Delft University of Technology

Agent based Modelling and Simulation in Air Transport
AE4422-20

Multi-Agent Path Planning and Coordination of Airport Surface Movement

Authors:
Maarten Hogenkamp, 5852935
Pieter Becking, 4685377

November 20, 2022

Contents

1 Introduction

Airports worldwide are struggling to improve their airport operations in order to cope with the high demand of modern air travel. This requires safe and efficient responses, especially for busy hub airports. Agent-based modeling techniques can be used to simulate global behavior and system performances, which may be precious in identifying flaws in systems upfront before investments are made. The only investment is the computational effort for a model.

This research analyses and compares three different planning methods: Prioritized Planning, Conflict-Based-Search (CBS) Planning, and Distributed Planning. The methods have been modeled by using a base model delivered by the supervisors of this project. Key performance indicators were established for the analysis and comparison of the models. The three algorithms are compared and analyzed in Chapter 6, and a sensitivity analysis is done in Chapter 7. The report ends with a conclusion and a discussion in the final chapter.

2 Assignment Description

The objective of this assignment is to explore and compare three multiagent path planning algorithms. The process of implementing, analyzing and comparing the algorithms is described in this report. In this process, real-world situations and metrics have been taken into account to obtain results being as relevant as possible.

2.1 Research Question

How do Prioritized planning, Conflict-Based-Search (CBS), and Distributed planning compare when it comes to planning and coordinating taxiing aircraft?

2.2 Model Structure

This section describes the setup and global directives of the assignment.

2.2.1 Environment

Three layouts of airports are being used to test the performance of the algorithms. Every cell in the environment is assigned to be either an obstacle or a free space, represented in grey and white respectively. Fig. 1 shows the three possible layouts with increasing obstacles. Start and goal locations of agents are placed in the free spaces of the environment in a random or semi-random manner (see Section 6.1.3). Each start and goal location has a unique cell so that a cell cannot be shared by either.

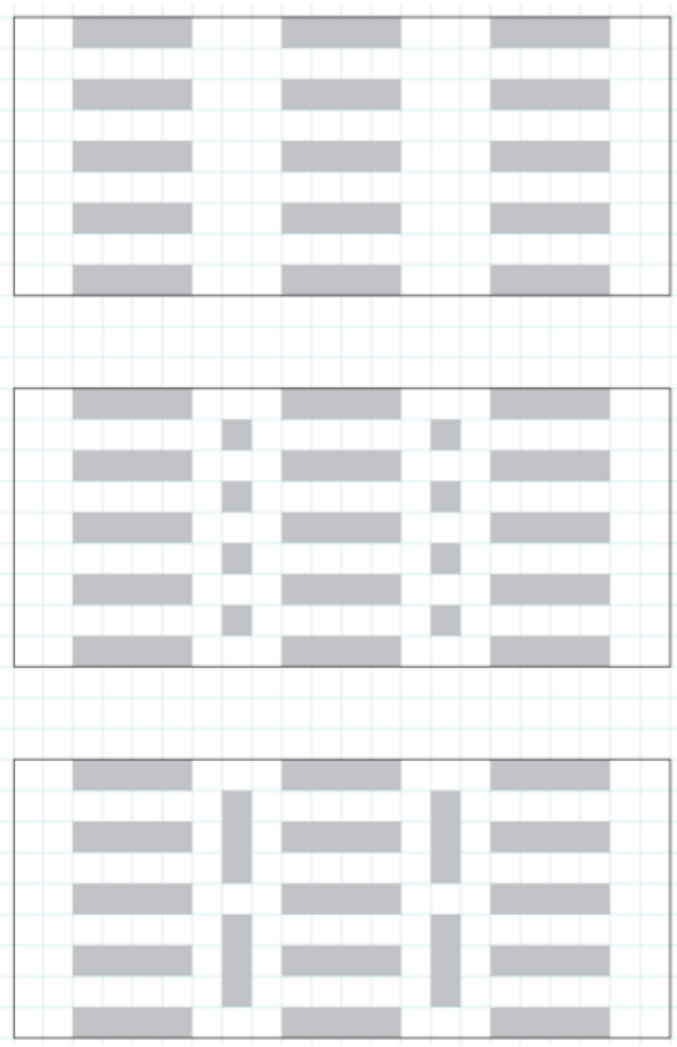


Figure 1: Three layouts for the environment

2.2.2 Agents

The taxiing aircraft are represented by agents. At t=0, each agent is at its start position. The agents are allowed to either wait at their location or move one cell per time step in the north, south, east, or west direction. The agents must not collide with the objects in the environment, nor with each other. Agents swapping locations would also result in a collision. The goal of each agent is to reach its goal location. The agent will stay in the environment until all agents reached their goal locations.

2.2.3 Python Files

The following structure for the python files is used:

- *main.ipynb* is the main file to generate and run sets of experiments, save the data and plot the results.
- *run_experiments.py* runs the experiments according to the input from *main* file and returns the result.
- *plotting.ipynb* includes some of the functionality to create plots
- Additional files are used for the individual algorithms, these, however, correspond to the files as introduced in the tutorial and are therefore not detailed here

3 Prioritized Planning

In Prioritized Planning, an order of priority is used to find a solution. The agent with the highest priority will follow the path resulting from the single agent planner without constraints, i.e. the shortest path. Agents next in line use the paths of the agents of higher priority as constraints to compute their shortest path without collisions.
This method is relatively effortless in terms of computing power. However, as the constraints accumulate for lower-ranked agents, the algorithm may not find solutions to straightforward situations.

3.1 Local properties of agents

At starting time t=0, all agents are allocated an identification number, ranging from 0 for the first agent to $n-1$ for the n'th agent, based on the order of creating the agents. The agents use A* to compute their shortest path while avoiding constraints and obstacles. The agents have the following local properties:

- **Environment**: The environment is a field of cells, each cell being a free space or obstacle. The agents can only be in free spaces. In each time step, agents can move no cell or move one cell up, down, left or right.
- **Initialization**: The start and goal locations are initialized on the map in different ways as detailed in section 6.
- **LP.1** (behavioral) Each agent plans its shortest path at the initial time step, but after their preceding agent has planned its path.
- **LP.2** (cognitive) Each agent is aware of the paths planned by its preceding agents.
- **LP.3** (behavioral) Each agent treats the paths of its preceding agents as time-dependent obstacles while planning its own path.
- **LP.4** (behavior) If an agent reaches its goal location, it becomes a permanent obstacle for subsequent agents as it stays there until the experiment has ended.

3.2 Implementation in Python

Algorithm 1 shows the pseudo-code of the Prioritized planner. To calculate the paths, it uses the A*-algorithm which is imported from `single_agent_planner.py`. The A*-algorithm required some modifications as well to deal with constraints at different time steps. To deal with goal constraints, an agent which has reached the goal location turns into a permanent obstacle and hence, is treated as such. For this purpose, the *heuristics* and *map_data* is modified to represent a goal as an obstacle.

Algorithm 1 Prioritized Planning

Inputs: map, start locations, goal locations, heuristics
Output: collision-free paths for all agents
result_paths ← *empty*
for *agent* in *agents* **do**
  for *path* in *result_paths* **do**
    agent_constraints ← create constraints arising for *agent* due to already planned *path*
  end for
  path of *agent* = **a_star**(map, agent information, *agent_constraints*)
  if *path* is none **then**
    return **FAILURE**
  end if
end for
result_paths = *result_paths* + *path* of *agent*
return paths for all agents

4 CBS Planning

In Conflict-Based Search Planning, all agents initially plan their paths with the single-agent planner without constraints. The CBS algorithm detects the *conflicts* caused by colliding paths. The High-Level Search searches the collisions and chooses a collision to be transformed into two constraints (one for each agent).
Planning with CBS results in optimal planning, but it comes at a cost of increased computational expense.

4.1 Local properties of agents

- **Environment**: The environment is a field of cells, each cell being a free space or obstacle. The agents can only be in free spaces. In each time step, agents can move no cell or move one cell up, down, left or right.
- **Initialization**: The start and goal locations are initialized on the map in different ways as detailed in section 6.
- **LP.1** (behavioral) When an agent plans its route it does so without knowing if it will collide on its route, therefore it may also re-plan
- **LP.2** (behavioral) If an agent has reached its goal location, it stays there and becomes a permanent obstacle for other agents until the experiment has ended
- **LP.3** (cognitive) An agent can obtain a constraint for its next planning iteration when a collision is detected

4.2 Implementation of the Model

Algorithm 2 shows the pseudo-code of the CBS planner. It was implemented in python in a very similar way. Note that some functions such as **standard_splitting** and **detect_collisions** were additionally implemented, these are not detailed here, however. The reader is referred to the python code. As previously in section 3, the A*-algorithm is imported from `single_agent_planner.py`.

Algorithm 2 Planning using CBS

Inputs: map, start locations, goal locations, heuristics
Output: collision-free paths for all agents
open_list ← *empty*
initial_paths ← independent paths planned using A* with no constraints
initial_collisions = **detect_collisions**(*initial_paths*)
first_node ← *initial_paths*, corresponding cost and *initial_collisions*
push *first_node* into *open_list*
while *open_list* is not empty **do**
    curr_node ← pop node from *open_list* with lowest cost
    if no collisions in *curr_node* **then**
        return paths for all agents
    else
        collision ← random collision from *curr_node*['collisions']
        constraints = **standard_splitting**(*collision*)
        for *constraint* in *constraints* **do**
            queue=*curr_node*
            constraints of *queue* = constraints of *queue* + *constraint*
            agent_id=agent for which *constraint* is active
            path of *agent_id* = **a_star**(map, agent information, constraints in *queue*)
            if path is none **then**
                continue
            else
                queue['paths']['agent_id']=path
                queue['collisions']=**detect_collisions**(*queue*['paths'])
                queue['costs']=**get_sum_of_costs**(*queue*['paths'])
                push *queue* into *open_list*
            end if
        end for
    end if
end while
return **FAILURE**

5 Distributed Planning

In the developed distributed planning model, agents try to individually plan their paths by considering other agents in their area and their corresponding paths. Specifically, an approach is chosen, where agents that are within each other's *radar* enter a negotiation if their plans collide and resolve any collisions in that plan. The negotiation resembles the CBS algorithm as introduced previously on a local scale (i.e., within the agent's radars). However, instead of re-planning right away, agents try to wait for the agent in conflict to reduce additional fuel spent due to diversions. Negotiations are resolved by determining a winning agent. In this chapter, details of the different aspects of the distributed planning algorithm are described.

5.1 Local properties of agents

For the distributed planning algorithms, the concepts of path and plan are introduced as two different notions and therefore a strict differentiation must be made. Specifically, the distinction is explained as follows:

- A **plan** of an agent is a list of positions on the map that corresponds to the intended next nodes. It is not necessary that the agent will act upon this plan and it might change as part of the negotiations
- A **path** of an agent is a list of positions from the start location to the goal location. Considering an agent at time t, the plan consists of a deterministic part that describes the nodes already visited up to time t as well as a part from $t+1$ to t_{end} which corresponds to the plan (i.e., the next intended nodes)

The agents in the distributed planning algorithm try to find paths using the A* algorithm as discussed in section 3. They are equipped with a radar that scans the environment around them and identifies other agents. If agents are in each other's field of view, they will enter a negotiation where conflict-free plans of all agents are sought after. The negotiation is based on the local properties that the agents have, which are the following:

- **Static environment**: The static environment with its obstacles is known to all agents in its entirety. This means that also obstacles outside of the agent's radar distance are known to the agent. This is motivated by the fact that good map data is usually available for aircraft.
- **Dynamic environment**: The dynamic environment consists of other agents which also live on the map. An agent can only see other agents up to its scanning distance with a certain update rate (scanning frequency). If an agent sees another agent within its field of view, it obtains its full plan. Hence, in the negotiation, every agent knows the other agent's full intentions.
- **Initialization**: The start and goal locations are initialized on the map in different ways as detailed in section 6.
- **LP.1** (cognitive) At each time step, an agent updates its paths by re-planning depending on the dynamic conditions
- **LP.2** (behavioral) When an agent reaches its goal location it stays there until all agents have reached their goal locations by becoming a permanent obstacle for other agents
- **LP.3** (cognitive) If an agent sees other agents on its radar, it enters a negotiation with all agents on its radar as well as on the radars of other agents in the negotiation
- **LP.4** (behavior) An agent lets the agent with the highest delay decide which agent in the negotiation obtains a constraint
- **LP.5** (behavior) If delays are equal among agents, the agent lets the agent with the highest distance to cover decide which agent in the negotiation obtains a constraint
- **LP.5** (behavior) If it is deemed that an agent has to resolve the collision at hand, the agent always first tries to wait to resolve the collision, and by this, aims to maximize fuel savings.

5.2 Observation radar and negotiation

Each agent is equipped with an observation radar which allows the agent to see other agents. As detailed before, the agents which are visible in the radar transmit their current positions and their plans for the next time steps. Specifically, a radar frequency and distance can be selected. The radar scans in a circular fashion around the agent with as radius the radar distance. This is motivated by the fact that usually, radars in aircraft

emit a signal which is reflected back, hence creating an image. However, the intensity decreases quickly as more distance is traveled and correspondingly, the maximum range is determined by the distance the signal must travel. All points of the circumference of a circle correspond to the same euclidean distance, and therefore a circular scanning mode is the most logical.
It is reasonable to choose the radar distance small enough that not the entire map is covered but large enough that approaching agents are detected in time.
Agents within each other's radar distance will negotiate over how collisions are being resolved (i.e., which agent will receive a constraint). Moreover, if agent i sees agent j which also sees agent k, their negotiation will be joined as displayed in Figure 2. This is due to the fact that agent j cannot coordinate its plans without considering both agents i and k.

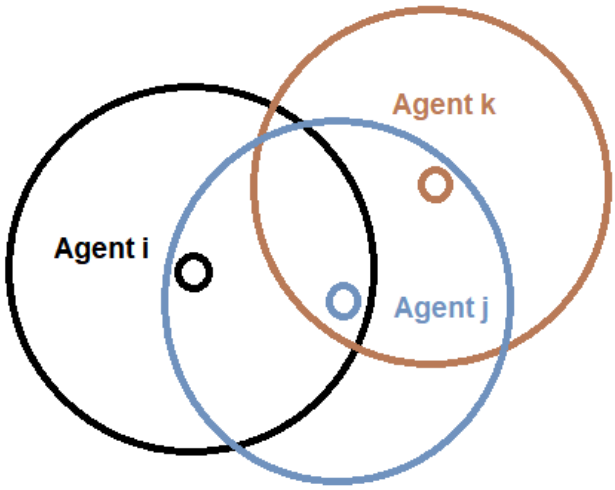


Figure 2: Even though agents i and agents k are not within each other's radar distance, they are still put in the same negotiation as decisions of agent j must be coordinated with both agents

For the negotiation part of the planning, it is always first aimed to resolve the conflict by waiting (i.e., not by re-planning using A*). As detailed in 5.5.1 and its corresponding pseudo algorithm, a certain maximum time to wait is admissible to resolve a constraint. By this, it is aimed to increase fuel savings at the cost of delay time. If the constraint is not resolved after a certain time, it is assumed that it cannot be solved by waiting, and, hence, normal A* re-planning with additional constraints is performed.

5.3 Resolution distance

In addition to the radar distance, the notion of a resolution distance is introduced. When agents enter a negotiation, they are aware of each other's plans. Therefore, they can also find out their collisions up to their corresponding goal locations. However, since the local scale of resolution might not foresee agents that come into the radar at a later point in time, it might be nonsensical to resolve all collisions of the agents in the negotiation already. Hence, the resolution distance determines how many time steps ahead the negotiation looks, i.e., in how distant a future collisions are resolved. It seems logical to choose this value not much larger than the observation radar, however, this also depends on how agents on the map spawn and if they travel in opposing directions (in this case new appearing agents on the radar that an agent has not seen before is much more likely).

5.4 Winning agents

As detailed in subsection 5.5, a precedence must be established for agents in a negotiation to establish which agent is constrained in the next time iteration. For this purpose, a cost function determines the current delay for each individual agent. The agent with the highest delay may decide which resolution strategy is chosen. If all agents have the same delay, the agent with the distance towards its goal may decide. If also this does not yield a clear ordering, a utilitarian approach is chosen where the cost of all agents summed up is reduced.

5.5 Working mechanism of the model

The exact working mechanism is detailed in pseudo algorithms 3, 4, 5, and 6. Generally, the algorithm works in the following order:

- The agents look at their radar and detect other agents (Algorithm 3)
- The agents that are within each other's radar distance join a negotiation to resolve their collisions (Algorithm 4)
- Agents are found up to a certain resolution radius (Algorithm 6)
- By considering the cost functions of the individual agents (see section 5.4), a winning agent is selected to decide which node is popped in algorithm 4 (Algorithm 5)

Algorithm 3 Distributed Planning

Inputs: map, start locations, goal locations, heuristics, scan distance, scan frequency, resolve radius
Output: collision-free paths for all agents
$t_S = 0$
while not all agents at goal location **do**
  for agent in agents **do**
    scan_ data for agent ← data obtained by scanning environment
  end for
  negotiation_ list ← find all negotiations based on obtained scan data
  for *negotiation* in *negotiation_ list* **do**
    curr_ collisions= **detect_reachable_collisions**(**detect_collisions**(plans of agents), resolve radius)
    plans of agents in *negotiation* = **negotiate_plans**(*negotiation*, *agent_ info*)
    if *negotiation* cannot be resolved **then**
      return **FAILURE**
    else
      update current positions of agents in negotiation by newly obtained plans
    end if
  end for
  publish new plans and new position of each agent
  $t_S = t_S + 1$
end while
return paths for all agents

5.5.1 Negotiation structure

Pseudo algorithm 4 details how negotiations are held and how solutions to these negotiations are found. As detailed previously, it is the approach of the algorithm to resolve collision initially by simply waiting for the other agent to pass (instead of completely re-planning that agent). Through this, an effort to reduce fuel consumption can be made. Notably, agents only wait if the type of collision is not 'goal', which means that the collision does not occur due to the fact that one agent in the collision has reached the goal. This is because a goal collision is usually not resolved by waiting, but re-planning will likely be required in this case.

Algorithm 4 Perform negotiation between agents: **negotiate_plans**(negotiation, initial_collisions)

Input 1: *negotiation* ▷ all agents involved in negotiation
Input 2: *initial_collisions* ▷ collisions of current plans of agents in *negotiation*
Output: plans of agents in negotiation after resolution
open_list ← *empty*
first_node ← cost caused due to negotiation for each agent, plans of each agent, *initial_collisions*, active constraints ▷ first node: cost caused and constraints empty, plans and collisions are function inputs
push *first_node* into *open_list*
while *open_list* is not empty **do**
  winning_agent = **get_winning_agent**(*negotiation*)
  curr_node ← pop node from *open_list* chosen by *winning agent*
  if no collisions in *curr_node* **then**
    return plans for agents in *negotiation*
  else
    for *collision* in *collisions* **do**
      constraints = **standard_splitting**(*collision*)
      for *constraint* in *constraints* **do**
        agent_id=agent for which *constraint* is active
        queue=*curr_node*
        if type of *constraint* not 'goal' and maximum waiting time of *agent_id* not exceeded **then**
          t_{coll} ← time step when *collision* occurs
          extend plan of *agent_id* to wait at position before collision until t_{coll}
        else
          constraints of *queue* = constraints of *queue* + *constraint*
          plan of *agent_id* = **a_star**(map, agent information, constraints in *queue*)
        end if
        queue['cost']=**calculate_cost**(plan in queue for *agent_id*, *plan*) ▷ compare old and new
        if path is none **then**
          continue
        end if
        replace plan of *agent_id* in *queue* with *plan* obtained
        new_collisions=**detect_collisions**(*queue*['plans'])
        queue['collisions']=**detect_reachable_collisions**(*new_collisions*, resolve radius)
        push *queue* into *open_list*
      end for
    end for
  end if
end while
return **FAILURE**

5.5.2 Determining agent order

Using the ordering system as introduced in section 5, algorithm 5 determines which agent has the highest importance and may choose how the negotiation proceeds.

Algorithm 5 Determine which agent can pop node: **get_winning_agent**(*negotiation*)

Inputs: *negotiation*
Output: list of agents ordered by winning to losing
costs, *distances* ← calculate from current plans of agents in *negotiation*
delays = *costs-distances* ▷ Cost captures both wait time and distance covered
if max(*delays*) is unique **then** return agents ordered from highest to lowest delay
else
  if max(*distances*) is unique **then**
    return agents ordered from highest to lowest distance
  else
    return empty list ▷ In this case, a utilitarian approach to pop the node is chosen
  end if
end if

5.5.3 Resolution distance

The theory introduced in section 5.3 is implemented using pseudo algorithm 6.

Algorithm 6 Find reachable collisions: **detect_reachable_collisions**(*collisions*, resolve radius)

Inputs: *collisions*, resolve radius
Output: collisions that are detectable (*detectable_collisions*)
detectable_collisions ← *empty*
for *collision* in *collisions* **do**
  t_{coll} = *collision*['timestep']
  if $t_{\text{coll}} \leq$ resolution radius **then**
    detectable_collisions = *detectable_collisions* + *collision*
  end if
end for

6 Algorithm Performance

6.1 Approach

To compare the three planners, it is necessary to run all models under a variety of conditions, i.e. the Input Variables. This ensures that a fair comparison can be made, as the performance is highly dependent on these inputs.
Key Performance Indicators are chosen to quantify the performance of each algorithm. These metrics are formulated in a way to represent real-world indications of the performance of a system at an aggregate level. Subsequently, the values for their KPIs are computed and compared between the three algorithms using a set of experiments.

6.1.1 Input Variables

A set of runs with different input variable values is needed to make a fair comparison. In table 1, the input variables and their corresponding data types are shown.

Table 1: Input Variable specification

Input Variable	Data Type	Explanation
Algorithm	Categorical	The algorithm used to find a solution
Obstacle Ratio	Categorical	Different environments with increasing ratio of obstacles
Start-Goal Alignment	Categorical	Describes how the agent's start and goal locations are determined
Number of Agents $[-]$	Quantitative	Amount of agents placed in the environment
Maximum run time $[s]$	Quantitative	Determines the maximum duration of the run

6.1.2 Key Performance Indicators (KPIs)

A combination of metrics is used to analyze and compare the three multiagent path planning algorithms. To assess these metrics correctly, they are separated into two classes: System-Wide Performance Indicators and Agent-Specific Indicators. Below, the Key Performance Indicators are explained briefly and their relevance to real-world situations is noted. Later in this section, their calculations are further elaborated on.

- System-Wide Performance Indicators
  - **Solution Finding Percentage** refers to the ability of the algorithm to find a solution within the given CPU time limit.
  - **Mean CPU time** refers to the time it takes for the algorithm to find a solution.
  - **Scalability** refers to how an increase in demand (No. of Agents) affect the other KPI values. Using Sensitivity Analysis, an assessment is made in Chapter 7.
- Agent-Specific Indicators
  - **Mean Delay Time** refers to the extra time it takes for the agent to get to its goal position, compared to its shortest path. This metric is relevant for the aircraft taxiing application as delay times are usually important KPIs for airline experience and reputation.
  - **Maximum Delay Time** of an experiment (run) is measured by taking the maximum value of the delay times of all aircraft.
  - **On Time Performance (OTP)** refers to the percentage of agents whose travel time is not more than 15% greater than the travel time of its shortest path. For airlines, this is a well-known KPI to measure service quality.
  - **Total Fuel Usage** of agents is assumed to be linearly dependent on the total travel distance of an agent. Considering environmental impact, this is an important metric in this research.

6.1.3 Schedule of Runs

In the table below, for each Input Variable, the values used in the runs for comparison are specified.

Table 2: Schedule of Experiments per Algorithm

Input Variable	Values	Sum of variations
Algorithm	CBS, Prioritized and Distributed	3
Obstacle Ratio	Fig 1.1, Fig 1.2 and Fig 1.3	3
Start-Goal Alignment	random, random_left_right and random_top_bottom	3
Number of Agents [−]	5, 10 and 15	3

A set of 81 different input variables are used to generate experiments. Each of these sets of values is used 333 times to compensate for the randomness in the Start-Goal Alignments. This results in a data frame of 26973 runs to compare the three algorithms (8991 per algorithm). The maximum run time is set to 10 *seconds*. The Start-Goal Alignments correspond to different ways agents are generated on the maps, specifically:

- random: Agents have random start and goal locations
- random_left_right: Agents have random start locations in the left margin of the map and random goal locations at the right margin of the map
- random_top_bottom: Agents have random start locations in the top margin of the map and random goal locations at the bottom margin of the map

Furthermore, for the Distributed algorithm, the radar distance was chosen to be 5, and its frequency $1/s$. The resolution radius was set to 7. More information on these parameters is provided in section 5.

6.2 Results

The different working principles of the algorithms make their performance heavily dependent on the Input Variables. Analysis of how the Input Variables correlate with the KPI values is needed to identify the strengths and weaknesses of the different planners. Firstly, average KPI values for all experiments are computed and compared along the three algorithms. Thereafter, each KPI is further investigated and analyzed.

6.2.1 Key Performance Indicators

The correct interpretation of the data is of high importance. When averaging out the results to get representative data, the following steps have been taken to avoid inaccuracies. To obtain the correct Mean CPU Time, Mean Communication Costs, Mean Delay Time, OPT, and Average Fuel Usage, solely values of experiments where all three algorithms were able to find a solution are included. The performance is analyzed for all number of agents individually. The success rates were found to be heavily dependent on the number of agents (refer to section 6.2.2). In Figure 3, the averages of total time, fuel consumption, delay time, and max delay are plotted as a function of the algorithm (Distributed, CBS and Prioritized). As can be seen clearly, CBS performs best when considering the costs of the algorithms. Notably, the fuel consumption is lowest for distributed, which corresponds to the aim that was introduced in section 5. However, it is also clear that this comes at the expense of additional delay introduced.

Moving on, considering the same plot for $n = 10$ agents (see Figure 4). Results for $n = 15$ agents are not plotted as most of these experiments failed, as further detailed in section 6.2.2. The results look very similar, however, it can be noted that Distributed planning loses on all important performance metrics in this case. Fuel consumption, previously slightly lower, is now higher for Distributed than for CBS.

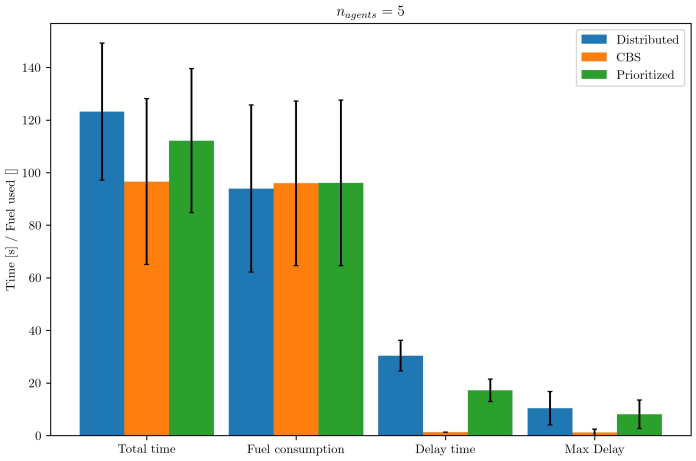


Figure 3: Average of key performance indicators, alongside their standard deviation as a function of algorithms for n=5 agents

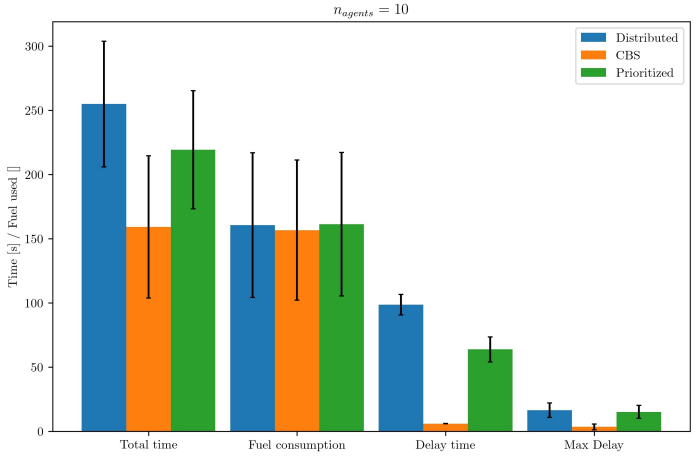


Figure 4: Mean of key performance indicators, alongside their standard deviation as a function of algorithms for n=10 agents

As introduced in 6.1.2, On Time Performance (OTP) is another important performance metric. Figure 5 shows the ratio of aircraft that are able to reach their goal with no more than 15% delay as compared to their paths being planned with independent A* planning. Here, it is clearly visible what could have already been assumed from Figures 3 and 4. Especially for $n = 5$, an OTP of almost 100% is achieved for CBS. For Distributed and CBS, however, on-time performance is achieved in significantly fewer cases.

6.2.2 Solution Finding Percentage

Analyzing the success rates of the different algorithms as a function of agents, the success rates are plotted in Figure 6. For the failures, distinctions are made between an algorithm failure and a timeout failure. Since many experiments were run, a time limit had to be chosen for the experiments. Hence, the failures due to timeout are especially high for experiments with many agents. Notably, the algorithm that outweighs the other algorithm is not always the same for all agents. Specifically, for lower agents, CBS is more likely to find a solution, whereas, for higher agents, Prioritized was able to find more solutions. This could however also be due to the time constraint, as algorithm failure for a high number of agents was exclusively due to a timeout failure.

6.2.3 Mean CPU Time

At last the CPU time KPI was analyzed, again for our three experiments. For this purpose, only the cases where the algorithm was able to find a solution were considered. It is for this reason that Figure 7 does not include $n = 15$ (as explained previously). As can be observed, for increasing agents, the CPU time for the distributed

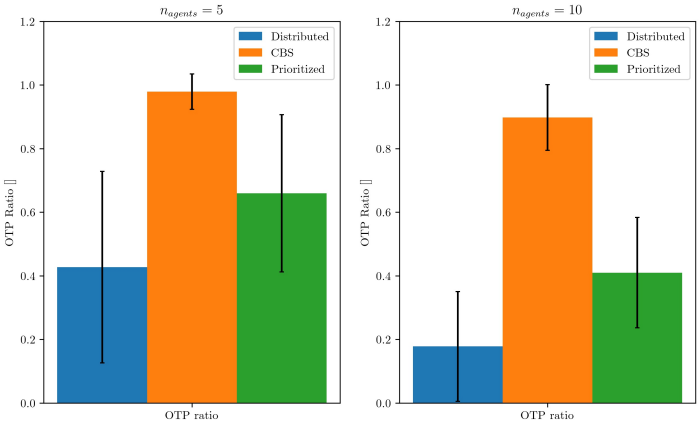


Figure 5: Mean On Time Performance of the three planning algorithms for $n = 5$ and $n = 10$ agents

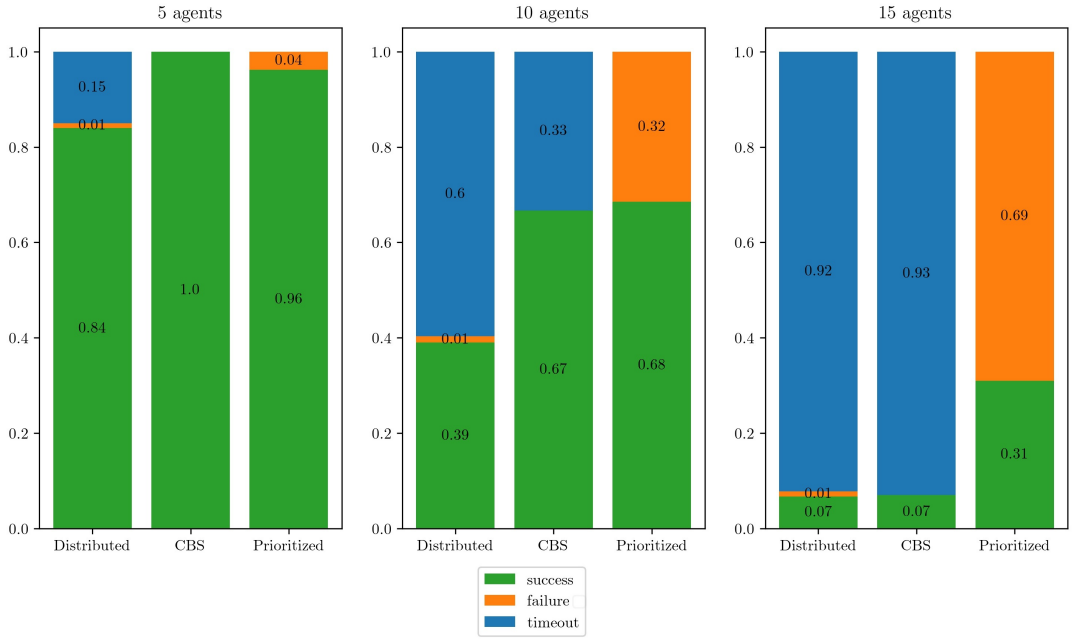


Figure 6: Success rates of different algorithms with increasing agents. As can be seen clearly, higher agents cause the algorithm to fail more often. Failures are distinguished into algorithm failures and timeout failures

planning algorithm is lower than for CBS. This could be seen as an advantage for the Distributed algorithm. Please note that some of the KPIs from 6.1.2 are analyzed not in the performance section but in section 7.

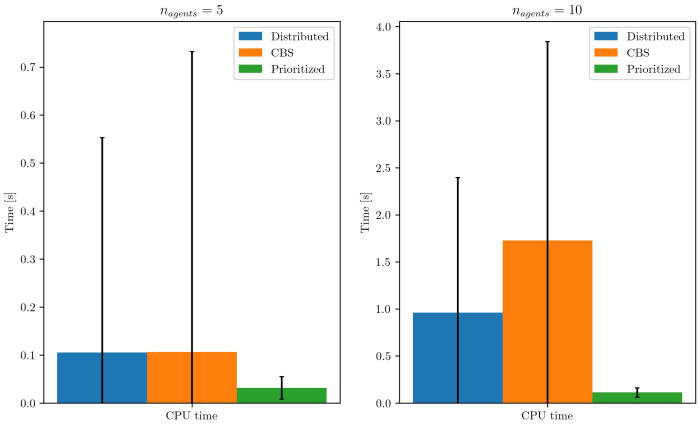


Figure 7: Mean CPU time of the three planning algorithms for $n = 5$ and $n = 10$ agents

6.3 Statistical Significance

To determine whether the three planning algorithms have significantly different outputs, the mean delay time caused by the different planners was compared. The aircraft data of different start-goal locations and different obstacle ratios were combined for each algorithm and each demand: low (5 aircraft), medium (10 aircraft) and high (15 aircraft).

A right-tailed Mann-Whitney U-test with $\alpha = 0.05$ was conducted on the combined populations to compare the mean delay times of each algorithm for each of the demand situations.
Test hypotheses:
H_0: Delay time of Population 1 is smaller than or equal to delay time of Population 2
H_1: Delay time of Population 1 is greater than delay time of Population 2

6.3.1 Low Demand

With low demand on the airport, 5 aircraft are moving from start to goal location.

- The distribution of the average delay time of the CBS algorithm is greater than the distribution of the average delay time of the Distributed algorithm. The p-value is: 0.0. The null hypothesis is rejected.
- The distribution of the average delay time of the Prioritized algorithm is greater than the distribution of the average delay time of the Distributed algorithm. The p-value is: 0.0. The null hypothesis is rejected.
- The distribution of the average delay time of the Prioritized algorithm is not greater than the distribution of the average delay time of the CBS algorithm. The p-value is: 1.0. The null hypothesis is **NOT** rejected.

6.3.2 Medium Demand

With medium demand on the airport, 10 aircraft are moving from start to goal location.

- The distribution of the average delay time of the CBS algorithm is greater than the distribution of the average delay time of the Distributed algorithm. The p-value is: 0.0. The null hypothesis is rejected.
- The distribution of the average delay time of the Prioritized algorithm is greater than the distribution of the average delay time of the Distributed algorithm. The p-value is: 0.0. The null hypothesis is rejected
- The distribution of the average delay time of the Prioritized algorithm is not greater than the distribution of the average delay time of the CBS algorithm. The p-value is: 1.0. The null hypothesis is **NOT** rejected.

6.3.3 High Demand

With high demand on the airport, 15 aircraft are moving from start to goal location.

- The distribution of the average delay time of the CBS algorithm is greater than the distribution of the average delay time of the Distributed algorithm. The p-value is: 0.0022. The null hypothesis is rejected.

- The distribution of the average delay time of the Prioritized algorithm is greater than the distribution of the average delay time of the Distributed algorithm. The p-value is: 0.0022. The null hypothesis is rejected.
- The distribution of the average delay time of the Prioritized algorithm is not greater than the distribution of the average delay time of the CBS algorithm. The p-value is: 1.0. The null hypothesis is **NOT** rejected.

7 Sensitivity Analysis

Some types of sensitivity analysis with varying demands (number of agents) have already been depicted in the chapter above. In this chapter, the more elaborate sensitivity analysis performed on the results is shown.
To compare how the three algorithms react to a variation in a single one of the Input Variables, a set of inputs is chosen to be the Baseline Condition. From there, changes in the values of the Input Variables are introduced to obtain the effect it has on each KPI for each algorithm.

7.1 Baseline Conditions

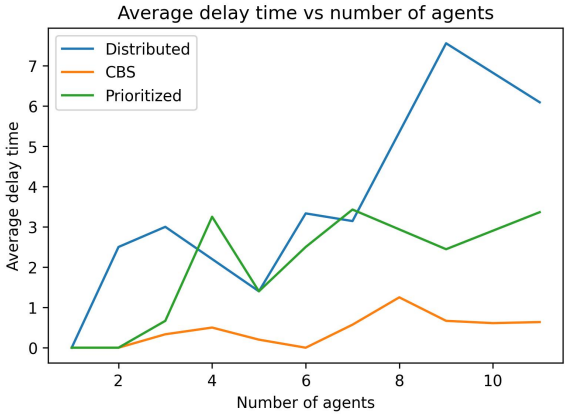
Table 3 shows the baseline conditions for the Input Variables.

Table 3: Baseline Conditions for Input Variables

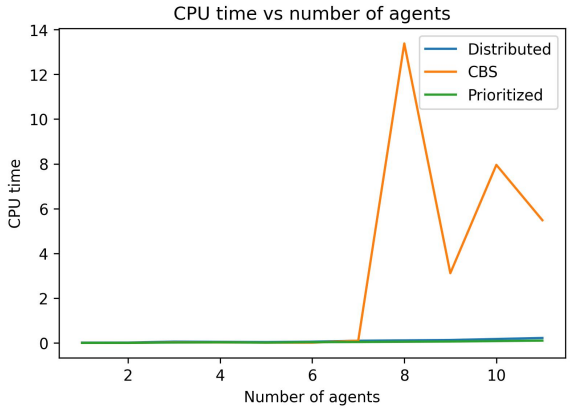
Input Variable	Baseline Value
Obstacle Ratio	Fig 1.2
Start-Goal Alignment	random_left_right
Number of Agents	10

7.2 Demand Sensitivity

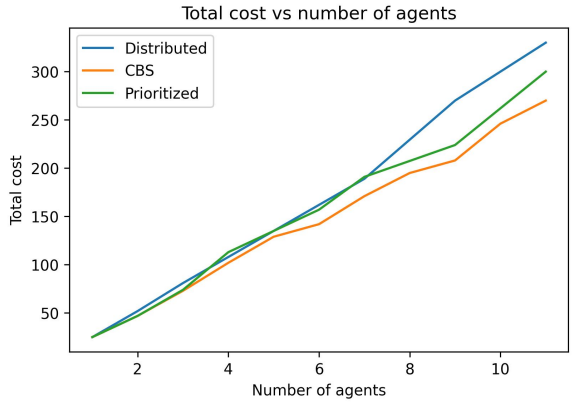
To compare each algorithm's scalability (reaction to an increasing demand), sensitivity analysis is done for the Baseline Conditions, but with an increasing amount of aircraft from 1 through 15. Results for several KPI are shown in the figures below.



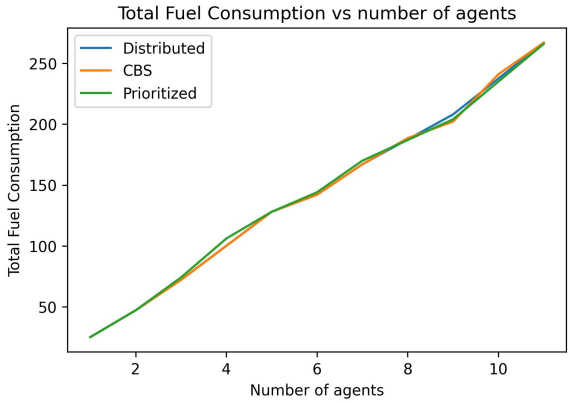
(a) Average Delay Time plotted against No. of Agents



(b) CPU Time plotted against No. of Agents



(c) Total Cost plotted against No. of Agents



(d) Total Fuel Consumption plotted against No. of Agents

Figure 8: The sensitivity to an increasing amount of agent for each Algorithm

As shown in Fig. 8., an interesting sensitivity with respect to the average delay time is observed. Although each algorithm's delay times typically get larger with an increasing demand, it is not the same for each increase and each algorithm. Arguably, this effect is caused by the layout of the environment.
Fig. 8, shows how the CPU time scales with an increasing demand for each algorithm. Notably, although not consistently, as the demand is getting above 7 aircraft, the CBS Planner takes a considerably longer time to compute a solution.
The performance of each algorithm's total cost is similar until more than 7 aircraft need to be planned. Note that this is at the same demand where the CBS Planner takes considerably more time. Comparing CBS and Distributed, it could be noted that where CBS continues searching for the lowest total cost, Distributed saves CPU time to compromise in delays. Note that it does not lose performance on Total Fuel Consumption.

8 Discussion and Conclusion

8.1 Conclusion

The goal of this report is to describe how Distributed Planning, CBS Planning, and Prioritized Planning compare as methods for planning and coordinating taxiing aircraft. All planners have been used to find solutions to certain problems. Although the problems being of a very specific nature, variations in the problems have been introduced to quantify the algorithm's strengths and weaknesses.
Statistical Significance in chapter 6.3 showed that for any demand, CBS is the preferable Planner to reduce delay times. However, the sensitivity analysis in chapter 7, showed that its CPU time is exceptionally higher.
The Distributed Planner was designed to make quick decisions and prefer waiting at its location over emigrating from its shortest path, in order to prevent fuel consumption rising. Inevitably, this resulted in longer delays and a lower solution finding percentage.
As the Key Performance Indicators' values vary to great extend, not one algorithm can be appointed as best performer. The CBS Planner tends to score well, but it should be noted that in high-stakes situations where quick decisions need to be made (e.g. with changing environments), the CBS Planner may have its deficiencies.
Due to the above findings, it is concluded that the Distributed Planner qualifies as a viable alternative to the CBS Planner and Prioritized Planner in particular situations. Although, further research should be done to improve its model.

8.2 Discussion

This section describes discussion points, assumptions, and suggestions for further research regarding this topic.

- For Algorithm Performance and Sensitivity Analysis, a maximum CPU run time of 10 *seconds* and 20 *seconds* was used, respectively. Longer maximum CPU times may result in more accurate comparisons.
- For Algorithm Performance and Sensitivity Analysis, 26973 and 450 runs were done, respectively. More runs and more variations in the input variables may result in more accurate comparisons.
- The problems handled by the planners were of a rather specific nature. Further research could explore a more diverse set of problems to extend knowledge on the performance of the algorithms.
- Although the Distributed Planner preferred waiting over emigrating from its shortest path and by doing so, restraining the fuel consumption, waiting may not be helpful if the constraint it avoids is an edge constraint. Further research could improve the Distributed Planner by additional communication between the agents regarding edge constraints.
- No statistical analysis was applied to the sensitivity analysis conducted in the research. Further could expand on the significance of these results.