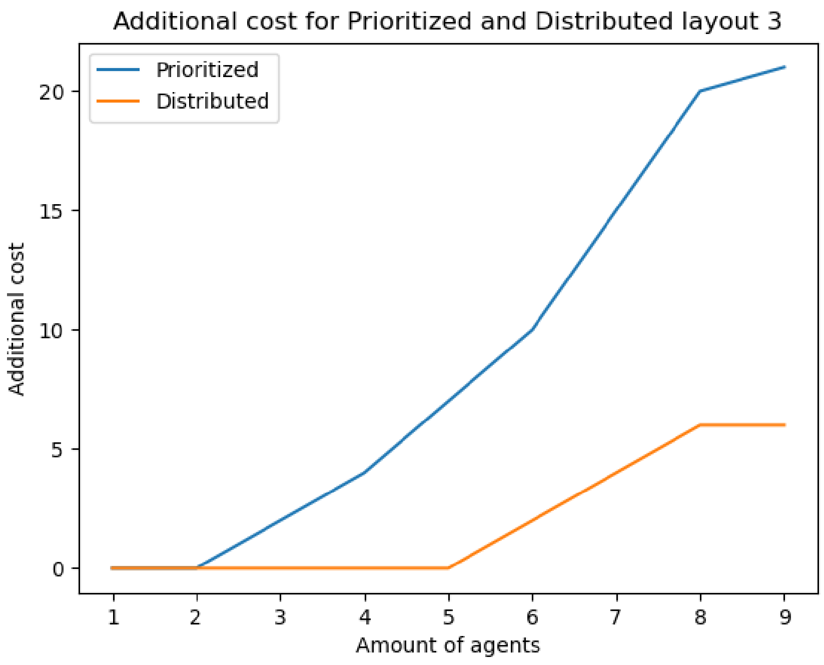

Figure 5.6: Additional cost of the simulation of the prioritized and distributed models with respect to CBS using layout 3.

5.3.2. Comparison of Simulation Time

Figure 5.7 shows the computation time of the models for an increasing number of agents using layout 1. Up to 9 agents, all models give fast results, and perform equally well. When the number of agents increases, especially CBS becomes much more time consuming. The CBS computation time with 12 agents, not visualized in this figure for clarity, was almost 30 minutes. Prioritized performs a bit better than distributed.

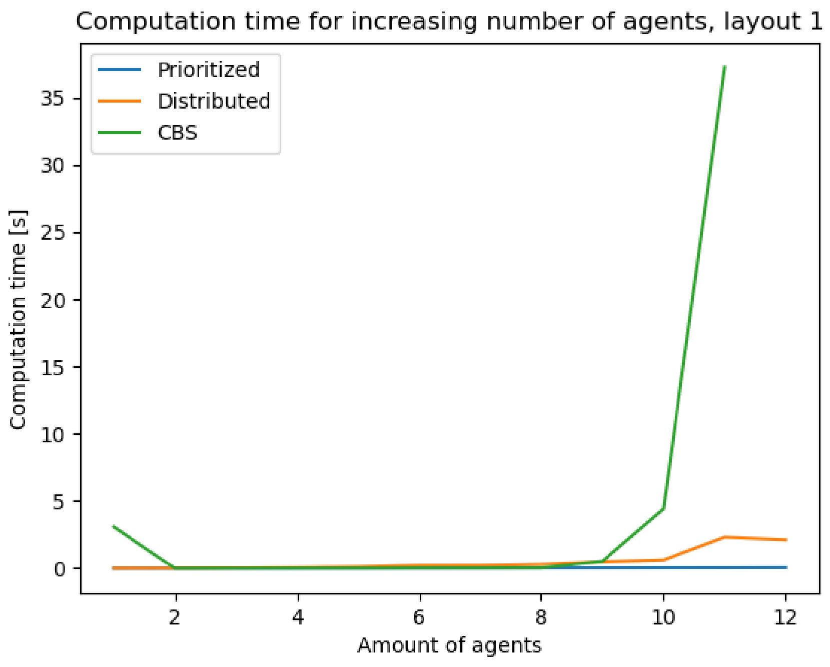

Figure 5.7: Total simulation time for increasing number of agents for layout 1.

The same test was run with layout 3. The results are shown in fig. 5.8. Computation time was shorter, but the number of agents was smaller as well. CBS showed an off the chart rise in computation time for 9 agents.

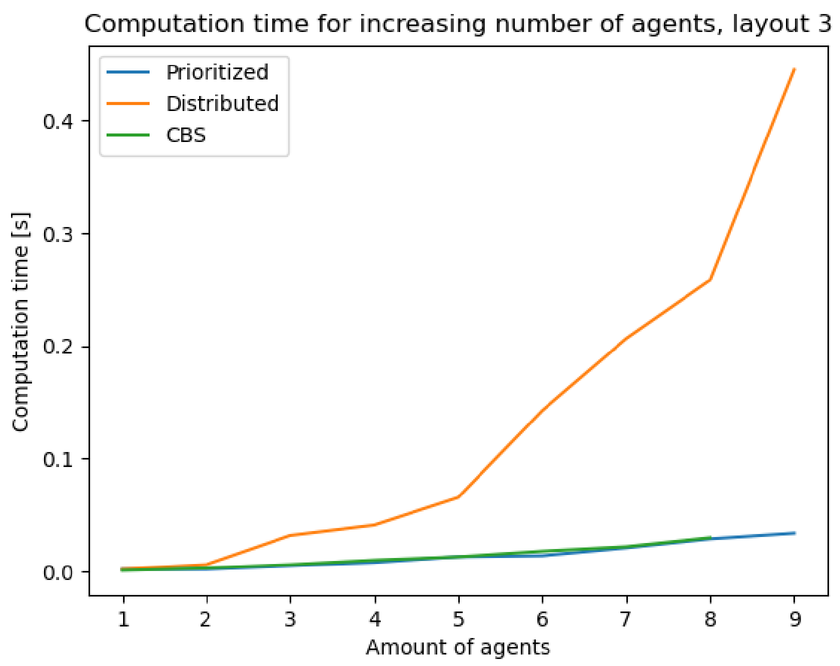

Figure 5.8: Total simulation time for increasing number of agents for layout 3.

Table 5.3 and table 5.4 show the simulation results. These can not be compared directly, since the simulation for layout 1 was run for up to 12 agents, and layout 3 up to 9 agents. What can be concluded is that in general, the Distributed model is more accurate than the prioritized model. The relative performance of the distributed model was better with layout 3 than layout 1. For the prioritized model this was the other way around, so the performance is better with layout 1. The overall robustness of each planner was not of any problem during the test runs of the standard layouts with limited amounts of agents.

Table 5.3: Path length and simulation time for layout 1

	Prioritized	Distributed	CBS
Average path length [-]	183.083	175.5	170.083
Total computational time [s]	0.31167	6.5395	1747.94
Average time per map [s]	0.025972	0.54496	145.661

Table 5.4: Path length and simulation time for layout 3

	Prioritized	Distributed	CBS
Average path length [-]	153.33	145.22	144.55
Total computational time [s]	0.1237	1.857	24.28
Average time per map [s]	0.01374	0.2063	2.70

5.4. Recommendations for the Distributed Planner

In the previous sections, the results from the smaller test cases point towards the distributed planner being in the middle ground between the prioritized and CBS algorithms, both in terms of computation time and accuracy. In larger cases with a high agent density, the distributed planner has a similar accuracy as the prioritized. This can also be observed when running the layouts with many agents and observing their paths. It can be concluded that the part of the algorithm in which the agents decide their next move, needs tweaking. It is currently rule based (most optimal solution wins), but in future developments it would be recommended to investigate other algorithms, such as negotiations, and rules concerning the specific map for example. Furthermore, the removal of constraints if they are no longer necessary should be investigated, if it functions optimally, or can be improved. These improvements will provide for a more optimal result, while also reducing iterations, thus decreasing computation time. Another area in which further advancements can be made is the radar. A larger radar would be very intensive computationally, but too small of a radar and the accuracy of the planner decreases, especially in complicated maps with many agents. For example, a variable radar with respect to the amount of agents, map size, and agent location could be implemented and evaluated to lower computation costs, while maintaining accuracy.

6 Conclusion

In this report, three different agent based models for path finding were explained. When looking for accuracy, the Conflict Based Search (CBS) model is the best choice according to the simulation results. This was in line with expectations, since it is an algorithm which is designed to find the optimal solution. The results of the Distributed model are on average more accurate than the results of the Prioritized model, especially with a larger number of agents. This was as expected as well, since the the lowest priority agents have many constraints, and may have to wait a long time, or have to go the long way around.
When looking into the numerical performance indicators, the following conclusions were drawn:

- With a small amount of agents, and a small map size, all three planners perform similarly with respect to computation time, while CBS is the most accurate.
- With more agents, and more challenging maps, both the prioritized and distributed algorithms are significantly faster than the CBS algorithm, while distributed is slower than prioritized.
- The CBS algorithm has more extreme outliers, making it less suitable for applications which require consistent computational speed.
- The CBS algorithm is not suitable for large maps with many agents, as it is computationally very intensive, thus the scalability is low.
- The robustness of the prioritized planner is not up to par with the other planners, and can cause issues with some test cases.
- The distributed planner is computationally less intensive than CBS, while being more accurate than prioritized in simpler but larger scenarios.
- The distributed planner is theoretically able to be a middle ground with respect to computation time and accuracy between CBS and prioritized. However, the accuracy for large maps with an high agent density, and thus scalability of this implementation is not as high as expected.

When looking at the distributed model, this implementation is not perfect yet. Improvements could be made in the negotiation between agents, the removal of constraints which are no longer applicable, and taking into account more agents and paths. This could both improve model accuracy and lower computation time if implemented correctly. Then, the distributed planner can be a powerful tool in planning large scale agent-based planners at relatively low computational cost, and high accuracy.

A
Results Layout 2

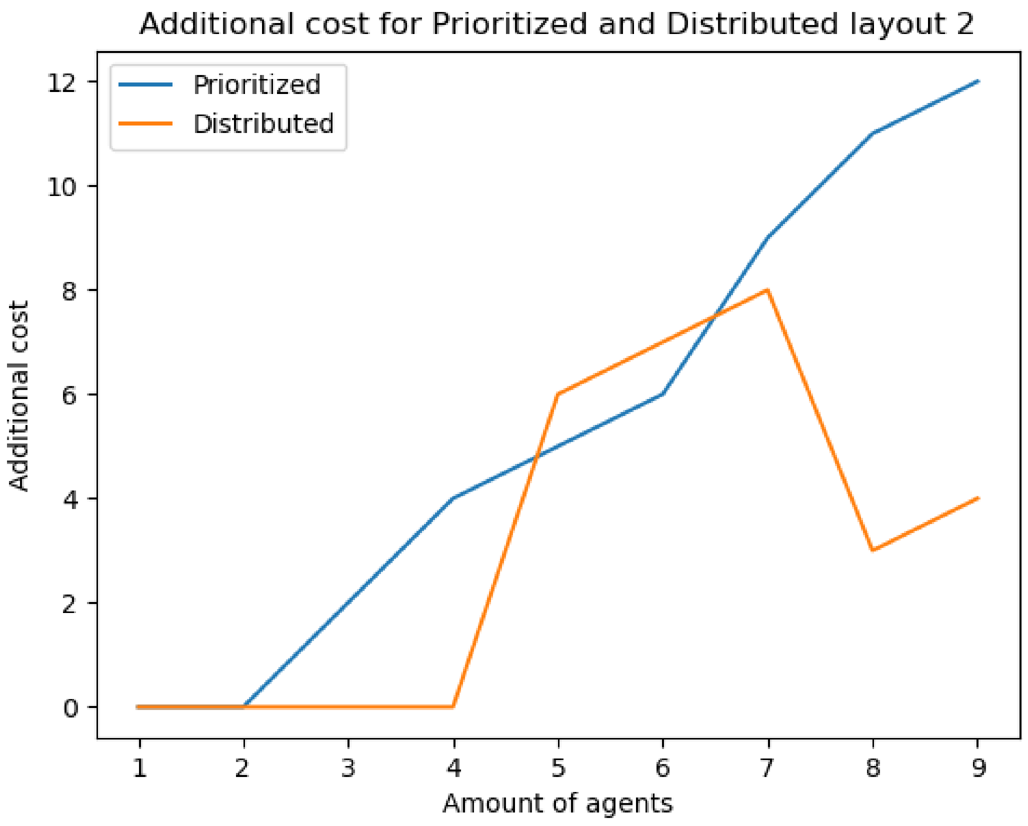

Figure A.1: Additional cost of prioritized and distributed models with respect to CBS using layout 2.

B
Batch Results Test Files

```
test 1.txt,  [path, waiting time]: [40, 1], collisions: 0, cost wrt optimal: 0
test 10.txt,[path, waiting time]: [19, 0], collisions: 0, cost wrt optimal: 0
test 11.txt,[path, waiting time]: [35, 0], collisions: 0, cost wrt optimal: 0
test 12.txt,[path, waiting time]: [36, 0], collisions: 0, cost wrt optimal: 0
test 13.txt,[path, waiting time]: [35, 2], collisions: 0, cost wrt optimal: 1
test 14.txt,[path, waiting time]: [24, 0], collisions: 0, cost wrt optimal: 0
test 15.txt,[path, waiting time]: [51, 1], collisions: 0, cost wrt optimal: 2
test 16.txt,[path, waiting time]: [55, 0], collisions: 0, cost wrt optimal: 4
test 17.txt,[path, waiting time]: [39, 0], collisions: 0, cost wrt optimal: 0
test 18.txt,[path, waiting time]: [34, 0], collisions: 0, cost wrt optimal: 2
test 19.txt,[path, waiting time]: [47, 0], collisions: 0, cost wrt optimal: 0
test 2.txt,  [path, waiting time]: [18, 0], collisions: 0, cost wrt optimal: 0
test 20.txt,[path, waiting time]: [28, 0], collisions: 0, cost wrt optimal: 0
test 21.txt,[path, waiting time]: [46, 0], collisions: 0, cost wrt optimal: 0
test 22.txt,[path, waiting time]: [53, 0], collisions: 0, cost wrt optimal: 2
test 23.txt,[path, waiting time]: [32, 0], collisions: 0, cost wrt optimal: 0
test 24.txt,[path, waiting time]: [47, 1], collisions: 0, cost wrt optimal: 1
test 26.txt,[path, waiting time]: [46, 7], collisions: 0, cost wrt optimal: 11
test 27.txt,[path, waiting time]: [39, 1], collisions: 0, cost wrt optimal: 0
test 28.txt,[path, waiting time]: [39, 2], collisions: 0, cost wrt optimal: 0
test 29.txt,[path, waiting time]: [48, 0], collisions: 0, cost wrt optimal: 0
test 3.txt,  [path, waiting time]: [28, 0], collisions: 0, cost wrt optimal: 0
test 30.txt,[path, waiting time]: [42, 1], collisions: 0, cost wrt optimal: 0
test 31.txt,[path, waiting time]: [39, 1], collisions: 0, cost wrt optimal: 1
test 32.txt,[path, waiting time]: [28, 3], collisions: 0, cost wrt optimal: 1
test 33.txt,[path, waiting time]: [29, 2], collisions: 0, cost wrt optimal: 3
test 34.txt,[path, waiting time]: [33, 0], collisions: 0, cost wrt optimal: 0
test 35.txt,[path, waiting time]: [30, 0], collisions: 0, cost wrt optimal: 0
test 36.txt,[path, waiting time]: [23, 0], collisions: 0, cost wrt optimal: 0
test 37.txt,[path, waiting time]: [37, 1], collisions: 0, cost wrt optimal: 0
test 38.txt,[path, waiting time]: [28, 0], collisions: 0, cost wrt optimal: 0
test 39.txt,[path, waiting time]: [35, 0], collisions: 0, cost wrt optimal: 0
test 4.txt,  [path, waiting time]: [32, 0], collisions: 0, cost wrt optimal: 0
test 40.txt,[path, waiting time]: [24, 0], collisions: 0, cost wrt optimal: 0
test 41.txt,[path, waiting time]: [46, 4], collisions: 0, cost wrt optimal: 5
test 42.txt,[path, waiting time]: [57, 0], collisions: 0, cost wrt optimal: 0
test 43.txt,[path, waiting time]: [43, 3], collisions: 0, cost wrt optimal: 3
test 44.txt,[path, waiting time]: [32, 1], collisions: 0, cost wrt optimal: 0
test 45.txt,[path, waiting time]: [24, 0], collisions: 0, cost wrt optimal: 0
test 46.txt,[path, waiting time]: [60, 1], collisions: 0, cost wrt optimal: 4
test 47.txt,[path, waiting time]: [69, 10] collisions: ,0, cost wrt optimal: 14
test 48.txt,[path, waiting time]: [38, 0], collisions: 0, cost wrt optimal: 2
test 49.txt,[path, waiting time]: [40, 2], collisions: 0, cost wrt optimal: 0
test 5.txt,  [path, waiting time]: [25, 1], collisions: 0, cost wrt optimal: 0
test 50.txt,[path, waiting time]: [43, 5], collisions: 0, cost wrt optimal: 0
test 6.txt,  [path, waiting time]: [24, 0], collisions: 0, cost wrt optimal: 0
test 7.txt,  [path, waiting time]: [34, 0], collisions: 0, cost wrt optimal: 0
test 8.txt,  [path, waiting time]: [37, 1], collisions: 0, cost wrt optimal: 0
test 9.txt,  [path, waiting time]: [24, 0], collisions: 0, cost wrt optimal: 0
total cost: 1866
runtime: 0.6095340251922607
```