Target Search Planning in Uncertain Environments

Hao Wang

University College London

Gower Street, London, WC1E 6BT

A dissertation submitted in partial fulfilment

of the requirements for the degree of

Master of Philosophy

of

University College London

Department of Computer Science

University College London

March 2015

Declaration

I, Hao Wang, confirm that the work presented in this thesis is my own. To the best of my knowledge and belief, where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Hao Wang

March 2015

Abstract

The autonomous robots are useful for lot of things, such as rescue in dangerous environments. In this thesis, we consider how autonomous robots, the Unmanned Aerial Vehicles (UAVs), make a plan to travel in an indoor uncertain environment. At the same time, the robots will observe and update the environment representations with their on-board sensors and plan the path for each robot in the robot group. They will avoid collisions and cooperate with others in the Complete Mission Process (CMP), which includes all operations of robots before the mission is completed (all targets are visited).

The environment cannot be represented exactly because of the inaccurate representation model and the sensor noises. In order to complete the mission efficiently, single robot requires a method to plan a path for efficient travelling from a start point to a target point, plan an assignment for visiting all its targets one by one. For multiple robots in a robot group, we need to plan an allocation for allocating multiple targets to multiple robots in order to make sure that all robots can cooperate together. All these planning operations have to be done based on an inexact representation of the environment.

This thesis focuses on the path/assignment/allocation planning problem in environments which are not completely known, based on a reduced/simplified — Partially Observable Markov Decision Process (POMDP) — framework. The former researches only consider the initial plan but neglect the later replans. Our approach considers the plan and the re-plans from the start to the completion of the mission. Our novel Monte Carlo based planning approaches will plan a path for one robot to move efficiently from one point to one target, plan an assignment for one robot to visit multiple targets by travelling the shortest route and plan an allocation for multiple robots to cooperate and visit multiple targets as soon as possible (the planning time plus the travelling time is minimized).

Our approach is based on a Monte Carlo sampling strategy. In order to decrease its computational cost, two strategies are proposed. We then extend our approach to multiple robots and multiple targets scenario.

Finally, the approaches are extended to multiple robots and multiple targets scenario. They are characterised and evaluated experimentally through simulation. When we compare it with similar methods from the literatures, our approach can provide the better solution.

Acknowledgements

Sincerely, I thank all people who encouraged and supported me during my research period.

Contents

List of Figures

List of Symbols

The following symbols are used throughout this thesis:

k : the interval index

x : state vector

X : state space

u : action

U : action space

$r(x)$: reward on action x

R : reward space

T : transition probability space between states

γ : discount factor

π : policy to select an action

$V(x)$: value function on action x

O : observation space

Ω : observation probability space

B : belief space

c_0 : the initial information available to the robot

$\tau(b, u, o)$: transition at belief state b, action u, and observation o

M_{oc} : Occupancy Cell (OC) Map, where each grid cell records an independent probability of being occupied by an obstacle

$|M|$: grid cell amount in map M

A : the set of Targets

$|A|$: the number of Targets

(x, y) : grid cell coordinate

$m_{x,y}$: grid cell at coordinate (x, y)

Q : robot team

$\mathbb{D}$: a finite set of N robots

$\mathcal{U} = x_i \times U_i$: a finite set of possible joint actions, U_i is the finite set of individual actions for agent i

$\mathbb{O} = x_i \times O_i$: a finite set of possible joint observations.

M^i : the i^{th} map instance sampled based on a POMDP model

M_R: the actual and real environment

$p(M^i)$: the probability that M^i is the real environment

b : a belief in the POMDP model

b_i : the i^{th} belief node in the POMDP model

b_{start} : the start belief node

b_{target} : the target belief node

$p(b)$: the probability that the belief node b is empty

1 Introduction

1.1 The Planning Problems

We focus on the problem of developing algorithms to allocate multiple targets to multiple robots and find paths for autonomous robots to visit these targets one by one in an uncertain environment. We simulate the autonomous robot, like an Unmanned Aerial Vehicle (UAV), that can detect and update the environment representation when moving in uncertain environments. The uncertain environment, like a damaged building in an earthquake, is not known precisely at the time when the robots are launched.

In our scenario, there is a building with some dramatic events happened and the rescue is needed. In many situations, the prior information about the construction of the damaged building, such as the plan of a building, is available. However, as the Earthquake Engineering Field Investigation Team (EEFIT) reported in [4][5] (shown in Fig. 1 and Fig. 2), the effect of a catastrophic event (such as an earthquake) is to change the environment. So, any prior knowledge about the environment only shows what things used to be, not what they are now. As we can see, Fig. 1 shows the effect of an earthquake damage in a building in Bhuj, India. The earthquake opened a hole in a wall, and a pre-existing the door was locked. The damage caused by an earthquake in Ji-Ji Taiwan is shown in Fig. 2. We can see that the wall is broken and the debris from the collapse is lying on the ground, where the robot should avoid the curtains and the debris to passing the left door. In Fig. 3, the earthquake in Erzincan Turkey broke the cantilevered masonry panel, and the hole is unpredictable.

In emergency response situations, response time is critical [1] to save more survivors. The rescue team should enter the damaged building to save the survivors as soon as possible. However, it is dangerous to send the rescue team members into the damaged building directly. In case of this situation, teams of robots could be dispatched to search and locate missing persons inside a damaged building [2][3]. In our research, we use Unmanned Aerial Vehicles (UAVs) as the robot to do our research, because they can overcome the debris on the ground.

Fig. 1 Structural damage to a building caused by an earthquake[4]

Fig. 2 The broken internal wall and the debris on the ground [5]

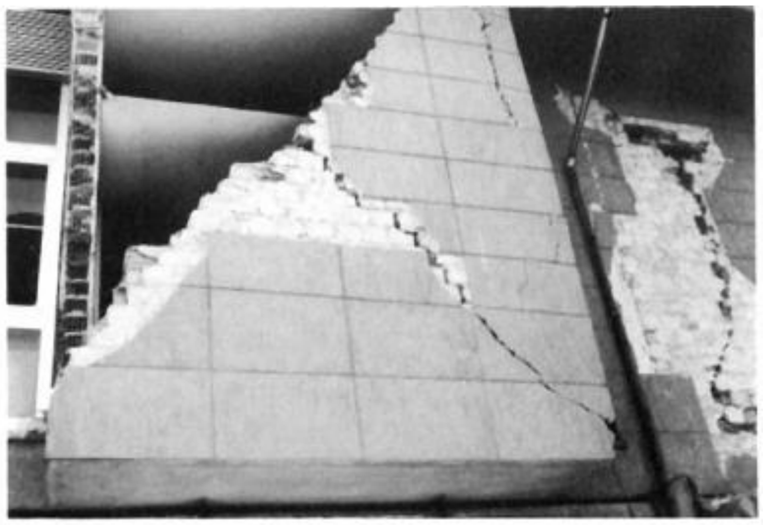


Fig. 3 Diagonal cracking of a cantilevered masonry panel in an earthquake [8].

After the robots have searched and located the survivors' exact locations inside the damaged building, the rescue team members can work more efficiently to save the survivors [1]. Robots can use the building structure map as a baseline to represent the uncertain environment of the damaged building and make their search plans autonomously. These maps of building structures are available for the National Safeguard (or Rescue Team) in most countries [6] (e.g., all public buildings' structral maps are available in some cases, for example, the National Government Agencies and Local Governments in Japan [7]).

In order to find the survivors efficiently, the rescue team members should evaluate the earthquake power to estimate the building damage and set up some targets for the robot team to visit. The robot platform has sensors to detect and update the environment representation when the robot is moving, for example in [80][90]. The navigation system will make a re-plan, including re-allocate targets for robots, if the robot is stopped by an obstacle.

Therefore, we define the idea of Complete Mission Process (CMP) as: we send out the robots to visit those targets and all operations of robots are included before all targets are visited (or make sure that the remaining targets are not reachable). These operations include detecting and updating the representation of uncertain environments, planning and re-planning the allocation for all robots, and recording the actually travelled route for each robot. Although the full robot operation carries out many replans, most existing algorithms are only assessed based on their results from a single plan. This fundamentally does not represent the true behaviour of the system. Therefore, we are taking a holistic approach to the whole end-to-end problem, including the initial planning and the later re-planning(s) before the mission completed. As we show later, the full performance can be very different. That is why we believe that comparing algorithms using the CMP is the only meaningful manner.

As illustrated in Fig. 4, the CMP may include several re-plans until the problem is solved. The uncertain environment is represented with an Occupancy Cell (OC) Map.

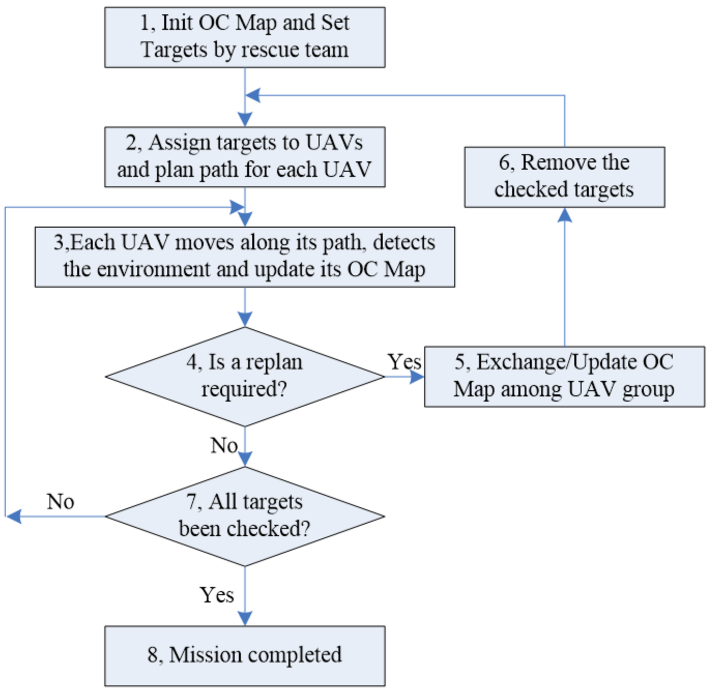

1, Init OC Map and Set Targets by rescue team
2, Assign targets to UAVs and plan path for each UAV
6, Remove the checked targets
3,Each UAV moves along its path, detects the environment and update its OC Map
4, Is a replan required?
Yes
5, Exchange/Update OC Map among UAV group
No
No
7, All targets been checked?
Yes
8, Mission completed

Fig. 4 The Complete Mission Process (CMP)

In prior research, the effects of the uncertainties are added into the planning algorithms using different strategies. Maeda [47] proposed a planner to take the uncertain terrain friction into account, because the uncertain terrain feedback will change the result of the robot's action (e.g. change the direction of movement). He proposed an algorithm to estimate the terrain feedback of different terrains and the planner uses this feedback value to adjust the planned path. Melchior [50] did research into the terrain uncertainty by extending Rapidly-exploring Random Tree (RRT) node to a group of points, where each point is a possible robot’s position when the robot reaches the RRT node. Mohibullah [107][108][109] researched the missing person’s possible moving trajectories in an outdoor environment. These trajectories are the heuristic to make a plan for robots to find the missing person.

In this thesis, we research the planning problem, which is affected by the uncertain locations of obstacles. Robot(s) will make re-plans many times to avoid obstacles in an uncertain environment. The targets may be re-allocated to the robot team, in order to visit all targets as quickly as possible. The actual travelled route of a robot is a sum of many segments of these planned and re-planned paths. Therefore, we want the actual travelled route of the robot to be the shortest.

If there is only one robot and one target, we can plan a path for the robot to visit the target. If there is one robot and multiple targets, we need to plan an assignment which contains the paths and the order of targets to be visited by the robot. For multiple robots and multiple targets, we will plan an allocation. In the allocation, the targets are allocated robots, and the assignment is planned for each robot.

Therefore, we want to find the shortest travelled route for a single robot, or the shortest travelled routes for a group of robots, in the CMP. For the robot group (there are several robots), we will find that the longest travelled route of robots is the shortest in the CMP.

To test the performance of different algorithms, we created a simulator which is capable of simulating the netire CMP: given a prior map and a set of destinations, it will step through simulating the movement of the robots, the output of the sensing systems, the update of the occupancy map and the trigger to do planning.

The prior works [12] consider various aspects of allocation planning for an robot group: remove uncertainty using a threshold [12] [38] [39] [96] [40], plan a sub-shortest allocation and adjust it gradually [102] [41] [47] [48] [49] [50] [51] [52], plan with the Most Likely State [42], vote the different decisions/policies [43] [44], use information entropy as heuristic [44] [97] [18], cluster nodes based on their locations [61] [62] [63] [64].

In this thesis, we consider the problem from the view point of Complete Mission Process (CMP). We also consider the Complete Mission Time, which contains two parts: one is the travelling time, the other is the planning time. Each re-plan will increase the planning time. The travelling time depends on the robot actually travelled route. We consider how to decrease the re-plan number and decrease the robot actually travelled route length at the same time. The difficulty we met is that the actual travelled route is a combination of many segments of the planned and re-planned paths for each robot, because the robot has to make re-plan many times in uncertain environments in CMP. In our research, we design our algorithm by studying the combination of the plans with the shortest travel length and the lowest probability of replanning. When we compare the robot actual travelled routes of different algorithms in CMP, our proposal produces much better results.

1.2 Problem Representation Model

In order to design our algorithm, we need a model to present our research problem. Markov Decision Processes (MDP) and Partially Observable Markov Decision Process (POMDP) are two popular models which are used by many researchers to design their planning algorithms. We compare these two models and select the POMDP model for our research problem and design our algorithm.

1.2.1 Markov Decision Processes (MDP)

A Markov decision processes (MDP) is a mathematical framework for modelling decision-making problem in situations where outcomes are decided by the current state and the control action [9]. The missing person search-planning problem requires a plan which contains a decision sequence from the initial state to the target state. When we make a decision at the initial state, we evaluate and compare different possible choices for the decision. Therefore, MDPs are useful for studying a wide range of planning problems. It forms the basis of our further works.

MDPs can be solved via dynamic programming [10] or reinforcement learning [11]. MDPs were known at least as early as the 1950s [9]. Today they are used in a variety of areas, including robotics, automated control, economics and manufacturing [9].

More precisely, an MDP is a discrete time stochastic control process, which provides a general mathematical model for the interaction between a robot and the world [9]. The world state is represented by a set of variables (discrete or continuous). The action (like move forward, turn left or turn right) chosen is based on a robot's possible observations and knowledge. The robot's state is transferred based on the pre-defined transfer probability of the robot's actions. Many classical AI planning algorithms can be formulated as special cases of MDPs, like the chess player [11].

MDP considers the simplified matters with an assumption [13], in which the previous states and actions will not affect the future states if we know the current state. Therefore, the future transition only depends on the current state and action. Generally, this assumption of the real word is fairly accurate, if we are given enough descriptive state vectors [12].

By framing the problem as an MDP, we can design a planning algorithm to compare different plans, including allocations, assignments and paths [14][15][16]. The output of many algorithms is a value function, which calculates a value of planned result for every possible state [12]. The greedy strategy can select the actions for a robot to immediately reach high-value states, since the plans are implicitly encoded in the action evaluation function.

MDP algorithms have solved many challenging problems, but the robot cannot always get the perfect knowledge of the states of the real environment for most cases. Therefore, the partially observable MDP is a more realistic choice.

1.2.2 Partially Observable Markov Decision Process (POMDP) Model

In an MDP, we assume that the robot completely knows the state of the world, including probabilities that the action transfers robot to different new states. This assumption is rarely true in real scenarios, especially for the problems researched in this thesis. A Partially Observable Markov Decision Process (POMDP) extends an MDP by adding imperfections to the robot's observations about the world. The assumption of the POMDP formulation is that the likelihood of observations about the hidden states can be decided by statistical probability distributions [12].

In [117], Whelan reports the ability to map a large environment with a root mean squared error of approximately 20cm. Since this is significantly smaller than the size of the grid cell used in our planner (60cm on a side), we assume that the occupancy state of a cell can be precisely determined by a single measurement from the robot (if the obstacle is not blocked by another obstacle). Furthermore, we assume that the robot can localize itself correctly to the grid cell which contains it. However, because the robot can only observe a subset of the environment, uncertainty still persists.

While a robot, using an algorithm based on MDP, can make its decision based on the state, a robot using an algorithm based on POMDP has to make its decisions based on a history of its actions and its observations [12]. Because the history records what the robot knows about the world, the history dependency makes the planning problem more complexly. The Markov property implies that a robot can safely summarise all history in a sufficient statistic state and ignore the long history descriptions [9]. The POMDP equation also assumes that its state obeys the Markov property, except that the state is not directly observable. The entire action and observation history is relevant, but can be summarised in the current representation sufficiently.

To avoid the dependency on explicitly storing the entire history, the dominant approaches prefer to maintain a belief space to describe the probability distribution over all states. The belief is a sufficient statistic for the history. That is, if a robot knows the current belief probability, the observations and actions which lead to that distribution are irrelevant for predicting the future [12]. Therefore, POMDPs provide a general and flexible framework for realistic problem, which enables people to propose algorithms to solve their problems in different areas. For example, to model the mobile robot search planning problems, the robot state, action, and observation spaces are large and cannot be represented exactly[12][13]. The robot will travel in some kinds of scenario and use sensors such as cameras, laser range-finders, and acceleration transducers to gather information [3][39][40][80][90][109]. The sensor measurements are imperfect because of the noise and limited detection distance. In this context, a POMDP solution represents a plan which allows the robot to gather the information and brings it to its target.

We will show the formulations of a POMDP in Chapter 2, where we demonstrate that a POMDP is a special kind of MDP. An unobservable state can be replaced by the observable information-state, which also can be summarised by a belief-state [12]. As the robot moves in the Complete Mission Process (CMP), the variables of POMDP are being updated. When the robot is blocked, it will make a re-plan with the new updated POMDP representation.

1.3 POMDP for Targets Allocation Planning

The effect of a catastrophic event, such as a fire, an earthquake or an explosion, leads to changes the environment. There might be lots of survivors. A key issue to the problem is to identify whether they are around, and where they might be. If we send the rescue team to search the entire building locating and saving survivors, more people will be endangered in the damaged building. Therefore, the rescue team can use a robot group to inprove the search efficiency in these uncertain situations.

Our research was carried out in three main parts. The first explored how to plan an efficient path in an uncertain environment. The second extended this to examine how to construct a plan to allow one robot to visit many targets. Third, we considered the problem of target allocation in which multiple targets are assigned to a group of robots. We introduce our research of these three parts in the following chapters of this thesis.

We use discretised spaces in our research. In terms of Fig. 5 (adapted from [17]), on the extreme right, the combination of a fine description with accurate calculations of utility will lead to perfect plans but it is hardly exist in realistic problems. On the extreme left, although simple heuristics (like distances to target) require little computation, they cannot provide a good approach to evaluate the affection of the uncertainty. The simplest approach is based on the assumption that both the environment and the pose of the robot(s) are known perfectly, or determine the global state at anytime. Most research defines heuristics or evaluation functions to calculate uncertainties in the map and improve their planning processes. An example of those algorithms of these approaches is showed in section 2.5. However, these approaches can only explain the evolution of the probability distributions over a very short planning horizon [12]. Therefore, the most practical planner needs to have a good trade-off between these two extremes, and it should be lying somewhere in the middle of the bar.

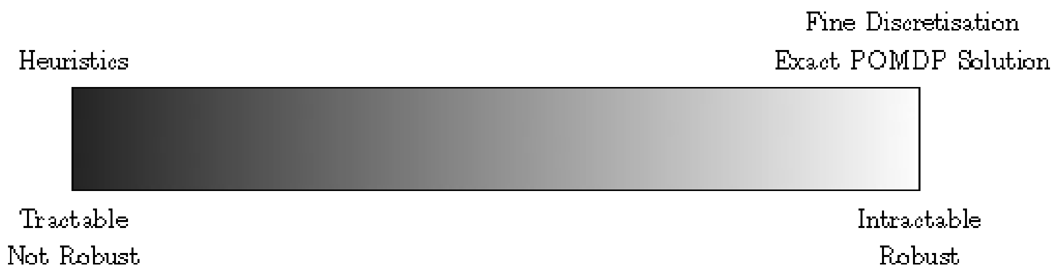


Fig. 5 POMDP solution methods span from left to right.

By discretising the state, action and observation spaces, we can approximate the uncertainty within a small number of dimensions, such as a group of grid cells. Then, the evaluation functions can use these uncertain grid cells to direct the searching direction of the planning process, as in [18][19][20].

1.4 Scope of the Research

We focus on the planning problem in an uncertain environment. Our simulator uses a simple on-board sensor strategy to update the representation of the uncertain environment, which is reasonable [117]. Therefore, the Simultaneous Localization And Mapping (SLAM) [29] is not included in our research.

There are many models of earthquake damage. For example, we can use the earthquake damage models discussed in [21][22][23][24] and the details of the space-dependent damage model correlated ground motions in earthquake events which are proposed in [25][26][27]. Earthquake damage cannot be predicted completely [28], so we can construct an uncertain environment by using a basic and empirical damage models in [21][22][23][24][25][26][27], or by setting the uncertainties manually. The planning system in the simulation system knows nothing about these damage models. The disturbance of the damage model is also supported by our simulator to test the sensitivity of our algorithm.

In our simulate experiments, the actual states of the environment should exist and be known by the simulator, in order to correctly simulate the movements of the robots. The sensors cannot see the situations behind an obstacle, like a solid wall, and their readings are given by the simulator. The robot does not know the exact states of the entire environment and can only make a plan based on its own representations. Our research focuses on how to inject and use the uncertainties efficiently in the planning process.

1.5 Publications

Our research results have been published in two conference papers:

The first one is “Path Planning In Partially Known Environments”, published in the Proceedings of the International Association of Science and technology for Development (IASTED), on 7-9 November 2011, at Pittsburgh of USA. We have introduced the PD Planner in this paper with a two-dimensional uncertain environment. The PD Planner planning process [71] will be introduced in Chapter 3.

The second one is “Reducing the Computational Cost of a Monte Carlo Based Planning Algorithm”, published in the IEEE International Conference on Systems, Man and Cybernetics (SMC) 2013, on 13-16 October 2013, at Manchester of UK. We have introduced two strategies to decrease the computational cost of PD Planner. The strategies’ details [95] will be introduced in Chapter 4.

1.6 Contributions

This section outlines the contributions of our research based on the POMDP model represented problem, and the thesis addresses these issues:

- We have found that the possible shortest paths in uncertain environments are likely to cluster naturally, which can be used to make a plan.
- We have designed the Path Distribution (PD) Planner to plan a path for one robot to visit one target in an uncertain environment. We also test the impacts to the PD Planner in situations where the occupancy probabilities are not known perfectly.

- We also have researched the sampling strategy of PD Planner for large environments. Two strategies are proposed to decrease the computational cost.
- We extend the PD Planner to handle the assignment case, which is called as the Assignment Distribution (AD) Planner. The AD Planner performs path planning for one robot to reach several targets efficiently in uncertain environments.
- Lastly, we have extended the AD Planner to Group Allocation Distribution (GAD) Planner, which can allocate multiple targets to multiple robots and plan path for each robot in a robot group to visit multiple targets efficiently in uncertain environments. Our evaluation of the performance of the algorithms on extensive simulations show that it leads to significant improvements.

The path length is used to evaluate the travelling time for completing the mission. The simulated robot can perfectly detect the uncertain environment within a limited distance, and update its representation of the environment.

We illustrate our planners in scenarios that range from simple to complex. Our first scenario is for one robot to visit one target in a damaged one-floor building. We developed an algorithm which uses clustering of solutions/paths to identify stable structures of paths for the robot to move from one position to another position efficiently in an uncertain environment. Then, the planner is extended to consider multiple targets assignment across multiple floors, for example one robot will visit four targets in a damaged two-floor building. We illustrate the scalability of map size and the fact that our approach is not restricted to a single floor. The assignment planning for one robot in an uncertain environment is named as AD Planner. At last, three robots will visit five targets in a damaged three-floor building, including a free space surrounding the building for robots to move freely. We will figure out how to make an allocation planning for multi- robots. The actual travelled paths of robots using different algorithms are recorded and compared.

1.7 Thesis Structure

In Chapter 2, we introduce the POMDP model and algorithms used in the planning and searching problems in an uncertain environment. We review the related literature, and find that most researchers evaluate their solutions for single planning enquiry.

In Chapter 3, we introduce the Path Distribution (PD) Planner, in a 2 dimensional OC Map. This is the first main challenge of our research: to travel effectively in an uncertain environment. We implemented a simulator to compare our research with other typical algorithms. The experiments show that the robot using our algorithm can reach the target by travelling shorter distance.

The Monte Carlo sampling strategy used in PD Planner requires many calculations when the OC map contains many grid cells. So, two strategies for reducing the computational cost are introduced in Chapter 4: 1) Sampling in Planning Process (SiPP) – only sample the grid cells required in the planning process, and 2) Hierarchical Path Distribution (HDP) Planner – decrease the map dimension based on pre-defined uncertain structures.

In Chapter 5, we extend the PD Planner to a 2.5 dimensional uncertain environment for one robot and multiple targets assignment planning. Then, we extend the approach to plan an allocation for a group of robots. We record the travelled routes of robots using different planning algorithms. The experiment results show that the robot(s) with our planner can travel in a shorter distance to complete the mission (visit all of the targets).

In Chapter 6, we show the critical discussion of our research. Some potential research directions of the future work are introduced, too.

2 Planning and Searching In an Uncertain Environment

This chapter formally describes the problem of planning paths to visit multiple targets in an uncertain environment. Each robot is equipped with a line-of-sight sensor, such as a laser scanner. This scanner can observe the free space in front of the robot out to a fixed detection radius. Here, we use the Occupancy Cell (OC) map to record the occupancy probability of each grid cell, if these grid cells are not detected by a robot.

We will introduce the basic terminology and concepts of MDPs (Markov Decision Process) and POMDPs (Partially Observable Markov Decision Process), and describe how they are used to describe the problem. We use the belief space to represent the state location probability of POMDP, which make the POMDP as a special case of MDP. After that, we will introduce some typical solution methods for POMDPs. Finally, we will figure out the short cuts and limitations in existing literatures and propose our research idea in the following chapters of this thesis.

2.1 Problem Statement

We consider the following situation. A building has experienced a catastrophic event such as a fire or an earthquake, as illustrated in Fig. 1 and Fig. 2. Prior information about the building – specified as a set of plans – has been provided to the emergency services. With these plans specific targets of interest (such as reported locations of people, or general identification of areas where the missing persons are likely to be found – such as stairwells [1]) - have been identified. These targets are defined by the rescue team in the search and rescue indoor environments. The goal is to visit these targets as soon as possible by sending a team of rescue robots to search inside a multi-floor building.

For our research scenario, the first key challenge is how to plan a path to travel efficiently in an uncertain environment. The second key challenge is to assign the targets with an efficient order for a single robot to visit them. The third challenge is how to allocate multiple targets to multiple robots in order to visit all targets as soon as possible.

The prior map before the catastrophic event describes a known environment, where the state of each grid cell is represented as free space or occupied by an obstacle. However, a catastrophic event, such as an earthquake, can have important effects: the building's internal structure may be transformed by some random damages, for example broken walls can block a corridor and holes create new routes through previously impassable walls.

Given these transformations, the map has to be updated to reflect the damage. We use an Occupancy Cell (OC) Map to record the occupancy probability of each grid cell. For example, Fig. 6 (a) is a certain map before transform (black cell is obstacle and white cell is empty), and Fig. 6 (b) is the earthquake damage transformed map, using a perturbation model in [28] (shake the building to propagate the uncertainties in OC Map), where each grid cell has an independent obstacle occupancy probability between 0 and 1.

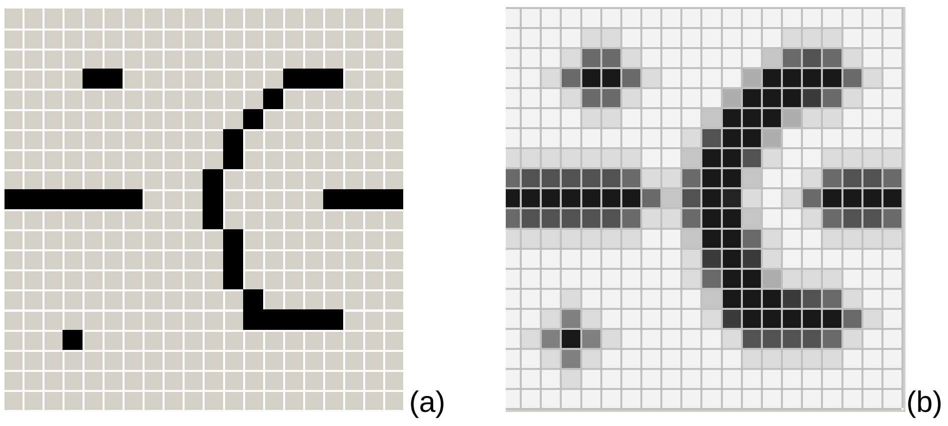


Fig. 6 Map example:a)0/1 map; b)OC map(darker cell had higher occupancy probability)

We should complete the following tasks to find the missing persons in our scenario:

(1) The research team analysis the building plans and probability distributin over the OC Map of the building, and proposes a set of locations where missing persons might be found.
(2) The robots then make a plan and re-plan to search inside the damaged building, each single robot can make a search plan for itself.
(3) The OC Map will be updated while the robot travelling inside the damaged building.

Our planning algorithm is designed to find paths for the robot Group to visit all targets. Our evaluation is the length of robot's actual travelled route.

For step 1, we can initialize the OC Map from an empirical damage model. For step 2, the re-plan condition could be that the current path is not executable, or a new and much better assignment is found. For step 3, we can use different exchange rules depending on the complexity of the scenarios (for example, the obstacles cannot block the wireless connection, or one robot should see another robot when they exchange data).

In our simulator, the sensor cannot see the grid cell's state behind an obstacle. We simplify the sensor noises to be zero, which means that the obstacles within the detect radius can be found exactly (if they are not blocked by another obstacle).

In Fig. 7, for example, the robot starts from the cell marked with a red dot (down-left part in each figure), its current position is marked by a red dot surrounded by a yellow cycle (mid-up part in each figure). The darker cells have higher occupancy probabilities, and the actual obstacles are marked with "x". The robot updated OC Map is shown as the brighter cells along the travelled route (visited by the robot on-board sensors). The red line is the route that the robot travelled, and the yellow line is the current planned path. The robot makes the first re-plan at t_1 because the current planned path (the yellow line) is blocked, see Fig. 7 (a). The second re-plan is launched at t_2, see Fig. 7 (b). The third re-plan is launched at t_3, see Fig. 7 (c).

The robot will make several re-plans before it reaches the target, because the new measurements may find obstacle(s) blocking the last planned path, or showing a shorter path through some new openings in a wall.

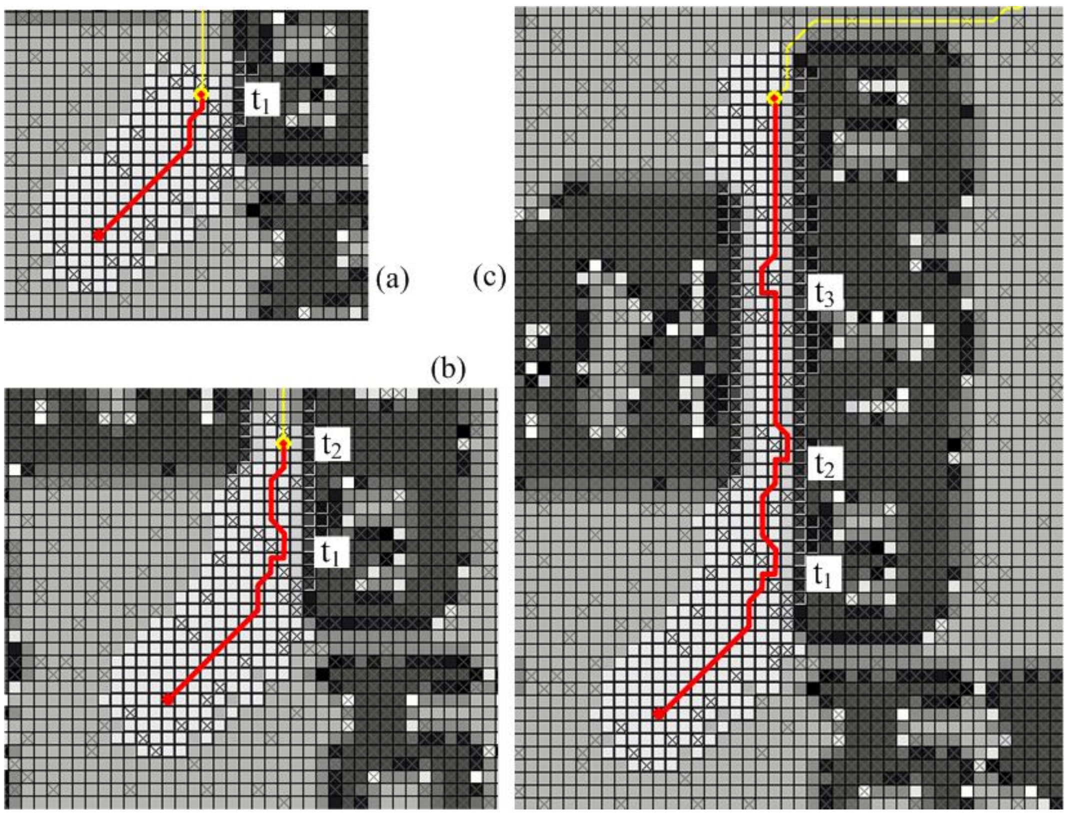


Fig. 7 A robot moving and re-planning at t_1, t_2 and t_3. The yellow line is the planned path, and the red line is the travelled route.

2.1.1 The Occupancy Cell Map

We use an Occupancy Cell (OC) map M_{oc} to represent the uncertain environment and the robot position is described with the grid cell coordinate (x, y) in M_{oc}. We define the **obstacles** as the places where the robot cannot or should not go through, and the sensors cannot penetrate it. So, the robot has to avoid them. The **free space** is where the robot is allowed to go through. When we say that an environment is uncertain, we mean that we are not sure of the number or the locations of the obstacles. The planning algorithms do not make plans that will lead the robot into an obstacle. Therefore, the obstacles that block a planned path are called **unknown obstacles**.

The occupancy grid cell map M_{oc} contains $|M_{oc}|$ grid cells. Each cell is a square although other shapes – such as octagons – could be used.. All cells have the same size. A grid cell can only connect with its immediate neighbours. Each cell has a unique coordinate (x, y). We define $m_{x,y}$ to represent the state (occupied by an obstacle or free) of the cell whose coordinate is (x, y), where

$$m_{x,y} = \begin{cases} 0 & \text{cell is free} \\ 1 & \text{cell is occupied} \end{cases} \tag{1}$$

To describe the uncertainty, a grid cell is not assigned a binary value, but instead is given a continuous value that represents the probability that the grid cell is occupied [35], $p(m_{x,y}) = p(m_{x,y} = 1)$. The cell's empty probability is $q\left(m_{x,y}\right) = 1 - p(m_{x,y} = 1)$. We assume that the occupancy probability for each cell is independent of all the other cells. Therefore, we do not limit the number, shape, position or complexity of obstacles and their probabilities in the environment. However, we do not exploit any potential correlation, in structure of obstacles which might exist, to predict something about what might be happening in other cells based on what the robot has observed.

We implement a simulator to do our research. In our simulator, we use a 2-dimensional floor map of a one floor building and a 2.5 dimensional map for a building with several floors as examples to illustrate the algorithms in this thesis. However, the proposed algorithms are also applicable to general and higher dimensional situations.

The simulator lets us do our research step by step, from simple scenario to complex scenarios. We start from single robot planning to visit single target. Then, a single robot will visit several targets. Lastly, the multiple robots will visit multiple targets.

We now introduce the symbols and the basic evaluations of the research models: Markov Decision Processes (MDP) and Partially Observable Markov Decision

Processes (POMDP). We use these models, as many researchers did, to represent our research problem. Our algorithms are defined to solve POMDP problems, using the MDP algorithms.

2.2 Markov Decision Processes (MDP)

In this section, we introduce the general setup of the Markov Decision Processes (MDP).

The environment used by the Classical Artificial Intelligence (AI) planner is assumed to be deterministic and finite, fully-observable, static and discrete [30]. In order to research the planning problems in this kind of environment, an MDP is used to model the entire environment, including the properties of robots. STRIPS [31] is the first major planning system, which presents the state of the environment with well-defined MDP symbols (like a graphic containing nodes and edges) in order to find a node/action sequence from the start node to the target node. A set of MDP actions are defined, and a set of pre-conditions and deterministic effects on the symbolic state is defined to each action. For example, the action “move to top-right” has a pre-condition that the grid cells of top, right and top-right must be empty. The STRIPS-style planner autonomously makes a fixed action sequence for a robot to move from a start state to a target state.

An important factor to think about is that the robots are required to satisfy various objectives simultaneously, but the classical evaluation only takes one objective into account [32]. For example, robots can maximize the rewards (such as gather the most information with the least amount of energy), but cannot minimize the risks (such as limiting the accumulated probability of being blocked) at the same time. Therefore, a reward function has to be used to specify these objectives [32][16].

An MDP involves a decision-making agent that interacts with a fully observable environment (assume that the sensors can detect the entire environment at any single state), and the action result may be different states with different probabilities. In this thesis, we assume that the robot state is discretised into a set of steps indexed by k. The state vector $x_k \in X$ is used to describe the state of the environment at step k. The environment state includes the updated OC map M_{oc}, the robots states, the targets set A for robot(s) to visit.

At each step, the agent chooses an action $u_k \in U$ which causes the state transfer stochastically from x_k to x_{k+1}, and results in the agent receiving the reward r_k. In our research, we adjust the transition probability from x_k to x_{k+1} and define the transition to be deterministic, because the autonomous robot can adjust its action(s) based on its observations to reach its next position. We can easily evaluate the rewards of different action sequences in the planning process. See Fig. 8 as an example, the agent produces a new action u_k based on the state in each iteration. The world samples a new state x_{k+1} based on the agent's action. The "$k \leftarrow k-1$" box simply alters time subscripts in preparation for the next iteration.

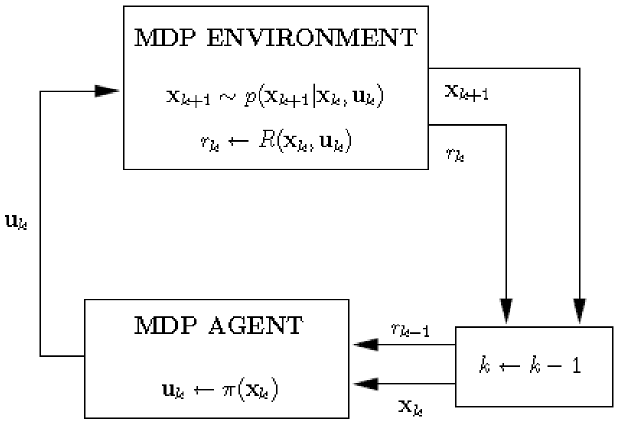

MDP ENVIRONMENT
$\mathbf{x}_{k+1} \sim p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k)$
$r_k \leftarrow R(\mathbf{x}_k, \mathbf{u}_k)$
$\mathbf{x}_{k+1}$
r_k
$\mathbf{u}_k$
MDP AGENT
$\mathbf{u}_k \leftarrow \pi(\mathbf{x}_k)$
r_{k-1}
$\mathbf{x}_k$
$k \leftarrow k-1$

Fig. 8 The MDP model [12].

Formally, an MDP is defined by the tuple

$$< X, U, T, R, x_0, \gamma > \tag{2}$$

where

- X is the state space

- U is the space of actions
- $T(x_k, u_k, x_{k+1}) = p(x_{k+1}|x_k, u_k)$ defines the transition probabilities between the states
- $R(x_k, u_k)$ defines a reward function for executing action u_k, at state x_k
- x_0 is the initial state of the environment
- γ is a discount factor

The agent executes a policy π, which specifies an action for every state:

$$\pi: X \rightarrow U \tag{3}$$

This policy can be seen as a conditional plan. After the first action is taken, the subsequent actions depending on the state transition is defined in T. The Markov property asserts that the state is a sufficient statistic, which means that the past conveys no extra information about the future if the present state is known [13]: $p(x_k \rightarrow x_{k+1}|x_i, u_i, i < k) = p(x_k \rightarrow x_{k+1})$. Then, we have: $x_k \in X$, $r_k \in R$, $u_k = \pi(x_k)$, and $r_k = R(x_k, u_k)$. Their relationships are shown in Fig. 8.

Let $V_\pi(x_k)$ to be a value function and denote the value of executing policy π starting from state x_k. $V_\pi(x_k)$ equals the discounted sum of expected future rewards:

$$V_\pi(x_k) = \sum_{j=0}^{K} \gamma^j \underset{x_{k+j}}{\mathrm{E}}[R(x_{k+j}, \pi(x_{k+j}))] = R(x_k, u_k) + \gamma \underset{x_{k+1}}{\mathrm{E}}[V_\pi(x_{k+1})|u_k] \tag{4}$$

where γ is the discount factor of future rewards ($\gamma \leq 1$) and K is the steps of forward inference based on $u_{k+j} = \pi(x_{k+j})$. (4) calculates the value of a policy includes both the direct reward and all future rewards along the action/policy sequence. These future rewards depend on the future states, which are decided by the probabilistic model of the environment. We use the expectation over future states in (4). The discount factor γ is used to weight rewards in the near future more heavily than rewards in the distant future. Theoretically, K might by infinity.

However, we only consider the discounted finite horizon case in this thesis, where $K \ll \infty$ is the potential travelling time to a target.

The aim of the MDP agent is to find the optimal policy

$$\pi^*(x_k) = \underset{\pi}{\operatorname{argmax}} V_\pi(x_k) \tag{5}$$

Combining this with (4) gives

$$V_{\pi^*}(x_k) = \max_{u_k}\left[R(x_k, u_k) + \gamma \underset{x_{k+1}}{\mathrm{E}}[V_{\pi^*}(x_{k+1})|u_k]\right] \tag{6}$$

Many decision-making algorithms find policies through exact or approximate solutions to this equation [16]. In our research, we simplify the evaluation of a path to its length. The other costs, like to change direction multiple times, are not counted.

2.3 From MDP to Partially Observable MDP (POMDP)

An MDP assumes complete knowledge of the environment. However, in a real situation, the robot's on-board sensors only have limited detection distances and their measurements always contain some noises and could be blocked by some obstacles. The robot cannot observe the entire environment exactly or precisely, and the action results will be influenced by these errors and lack of knowledge [33]. It implies that the state of the entire environment is not directly available to the robot. Therefore, plans will be made on the basis of the state of the world [30][14].

In this section, we will introduce the POMDP, which was developed in the operations research community [110]. Many artificial intelligence researchers use it to design and test their algorithms [111]. More formally, a POMDP is defined by the tuple to represent the more realistic world

$$< X, U, R, T, O, \Omega, B, c_0, \gamma > \tag{7}$$

where X, U, T, R, γ are defined to be the same quantities as in the MDP and the new symbols are

- O is the space of observations
- $\Omega(x_{k+1}, u_k, z_{k+1}) = p(z_{k+1}|x_{k+1}, u_k)$ defines observation probabilities
- B is the belief space
- c_0 is the initial information available to the agent

It is assumed that the initial information c_0 takes the form of a (possibly uniform) probability distribution over the state-space. At each step k, the robot receives an observation $o_k \in O$ of the sensor measurements around the robot.

The most prevalent approaches for solving POMDP problems use value iteration, which calculate a value for each node x_k based on the direct reward $R(x_k, u_k)$ of action u_k and the discounted rewards of its children, $\gamma R_{children}(x_k, u_k)$. The main idea is to get the optimal balance between the direct reward $R(x_k, u_k)$ and the future rewards $R_{children}(x_k, u_k)$:

$$V_{\pi^*}(x_k) = \max_{u_k}[R(x_k, u_k) + \gamma R_{children}(x_k, u_k)] \tag{8}$$

2.3.1 Belief States and Belief Space

A robot will update its belief at step k, after taking an action u_k and receiving an observation o_k. Since the state is Markovian, maintaining the POMDP belief over the states only requires knowledge of the previous belief state, the action taken, and the current observation. The belief state transfer operation is $b_{k+1} = \tau(b_k, u_k, o_k)$. The belief state is defined at each step k, so the belief space is also a discrete space. Next, we will describe how to update this POMDP belief.

In state x_k, the agent observes $o_k \in O$ with probability $p(o_k|x_k, u_k)$. Let $b(.)$ to be a probability distribution over the state space X and $b(x_k)$ denotes the probability that the environment is in state $x_k \in X$. Given $b_t(x_k)$ at time t, after taking action u_k and observing o_k, the new belief $b_{t+1}(x_{k+1})$ in new state x_{k+1} is calculated as

$$b_{t+1}(x_{k+1}) = \eta p(o_k|x_k, u_k) \sum_{x_k \in X} T(x_{k+1}|x_k, u_k) b_t(x_k) \tag{9}$$

where $\eta = 1/p(o_k|b_k, u_k)$ is the normalizing constant. Its value is given by $p(o_k|b_k, u_k) = \sum_{x_{k+1} \in X} p(o_k|x_{k+1}, u_k) \sum_{x_k \in X} T(x_{k+1}|x_k, u_k) b_k(x_k)$.

By adding the belief space B, the POMDP problem becomes a belief MDP problem [34]. The action is selected based on the current belief b_k, and the transition function is redefined as the belief transition function. The POMDP optimal action selection policy is constructed as follows:

$$\pi_k^*(b_k) = \underset{u_k}{\operatorname{argmax}} \left[\sum_{x_k \in X} b_k(x_k) R(x_k, u_k) \quad + \gamma \sum_{o_k \in O} p(o_k|b_k, u_k) V_k^*(b_{k+1}) \right] \tag{10}$$

with the expected reward cost:

$$V_k^*(b_{k+1}) = \max_{u_k} \left[\sum_{x_k \in X} b_k(x_k) R(x_k, u_k) \quad + \gamma \sum_{o_k \in O} p(o_k|b_k, u_k) V_{k+1}^*(b_{k+1}) \right] \tag{11}$$

2.4 Review of Planning Algorithms

A POMDP is a special case of Decentralized POMDP (Dec-POMDP), which is suitable for the problem of making a plan for multiple-robots. The algorithms for Dec-POMDP will be introduced in Chapter 5. In this section, we will review the planning algorithms which have been proposed by other authors, including heuristics, policy iteration, forward searching and solutions based on POMDP models.

These proposals handle uncertainties from different points of view. We will use the path planning algorithms as the implementation of these proposals when we need to explain their details clearly.

2.4.1 Value iteration -- Heuristic Approaches

The most common approach (at least for real-time applications such as mobile robot navigation) uses a heuristic function to estimate the value iteration (8) of the full POMDP solution [12]. There are three heuristic categories: (a) those which do not count the uncertainty into the evaluation directly, (b) MDP-based heuristics which consider stochastic actions without future uncertainty, and (c) those which can act in order to resolve or decrease uncertainty for reaching target.

2.4.1.1 Heuristics without Uncertainty Considerations

The simplest strategy is to leave the uncertainty to re-planning, with the assumption that the entire environment is currently known perfectly. A threshold is used to decide which cells are blocked and which cells are free in the representation of the uncertain environment. A path re-planning will be launched when the robot's on-board sensors find an obstacle which blocks the next movement. This strategy is simple and probably the most widely used in practice [12]. By removing the uncertainty with a threshold, we can use Dijkstra's algorithm [99]. Dijkstra finds one of the shortest paths from the start position to its closest neighbours within distance 1. Then, it finds one of the shortest paths to all neighbours within distance 2. By repeating this process, it is guaranteed to find one shortest path to the target position. Dijkstra extends the paths to all possible directions, so it is extremely expensive. To remove those extensions toward the hopeless direction, the A* [38] use a heuristic function to extend the next step toward the target position. The A* algorithm has two overlapped loops to maintain the OpenList and the CloseList. If the map has N grid cells, all grid cells will be added to the OpenList and the CloseList in the worst cases. Therefore, the A* complexity obeys $O(N^2)$.

For some cases (for example, newly-found obstacles block the current path locally), the searching tree of the old plan and the new re-plan have many same nodes and branches. Therefore, Stentz proposed D* [39] and Koening proposed D*-

lite [96] algorithms to keep the A* last search tree for the next planning in order to make the re-plans faster (depends on the similarity of the new searching tree and the old searching tree). We will compare against re-plan strategies in the robot navigation problem experiment in Chapter 3.

If the topological graph of uncertain environment, such as parking lots, is available, the planner could take it into account to decrease the complexity of the searching problem. Dolgov [102] proposed a two-phase algorithm to plan a path for an autonomous car to move in a semi-structured uncertain environment. Firstly, he uses the forward heuristic search to find a continuous kinematical feasible trajectory (the car can move forward, turn left or right; and to move back should be punished) based on the topological lane graph. Secondly, the numerical optimization is used to optimize the trajectory locally, which often gets the global optimum in his scenario. However, his algorithm does not guarantee that the minimal-cost path will be found, or the entire reachable state space will be searched.

The occupancy probability threshold can be used to modify the uncertain environment to a certain environment (the cell with an occupancy probability smaller than the threshold is marked as an empty cell) for the planning algorithms, especially in high dimensional environment. Some researchers use the Rapidly-exploring Random Tree (RRT) to construct the connection structure of the high dimensional environment based on many sampled nodes and make a fast planning in the tree [41]. The new nodes are sampled and connected with the RRT tree to extend the size of the tree. For example, Pepy [40] proposed a strategy to transfer uncertainty to certainty by using a threshold to sample RRT nodes for where the robot can go and where it cannot go. Jaillet [41] considers the cost function defined over the environment and uses a threshold in the transition tests to accept or reject the new sampled nodes of an RRT tree. The path is searched in the RRT tree instead of the C-space. But, the RRT may not find the path when a path exists in fact [41].

Unlike the threshold strategy, the Most Likely State (MLS) uses the most likely state (a probability matrix defined by user) to decide the actual states of the world of highest probability, and the shortest path is found with these states. MLS takes the corresponding action from the MDP policy [42]. This is a good approximation to the full POMDP solution when distributions are compact, and the most likely state is hardly far from the truth.

The Q_{MDP} method [112] votes on actions based on the belief distributions over discrete states of the uncertain environment, based on the evaluations of all state nodes [43]. The number of votes state x_i can cast is proportional to the probability that x_i is the true state. x_i casts its allotted votes by voting on actions in proportion to their MDP evaluations from state x_i. After voting, the agent will take the action with the most votes, which is a greedy choice. The main assumption for Q_{MDP} is that all the uncertainty will disappear after the robot takes its action. The action will be optimal if this assumption is true, otherwise the uncertain states may make the action not executable [44]. The MLS and Q_{MDP} are suitable for small discrete state-spaces (about several hundred discrete states in 2 or 3 dimensional state space).

2.4.1.2 Heuristics to Resolve Uncertainty

The problem with the heuristics above is that they only act to seek a reward, possibly taking into consideration their current uncertainty and the uncertainty of their actions [12]. Another strategy is to decrease the uncertainty for the future re-planning. This means that the uncertainty should be counted in the heuristic function to compare the effect of the uncertainties among different selections.

Action entropy is an example of a heuristic which is used to quantify the change of the uncertainty after the actions are executed [44]. The entropy is defined by information theory as $-p\log p$ [97], where $p = p(m)$ is the occupancy probability of a cell m in an OC map. The action based on entropy switches between two distinct modes: seeking a reward and seeking information. The entropy distributed in belief space can be used as the switching criterion. The entropy planner can select the action to reduce the belief uncertainty over a one-step horizon. In other situations, it follows one of the MDP-based heuristics. A typical example is the algorithm proposed by Bruns [18], which uses entropy to measure the information obtained for guiding a motion planner to achieve maximal progress toward the discovery of a path. It uses the entropy to redefine the edge costs among the sampled nodes and guide the A* searching process. The entropy can measure the uncertainty changes within the robot observation distance, which only provides a local measure of uncertainty and cannot avoid the local minimal problem. Therefore, the global view of the entire uncertain environment will hardly be calculated by entropy.

Thrun [45] plans a path by considering the quality of localization along that path. His algorithm is called Coastal Navigation, which calculates the information content of each state, based on the extent to which an observation from that state would modify a fixed priority. Then, each state gets a value combining a weighted sum of the information-based cost and a target-related cost. The planner can use this value to evaluate and compare different paths.

Candido [98] proposes a minimum uncertainty planning technique for mobile robots localizing with beacons based on a POMDP model. His algorithm analyses the evolution of the belief based on the pre-known problem-specific domain knowledge. The core idea of Candido's method is to use paths (such as nearly optimal trajectories) prescribed in the workspace as a heuristic, then to generate the belief-feedback policies with fixed modes and apply them to robot navigation localizing with beacons to find a minimum uncertainty path [98]. However, his algorithm requires other algorithms (which he does not specify) to help generate the explore policies in the belief space of POMDP. The explore policies will affect the performance of the planned result.

A number of authors have proposed different ideas to add uncertainty evaluations into the basic RRT algorithms. Tian [19] and Liu [46] presented a modified RRT algorithm for robot local navigation, which provides a biased direction for nodes to surround obstacles, which can remove the obstacle uncertainties from an uncertain environment. The planner can use these biased nodes to find a path toward the target and avoid the uncertain obstacles. However, the planned path cannot be guaranteed to be the shortest one.

Maeda [47] uses a mixture of a Model Predictive Control (MPC) for local RRT planning with an approximate-model global planner to provide sub-goals. Guibas [48] introduces the notion of a bounded uncertainty roadmap (BURM) and uses it to evaluate uncertainty by representing the collision probability bounds in different regions of the C-space. The Multipartite RRT (MPRRT) designed by Zucker [49] for fast re-planning a path in unknown or dynamic environments. It biases the sampling nodes toward the separated RRT sub-trees and re-using nodes/branches from previous planning iterations. Melchior [50] extends the single searching node with several possible nodes calculated from multiple simulations. The algorithms have to know the possible models of the coefficient of friction (different fictions make the robot move to different directions) in the given rough terrain, in order to simulate the robot movement in the unknown environment. Fulgenzi [51] proposed an algorithm based on the extension of RRT, where the likelihood of the obstacles trajectory and the probability of collision are explicitly taken into account, based on the on-board sensor measurements. His algorithm refines the path when the RRT tree is updated and extended as the robot moves (the local path is re-planned with high frequency). Pepy [52] proposed a new path planner by combining RRT nodes with a set representation of uncertain states. The uncertain state increases the number of RRT nodes as the planned path expanding toward the target.

2.4.1.3 Evaluate Uncertainty with Policy Iteration

Policy iteration seeks to evaluate the candidate policy or produce a modified policy, instead of evaluating the sequence of actions and states. By given each policy a value, which can be extracted from action-based approaches, the policies can be evaluated explicitly. There is an equivalence between the two approaches: where value iteration extracts a policy from a converged value function, policy iteration calculates a value function from a policy during each policy evaluation step [12].

Hansen shows how policies can be represented in Finite State Machines (FSMs) [53]. Each node in the FSM dictates a particular action, while each arc corresponds to a particular observation. Each step of policy improvement involves modifying the FSM by adding and removing nodes, and changing the actions associated with nodes (which changes the successor nodes associated with observations). Modifications are based on exact updates, hence convergence is guaranteed.

The path of belief nodes corresponds to the path of the policy nodes, when we transfer the belief nodes structure to the policy nodes structure. Therefore, we can find the path based on the policy nodes structure. However, policy iteration is another form of value iteration and cannot avoid the shortages of the value iteration algorithms [53].

2.4.1.4 Forward Search

The value-iteration-based approaches assume that the estimates of the value function evaluated in the future are available. These estimates are used to create a value function in order to transfer the discounted future value to the present value. The calculation of future values is generally defined over the entire belief-space, which can be pre-calculated in advance. If we hypothesise that the value iteration calculate the nodes' values at current time t, the forward searching calculates the nodes' values at time $t + \Delta t$ (target may be not reached at $t + \Delta t$, but the planner assumes the robot will not be blocked from t to $t + \Delta t$).

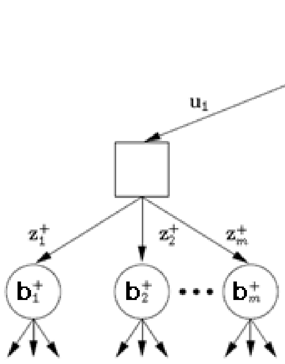


Fig. 9 A POMDP as a belief node and action tree, starting from belief b [30].

A number of POMDP solution algorithms use the search forwards strategy, starting from the current node [54][55][56]. A POMDP can be viewed as a game which alternates between the agent selecting an action and nature selecting an observation. Other agents' observations and actions will be estimated for Dec-POMDPs. A traditional approach to maximizing performance in turn-based games is to represent the game as a tree [30].

In the example shown in Fig. 9, we use circles to represent nodes from which the agent chooses a value-maximizing action from all possible next steps. The squares represent nodes from which the environment probabilistically chooses one observation. The value of each node is based on the rewards associated with belief-action transitions and the estimated values of the unexpanded leaf nodes. An action node value involves a maximum over the values of its children, whereas an observation node value involves an expect observation. Given a heuristic to estimate the values of the unexpanded leaf nodes of the tree, a naive approach to solving the POMDP is to perform brute-force search of this tree, expanding every action in a breadth-first order. Since the game-tree is valid only for those reachable belief nodes from a known starting belief node, this approach is often used for online computation.

Forward search can adapt to changes in the environment as long as the model is also updated [57]. For discrete actions and observations, a value function does not have to be represented over the entire belief space, because the reachable belief nodes are sufficient for decision-making.

The limitation of forward search is the exponential complexity in the planning horizon, at least for naïve breadth-first search. If each node has n children at each step, k-steps forward will generate $1+n+n^2+n^3+\cdots+n^k=\frac{n^{k+1}-1}{2}$ nodes. This approach is not suitable for making long-term plans. The computational complexity can be further reduced by scaling down the number of samples for calculations further down the tree, which has less effect on the topmost values due to the discount factor γ [12].

A number of authors reduce the computational complexity by expanding actions in a more narrow sense, using a search algorithm such as AO* [58], where the nodes are divided as “and-node” (combine the values of children nodes) and “or-node” (only the highest value among children nodes are kept) in order to calculate their values and propagate the change to parent nodes. For AO* to be effective, a good heuristic, for example the heuristic function designed for some special problem [57], is required to estimate the value of un-expanded nodes. In our research scenario, the robot only has an occupancy cell map, and the obstacles or damages are not known exactly. As a result, a good heuristic is unavailable for general situations.

These uncertainty-aware heuristics are an improvement over simpler heuristics, but also have some shortcomings [12]. Each method makes the optimistic assumption when it makes a plan, and expects no re-plan happens again. In fact, the path of plan/re-plans in uncertain environments is not same as the robot actually travelled route. The re-plan may be launched at any waypoints in the planned path. The actually travelled route is a combination of the partway of plan/re-plans, which means that the planner only evaluate part of the travelled path, not the entire travelled path.

2.4.2 Graphical POMDP Models

POMDPs are often described using graphical models [59]. An example is shown in Fig. 10: x is the state node, o is the observation node, I is the decision node (where we select an action); the shaded nodes are observable, while the un-shaded state nodes are hidden; the rewards are omitted for clarity; the graphical POMDP model shows how a joint probability distribution over all states, actions, observations, and the decision nodes can be factored into smaller conditional probability distributions [59]. This section reviews the authors' works based on this type of model.

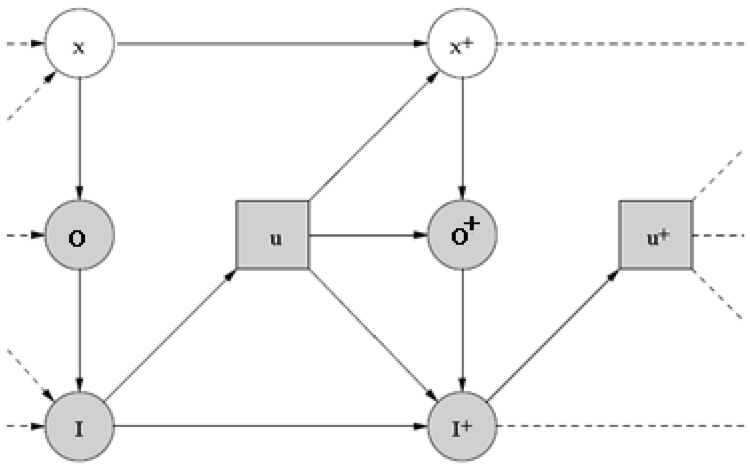


Fig. 10 Two time-slices of the POMDP problem, using a graphical model [59].

Inference in graphical models is often achieved by fixing the values of certain nodes (usually the observable variables), and then applying an inference algorithm to determine distributions over variables of interest. Attias proposes a novel approach using general graphical model theory to solve POMDPs [60]. Actions are written as random variables. If the maximum depth of the graphic searching tree is N time-steps, the N-th state is fixed to be the target state, and the first observation is fixed. Assuming a prior distribution over the (assumed random) action variables, standard inference algorithms can then be applied to find the Maximum A Priori (MAP) sequence of actions [60]. However, the prior action distribution may not be available in the real world to find the actions. One possible extension is to replace a single target state with some general reward function, which can be calculated in the planning process. By casting the problem as an inference problem in a graphical model, a powerful and general inference algorithm can be designed based on graphical search algorithms [60]. This planning process extends the searching inferences to all possible directions, and the leaf nodes provide the basic reward values for these inferences. In the real world, the maximum depth of the searching tree cannot easily satisfy both the shortest solution and the calculation limitation.

2.4.3 Node Clustering Algorithms

Node clustering in POMDP spaces is discussed in many papers, which aim to decrease the number of nodes in the belief space [61][73]. Node clustering algorithms make decisions with fewer belief nodes, by putting the main calculation on decreasing the degree or dimension of the belief space. Eventually, the solution in decreased belief space will be transferred back to the original belief space for the agent to execute. Its computational complexity obeys $O(ndke)$, where n is the node number before clustering, d is the dimensionality of each point, k is the number of clusters, and e is the number of iterations required for clustering [113].

Li [61] proposes an algorithm for off-line planning by clustering belief states based on the Euclidean distance between two nodes in the belief space. He proposes a belief space compression and node clustering algorithm in [62]. Rens [63] proposed an algorithm for online planning by clustering the belief nodes based on the states' distance in POMDP state space. Oliehoek [64] identified a criterion that allows for lossless clustering of observation histories (cluster more nodes with longer distance), and speed-up the planning process (empirically, at least 42% faster for horizon $h = 4$).

These clustering algorithms for POMDPs can be used in other POMDP approaches. One example is for heuristic algorithms, proposed by the authors in [65][66][67], which focus on the belief search space and eliminate searching branches over unreachable or improbable belief states. Another example is for POMDP compression algorithms by clustering nodes, proposed by the authors in [68][69][70]. They construct a policy space on the lower dimensional belief space, which is created by the belief nodes clustering operations. Then, they make a planning in the policy space and 'reinstate' the policies for the original belief nodes. The degree that a belief space can be compressed depends on the redundant space of the problem model. If the belief space is compressed too small, the result will be inaccurate. If we want an accurate result, the compressed belief space should have many belief nodes. It is not easy to define a compress model for general situations. Therefore, these authors define the compress operations based on the POMDP model of different problems.

This kind algorithm provides a research direction to decrease the computational cost of POMDP algorithms. We will propose a strategy based on this idea to reduce the computational cost of our proposal in Chapter 4.

2.5 Discussion

In this chapter, we introduced the problem of visiting targets as quickly as possible to locate missing survivors in an uncertain environment. The targets are given by the rescue team members and a group of robot will be sent to visit them.

We introduced the basic concepts and terminology of the MDP and the POMDP, which can be used to represent our research problem. The current mathematical models, strategies and algorithms proposed by different authors to solve the POMDP problems have been introduced in this chapter. These algorithms compare different paths/solutions and find the best one among them based on the representation of the uncertain environment.

However, the single shortest solution does not guarantee that the actual path travelled by the robot is the shortest, because the later plan may deny the current shortest plan. For example, the robot is moving toward the west along the current plan, but the later re-plan has a high probability to ask the robot to move back (move toward the east), because the west direction is blocked completely by new found obstacles. Then, the robot actual travelled route in CMP is far away to be the shortest route.

Therefore, these algorithms have one important limitation: they only compare the single planned path/solution, but ignore the effect of the re-plans in CMP. In uncertain environments, the evaluation cannot avoid unexpected obstacles (which will block the planned path) completely. The robots have to make several re-plans at some random positions in CMP, until they reach the target. The total time to complete the mission includes the total travelling time and the total planning time. The routes that the robots actually travelled are a combination of the partway of re-plans, which are different with the planned path in plan/re-plans. These routes decide the total travelling time. More re-plans increase the total re-planning time of the complete mission time. Therefore, the planner should make a trade-off between the total travelled route and the possible re-plan numbers in CMP. We have to add evaluation of the path length and the re-plans into the planning process until the problem is solved.

The difference in our research is that we seek the shortest path/solution for the entire CMP. On one hand, we want the route that the robot actually travelled to be the shortest. On the other hand, we want the lowest re-plan probability. We find that if the possible shortest paths are clustering in some areas, we can find the path with the optimal trade-off between the path length and the re-plan probability.

In the next chapter, we introduce the PD Planner.

3 Path Planning in the Presence of Uncertainty

This chapter proposes an algorithm to solve the problem of path planning in an uncertain environment in the following scenarios:

1, We only have one robot and one target location (given by the rescue team).

2, One robot will move from its start position to target position in an uncertain environment. It may take many re-plans before the target position is reached.

The task is to make the robots move through the uncertain environment efficiently. Therefore, we should add the effects of uncertainties and the possible re-plans into the planning process, when we consider this planning problem.

3.1 Introduction

Our research problem is simplified as: how to plan a path in OC map M_{oc} for robot to reach its target, as illustrated in Fig. 11. We assume that the robot moves at a constant speed, in order to describe the paths more conventiently and compare them exactly based on their grid cells' coordinates.

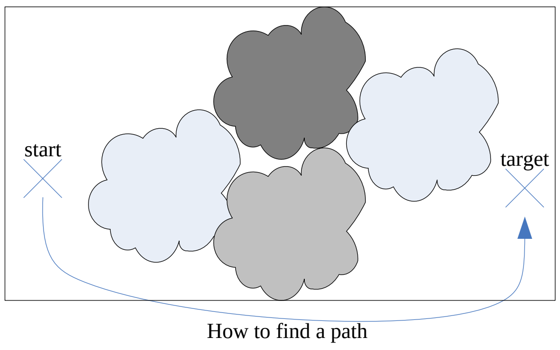


Fig. 11 Path Planning problem in an uncertain environment for a robot.

The path is planned in the OC map M_{oc} for the robot to reach its target cell. The clustering value of the belief node is the path distribution value in PD Planner algorithm, which will be introduced in the following section.

We use 2-dimensional environment to illustrate the planning algorithm. However, the algorithm can also work in 3- dimensional and higher dimensional environment.

The paths are evaluated in terms of the path length in our research, so the optimal path has the same meaning with the shortest path.

3.2 Path Planning in Known Environments

A* is one of the most widely used path planning algorithms to plan a path in known environments [38], which uses a heuristic to estimate the future cells in the searching tree. Based on the cell heuristic value given by the heuristic function, the cells with the shortest estimation length to target will be searched first, like Fig. 12.

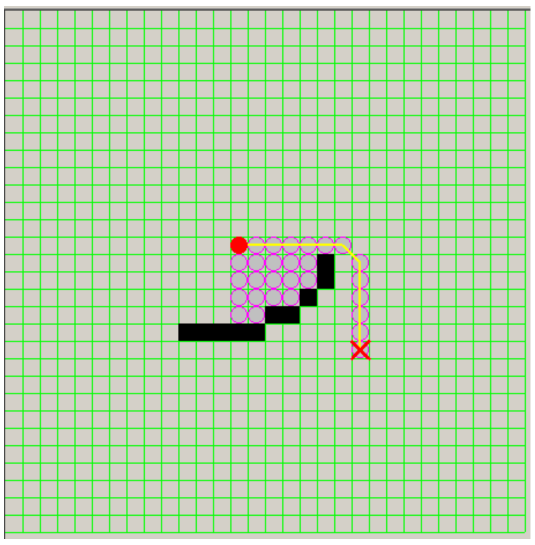


Fig. 12 A* algorithm searched area (cycles) and planned path (yellow line).

The A* algorithm defines the path estimate length function $f(m_{x,y})$ as follows:

$$f(m_{x,y}) = g(m_{x,y}) + h(m_{x,y}) \tag{12}$$

where $m_{x,y}$ is a cell with coordination of (x, y), $g(m_{x,y})$ is the actual cost of the shortest path from the start cell to cell $m_{x,y}$, and $h(m_{x,y})$ is the actual cost of the shortest path from cell $m_{x,y}$ to the target cell [74]. The A* searching process can find the minimal value of $g(m_{x,y})$ by comparing all paths between start cell and cell $m_{x,y}$.

However, we use $\hat{h}(m_{x,y})$ to estimate $h(m_{x,y})$ in reality, because we do not know $h(m_{x,y})$ before the searching process reach the cell $m_{x,y}$. $\hat{h}(m_{x,y})$ is called as the heuristic function. Hart [74] proved that, if $\hat{h}(m_{x,y})$ is a lower bound on $h(m_{x,y})$, the shortest path is guaranteed to be found. Based on the A* value function, the shortest path has the lowest value $f(m_{x,y})$.

By choosing the next cell $m_{x,y}$ with the lowest value $f(m_{x,y})$ in the searching process, the most likely cells to reach the target cell are searched first. This is the reason why few cells are searched in Fig. 12.

However, the performance of A* heavily depends on the heuristic function $\hat{h}(m_{x,y})$, which will add additional calculation to the searching process in order to extend the searching tree to the most hopeful directions of reaching the target. If the heuristic function is guaranteed to be less than, or equal to, k times the actual distance to the target cell (k is a real number), the found path is guaranteed to be no more than k times longer than the actual shortest path [38]. Here, A* algorithm requires $k < 1$. If it is not very important to find the actual shortest path, we can set $k \in [1,2]$ depending on the requirements and the pre-known conditions of different approaches [38]. In the worst case, the searching tree will cover all cells of the map, which means that the heuristic function cannot decrease the size of the searching tree. Generally, researchers define different heuristic functions for different applications, in order to effectively reduce the size of the searching tree and make the searching process faster [38]. For example, Buniyamin discussed about the Euclidean distance heuristic function in [103], and Dolgov proposed a hybrid A* in [104] to combine the non-holonomic heuristic and the 2D Euclidean distance heuristic to goal. In our path planner, the Euclidean distance heuristic function is used as A* heuristic. The reason is that our algorithm is suitable for 2D and higher dimensional environments.

The pseudocode for the A* algorithm, with distance heuristic function, is shown in Pseudo-code 1.

Pseudo-code 1 A* algorithm

```
AStarFun(GraphNodes:V, start:$u_{start}$, target:$u_{target}$)
1  for all u∈V
2      g(u)←∞
3              h(u)←heuristic function estimate distance between u and $u_{target}$
4  end for
5  g($u_{start}$)←0;    and add $u_{start}$ into OpenList
7  while (OpenList !=Φ)
8      u←minimal ( g(u)+h(u) ) in OpenList
9      remove u from OpenList and add u to CloseList
10     for all neighbors v of u
11         if (v == $u_{target}$)
12             bp(v)←u            //back pointer is set to point u
13             break while;
14         end if
15         if (v∈CloseList) and (g(u)+c(u,v) < g(v)) then
16           g(v)←g(u)+c(u,v)
17           bp(v)←u
18           if (v∈OpenList)
19               update g(v)
20           else
21               add v into OpenList
22           end if
23         end if
24     end for
25 end while
26 return the path following the back pointer bp($u_{target}$)
```

The pseudo code above uses u and v to denote graph nodes, which is the grid cell $m_{x,y}$ in a certain map. The D* [39] or D*-lite [96] algorithms reuse the former A* searching tree to re-plan a new path for a robot to move in an uncertain environment.

3.2.1 Rapidly-exploring Random Tree (RRT)

A* is not suitable for solving the high dimensional space path planning problem in a short time, because its searching step is short and there are still many nodes in the search tree. So, researchers proposed the RRT algorithm to construct a search tree with less nodes and longer distance between two neighbour nodes. RRT algorithms are designed for searching high dimensional space fast. The same as A*, it can be used to solve MDP problems.

RRT uses the robot start position as the root, and expands the branches toward the target until the target is added into the tree, or the limitation (like the max node number) is reached. A typical RRT search tree looks like Fig. 13, where the black blocks are obstacles and each node indicates a small collision-free area around it. Because each node is randomly sampled, many places in the map are not covered by the tree. In A*, we extend nodes from the current node to its near-by neighbours in the grid cell map. But in RRT, we sample nodes from the original grid cell map, and set up their connection (generally connect to the nearest nodes) to construct a roadmap tree. Then, A* will search through this roadmap tree and find a solution. The researchers focus on how to sample and construct this tree effectively. For example, increase the number of sampled nodes around obstacles and use different length limitation among sampling points.

The main advantage of the RRT is that the nodes of the search tree are less and it can be applied to a high-dimensional space. However, it has one important shortage: there might be a path existing, when the RRT algorithms find “no path existing”.

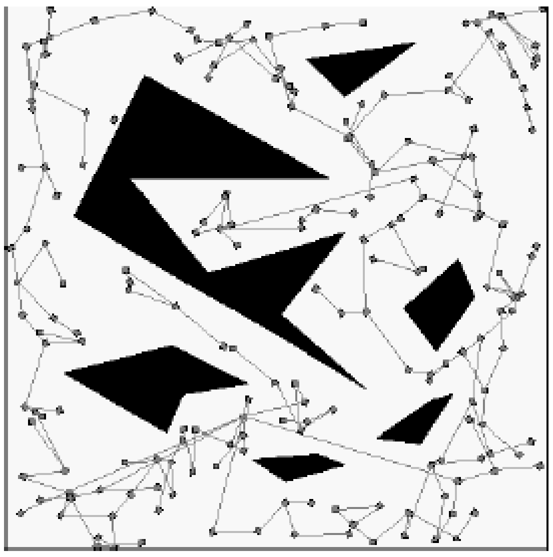


Fig. 13 An example of RRT: random nodes and their links [75].

3.3 The Path Distribution (PD) Map

In this section, we will introduce an algorithm to find a path for the robot in an OC map.

3.3.1 Definition and calculation

In our research, we find that the possible shortest paths in an OC Map tend to cluster in some areas. Therefore, we cluster these possible shortest paths to grid cells. We define the Path Distribution (PD) Map to record this cluster.

Given the occupancy cell map M_{oc}, we randomly draw a large number of samples of map instances M^i. The shortest and collision-free path $l_i(M^i)$ is constructed on M^i, which is computed using a suitable path planning algorithm in a certain environment. The path distribution is the probability of a possible shortest path going through a cell $m_{x,y}$, written as $p_d(m_{x,y})$. Marginalizing over the map instances, its value is given by

$$p_d(m_{x,y}) = \sum_{M^i \in M} \{p(m_{x,y} \in l_i(M^i) | M^i) P(M^i)\} \tag{13}$$

where $P(M^i)$ is the probability that map M^i is the actual description of the environment. In our research, we use the A* path planning algorithm to compute the path given a map realization. However, any algorithm that can find the shortest path can be used here.

A* is a deterministic path planning algorithm, and we have

$$p(m_{x,y} \in l_i(M^i) | M^i) = \begin{cases} 1 & when\ m_{x,y}\ lies\ on\ l_i(M^i) \\ 0 & otherwise \end{cases} \tag{14}$$

The PD Map is estimated using the Monte Carlo Sample strategy, as introduced in Chapter 3. We compute the probability in a similar manner to that proposed by Murphy [76]. A set of possible paths $S = \{l_1, l_2, \dots, l_k\}$ where $k \leq 2^{|M|}$. $p_d(m_{x,y})$ is evaluated by summing the number of paths which pass through each cell, then dividing by k. The pseudo code of the algorithm is as shown in Pseudo-code 2.

Pseudo-code 2 Compute Path distribution function

```
ComputePathDistribution(OC map:$M_{oc}$, StartCell:$m_s$, TargetCell:$m_t$,Samples)
1   Md = init the path distribution map
2   for(i=0; i<Samples; i++)
3       $M^i$=RandomMap($M_{oc}$)  //cell state is based on its occupancy probability
4       $l_i$ = AStarPathPlanner($M^i$,$m_s$,$m_t$)
5       record a new path goes through the cells of $l_i$ in map Md
6   end for
7   cells' path counts in Md is divided by s (get the probability of lying on a path)
8   return Md
9 End function
```

The PD map records where the POSs are likely to cluster. It is different with the Kearns' expectation map [56], which records the estimated travel costs in an uncertain environment.

In PD Planner, we use A* to find the shortest path on M^i. If the algorithm, like RRT, would like to find the sub-shortest path and cannot guarantee to find the shortest path on M^i, the path clustering may not be shown clearly in PD Map. The examples of PD Maps based on A* will be shown in the next section.

3.3.2 Examples of PD map

In this section, we illustrate the PD map examples based on the OC map and different start and target position pairs. An example of OC Map is shown in Fig. 15, where there are many corridors and the darker cells have higher probabilities of being occupied, which increase the uncertainty of the corridor. This map is based on the floor plan of the Yale Law School 1st Lefel Plan Map [77] (see Fig. 14), because it is a challenging example of a map in which there are many narrow corridors and alternative routes. We use the image convolution blur matrix to make the grid cells' states vague, which can be used to simulate damages. This operation is introduced in [78][79], which can generate the effects of shaking the building to create the OC map. The grid cells closer to the known wall will have the higher occupancy probability. The pseudo code is shown in Pseudo-code 3.

Pseudo-code 3 The Image Convolution function

ImageConvolution(image: input_image; image: output_image)

```
1  for each image row in output image:
2    for each pixel in image row:
3       set accumulator to zero
4       for each kernel row in kernel:
5         for each element in kernel row:
6           if element position  corresponding* to pixel position then
7             multiply element value  corresponding* to pixel value
8             add result to accumulator
9           end_if
10      end_for (element and kernel)
11      set output image pixel to accumulator
12  end_for (pixel and image row)
13  output new image
```

To simulate an uncertainty environment, the floor plan was convolved with a kernel, and all cells' occupancy probabilities are bigger or equal than 0.1. Furthermore, deliberate regions of high uncertainty (increase to 0.3) were inserted to simulate where existing routes could be blocked. These cells were placed in the channel around coordinate (120,120), Fig. 15.

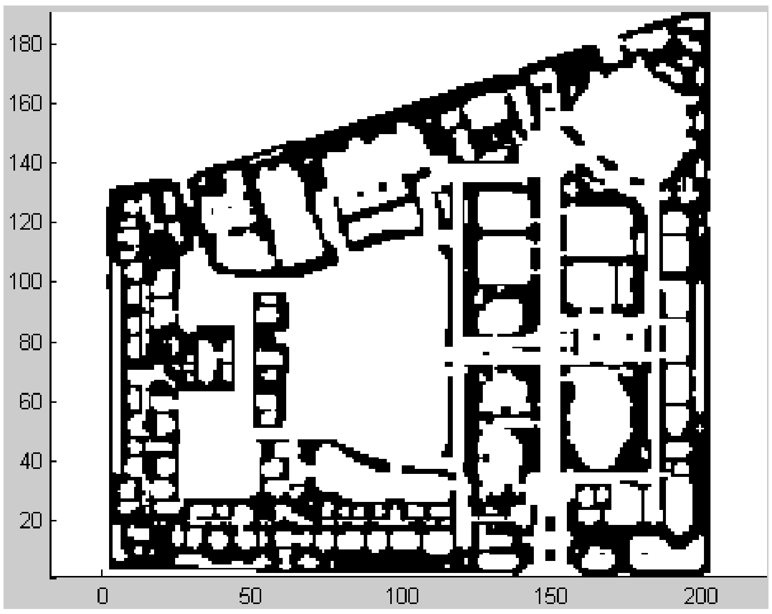


Fig. 14 The building plan map of Yale Law School 1st Lefel [77].

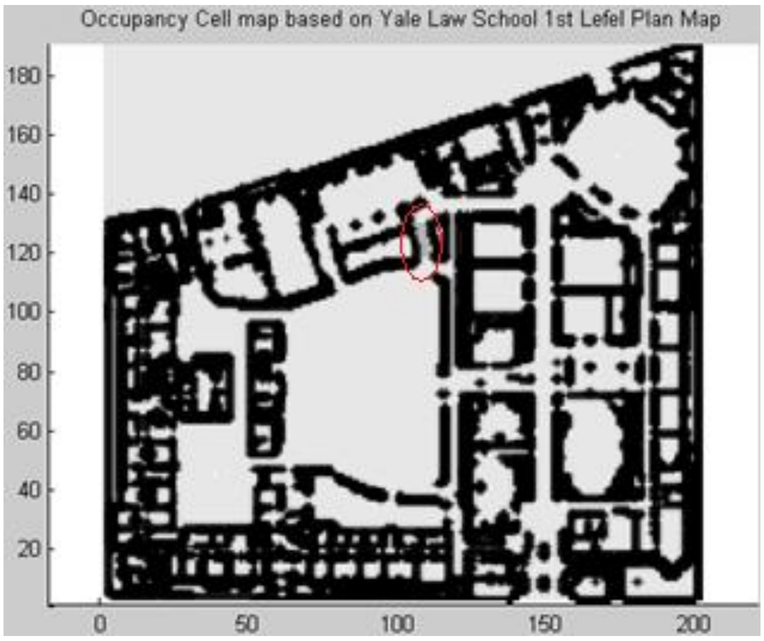
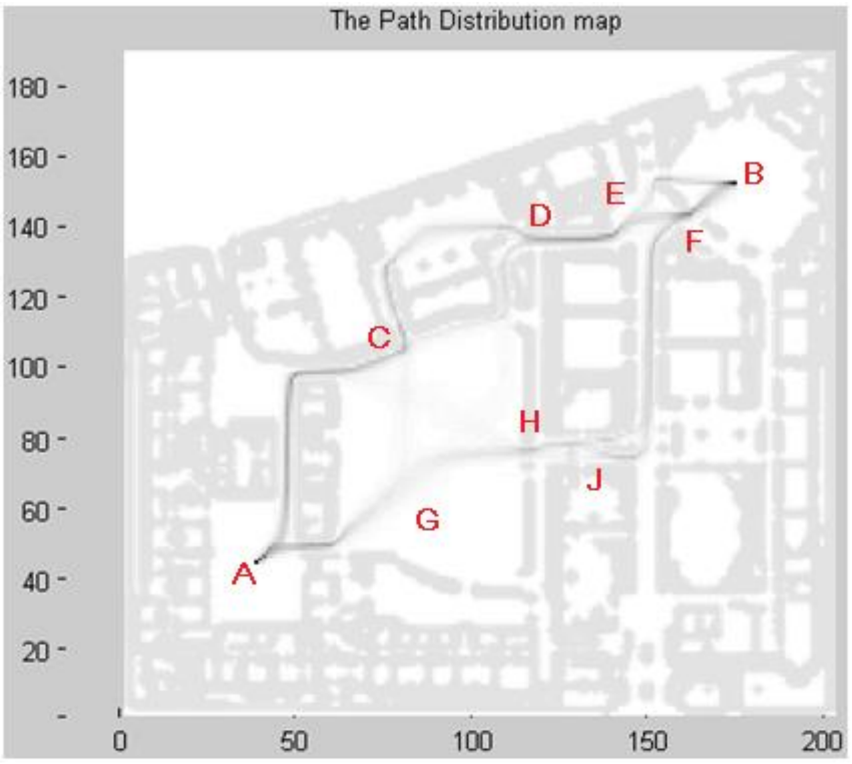
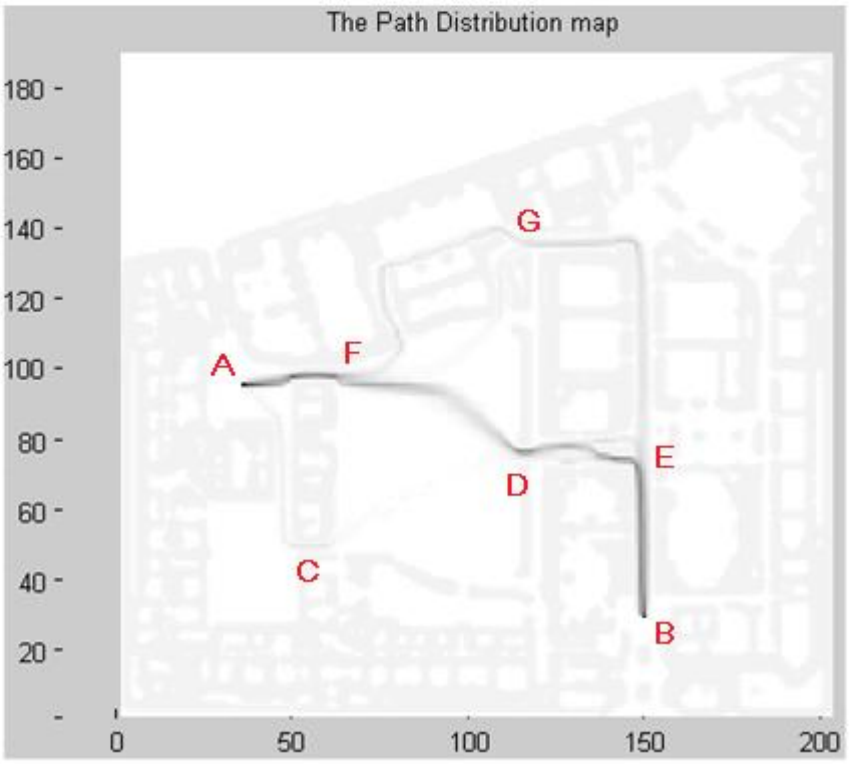

Occupancy Cell map based on Yale Law School 1st Lefel Plan Map
180
160
140
120
100
80
60
40
20
0
50
100
150
200

Fig. 15 The blurred OC map of Fig. 14, with a high occupancy area(red ellipse)

For each algorithm of A* and RRT, we demonstrate two PD Map cases with different start and target position, as shown in Fig. 16 and Fig. 17. We could see that the clusters in the RRT (1000 tree nodes) PD Map are more loose compared with the A* PD Map. The background in these two figures is the building map. The darker cells have higher probabilities p to lie on a path. In the centre area, figure (a) has more branches than figure (b) in Fig. 16 and Fig. 17. We use capital letters to mark the key positions of the branches.

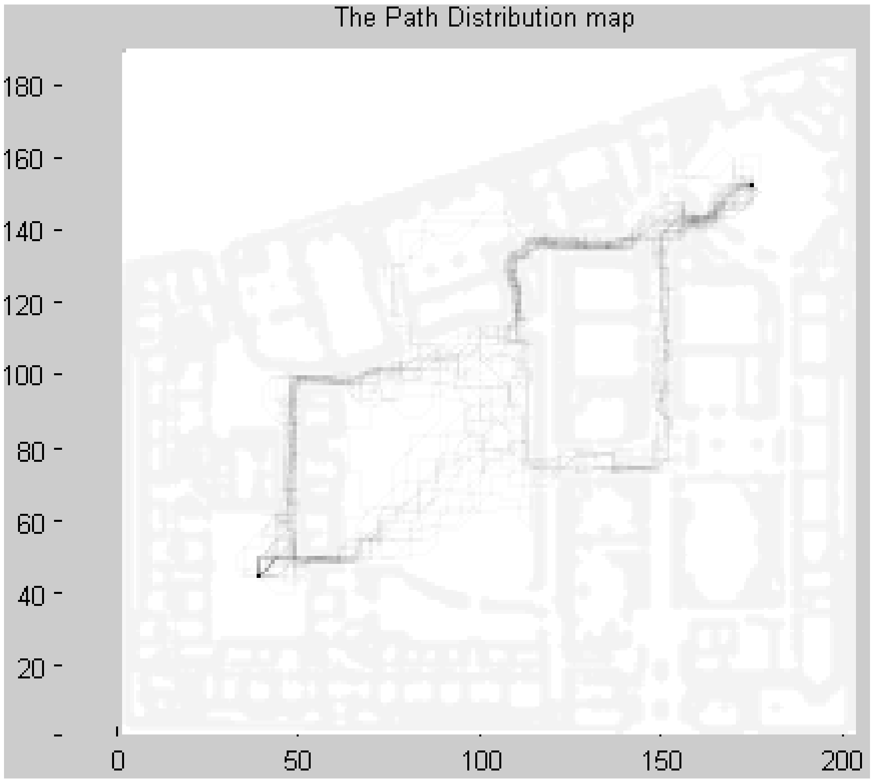


(a)

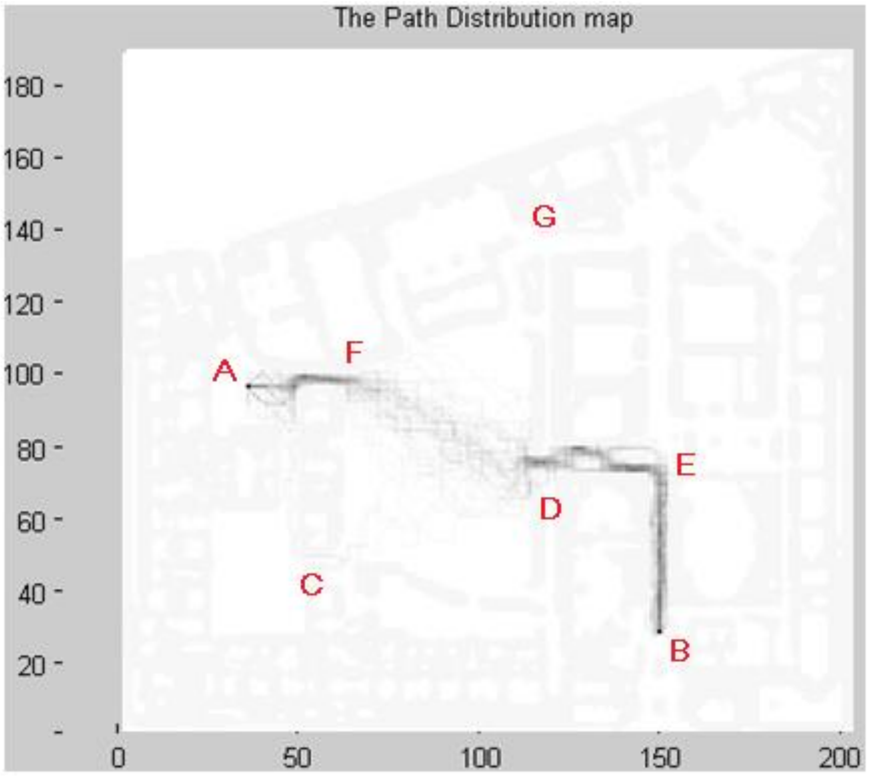


(b)

Fig. 16 The estimated path distributions based on 3000 map instances (A* paths).



(a)



(b)

Fig. 17 The estimated path distributions based on 3000 map samples(RRT paths)

It is clear that there is a natural clustering of several alternative routes, in Fig. 16 and Fig. 17. For example, in Fig. 16 (a), the paths between A and B are mainly clustering to A-C-D-E-B and A-G-H-J-F-B, the main branches appeared at C-D, E-B, J, and the low clustering routes locate among points C-G-H. There may be two main reasons that the wide area C-G-H shows less clustering: 1) uncertain obstacles construct many corridors to separate paths into different locations; 2) the shortest paths in this area are too sensitive to cluster in a small area. These indicate that the robot passing through the area C-G-H will jump among different possible shortest paths and increase the actual travelled route length.

The paths planned by A* and RRT have similar clustering phenomena, but RRT loses some branches (so we prefer A* in our planner). For example, the clustering routes F-G-E is found by A* in Fig. 16(b) but cannot be found by RRT in Fig. 17(b). The clustering route F-D is clearer in Fig. 16(b) than in Fig. 17(b). The RRT cannot guarantee to find the shortest path in these randomly sampled certain environments. Therefore, the lack of shortest paths leads to lack of clustering branches. These pictures show the strong structure of clustering routes which typically exists in these environments. The cluster in PD map will guide us to plan a path for the robot.

3.4 Planning the Path with the PD Map

A known obstacle marked in a map can be avoided by a path planner. However, the obstacles' positions are not known exactly when the environment is uncertain. Therefore, the obstacles are not always avoidable by the path planner. Based on the PD Map, we will find the path in the high clustering areas to avoid obstacles and reach the target by travelling the shortest route.

Given a belief state b, its child belief states (direct neighbours) are $\{b_1, b_2, \dots, b_n\}$. The child belief state $b_i = \{(x, y)_i\}$ corresponds to cell coordinate (x, y). One choice would be to use the path distribution probability of the cell $p_d(m_{(x,y)_i})$ as the reward value of the state. Then, at each grid cell (belief node), we can simply use the greedy algorithm to select the action which leads to the child state with the highest reward among children states. However, this greedy decision, like algorithm Q_{MDP} [43], will not generate the globally shortest solution in CMP. Therefore, we use the wide-first search algorithm to plan a path for the robot.

We call the algorithm *PD Planner*. The *PD Planner* seeks the path, l, that connects the start cell and the target cell by getting the maximum evaluation among the candidate paths. More formally, the goal is to find the path l such that

$$l^* = \underset{l}{\operatorname{argmax}}(p_d(l)) \tag{15}$$

where $p_d(l)$ is the distribution of the possible shortest paths along the path l, defined to be:

$$p_d(l) = p_d(l_{x,y}^1)p_d(l_{x,y}^2) \ldots p_d(l_{x,y}^l) \tag{16}$$

Because $p_d(l_{x,y}^1) \leq 1$, in general the longer the path l is, the smaller $p_d(l)$ will be. For numerical stability we find the path which minimizes the negative log likelihood

$$l^* = \underset{l}{\operatorname{argmin}}\left\{-\sum_{i=1}^{w} \log\left(p_d(m_{x,y}^i)\right)\right\} \tag{17}$$

The pseudo-code for the searching process of the PD Planner is shown in Pseudo-code 4.

Pseudo-code 4 Path Planning function based on the Path Distribution Map

```
PlanPathFunction(OC Map: $M_{oc}$, Start Cell: $m_s$, Target Cell: $m_t$)
1     if ($m_s$ == $m_t$)
2         return no path
3     end if
4     integer: Samples        //the number of samples to estimate PD map
5     Md = ComputePathDistribution($M_{oc}$, $m_s$, $m_t$, Samples)
6     if (all cells are zero in Md)
7         return no path
8     end if
9     Md = -log(Md)     //each cell's value is transferred to -log(value)
10    Init paths set S as empty set        //all paths in the searching process
11    S <= a path $l$ with only one cell $m_s$
12    while($m_t$ is not reached)
13        $l_{cur}$= the path with minimum sum of Md(the path's cells) in S
14        remove $l_{cur}$ from S
15        Init NewPathsSet as empty set
16        for(i=0; i<8; i++)        //try 8 directions for next step
17            $l_{new}$ = extend $l_{cur}$ one step on the i^th direction
18            if (no cells of $l_{new}$ is revisited by $l_{new}$)
19                NewPathsSet <= $l_{new}$
20            end if
21        end for
22        S <= NewPathsSet
23    end while
24    return the path $l$ (with the lowest sum in S and its last cell is $m_t$)
25 End function
```

The searching process evaluates the possible branches and extends those most hopeful branches at each step until the target is reached. This is a typical width-first search algorithm, like the Dijkstra algorithm based on PD Map. In the PD Map, the path distribution value can be seen as the travelled cost from start cell. Therefore, this searching process differs from the A* algorithm, which uses $f = g + h$, where g is the travelled cost from start and h is the estimated heuristic value to target (g and h are different for most cases).

For estimating PD Map, the PD Planner will sample s map instances based on the OC Map. Therefore, the computation cost of the PD Planner is about s times of the A* computational cost. We will discuss more details of the computational cost of PD Planner in Chapter 4.

3.5 Experiment results of different path planning algorithms

The path planned by PD Planner lies in the high clustering areas of the possible shortest paths. The possible shortest paths clustering in these areas could be the candidate paths for possible re-plans. This is why the PD Planner can provide a better solution for the Complete Mission Process (CMP).

In the experiment, we compare three different path planning approaches, which show three typical ways to deal with the uncertainty. These algorithms are: the Threshold A* path planning algorithm [38], the entropy-based path planning algorithm [18], and the PD Planner.

3.5.1 Additional Descriptions in Simulator

In this section, we describe our simulator for single robot and one target cases. The multi- robot and several targets cases will be described in section 5.

Our simulator uses the POMDP model and constructs a network structure of the belief nodes, which may be empty, holding an obstacle or a missing person. The robot can fly from one belief node to its immediate neighbour belief nodes. In two dimensional environment, we give each belief node a coordinate, so we can use $m_{x,y}$ to represent the belief node b at the coordinate (x, y), as described in sections above. We can use $m_{x,y}$ or b to describe the robot position.

The node b_0 is the initial belief node and the node b_{target} is the target belief node where the survivor might be found. The planning problem is to find the shortest belief node sequence to reach the belief node b_{target} from the initial belief node b_0.

The robot can move to any direction freely. For a 2D map, we limit its action set to 8 actions: up, down, left, right, up-left, up-right, down-left, as well as down-right. These actions correspond to the movement of the robot from one node to one of its immediate 8 neighbours, as shown in Fig. 18.

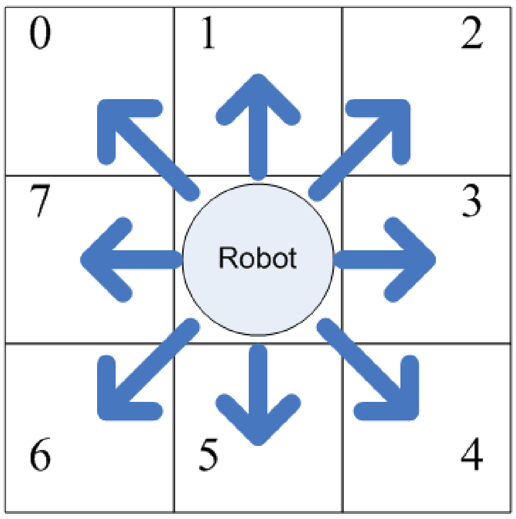


Fig. 18 Eight actions for a robot to move to its immediate neighbours

In our research, the robot uses the belief node structure to represent the uncertain environment and to make a plan. Therefore, the value function $V_\pi(b)$ of the POMDP is defined on the belief space B and is calculated for each belief node b. The value is used to compare and decide which belief node is the best one to move next, and we can compare different sequences of belief nodes with their values. In Section 2.4, the different algorithms are introduced to act as the value function, calculating the value for each belief node b, and making decisions of moving robot toward target b_{target}. These value functions are the main differences of various research. For example, the heuristic algorithm A* defines the evaluation function as $f = g + h$, where g is the path length from start to current position and h is the heuristic path length from current to target position.

Each robot has a limited detection distance, which can be adjusted in our simulator. However, the cell behind an obstacle cannot be detected by a robot. The robot can update part of its belief node structure based on its on-board sensor measurements, for possible re-planning in the CMP.

3.5.2 Thresholded A*

A* is a path planning algorithm in a certain environment. Former researchers use a threshold to transfer uncertainty to certainty and adjust the old A* searching tree to make a re-plan faster [39][96]. If the robot is blocked along the former planned path, a re-planning will find a new shortest path based on the updated OC map. The A* cannot guarantee that the robot travelled path is the shortest from the point of the CMP, sincethe actual travelled path is the combinations of the partway of the initial planned path and all re-planned paths.

The A* planner uses a threshold β to decide whether a cell will be treated as empty or blocked:

$$m_{x,y} = \begin{cases} 0 & p(m_{x,y} = 1) \leq \beta \\ 1 & otherwise \end{cases} \tag{18}$$

The certain map, recording the obstacles' exact positions, constructed from β is equivalent to assuming a particular map realisation. Higher value of β means higher risk, but the planned path length is shorter. It is hard to find a value of β to make sure that the obstacles' positions are similar to the actual states of the real environment. So, this algorithm is hardly to avoid the re-plans and decrease the actual travelled path length at the same time.

As the threshold increases, the algorithm will regard progressively more and more cells as if they were open. Although this means that there is a higher probability that the algorithm will discover a path if it exists, it also tends to pursue paths which have a very high probability of being blocked. Conversely, as the threshold value is reduced, the algorithm regards more and more cells as being occupied. This means that the algorithm will tend to find fewer paths, but the probability that those paths are open will be greater. Although we have tested with a range of parameter values, in this thesis we present results for $\beta = \{0.5,0.7,0.9\}$, these values are selected for situations from low risky to high risky based on the OC map.

3.5.3 Entropy-guided path planning

The entropy-based path planning algorithm [18] takes the local uncertainty, quantified through entropy, into account. It focuses on the information acquisition (to decrease the uncertainty) along a single path from start to target locations.

The standard entropy definition to measure the uncertainty of a probability distribution P over a domain D is:

$$H(D) = -\sum_{d \in D} P(d) \log P(d) \tag{19}$$

Information obtained is the change in the entropy of the distribution as a result of the mutual information measurement i obtained from an action, or observation:

$$I(D|i) = H(D) - H(D|i) \tag{20}$$

Burns [18] evaluates and plans a path in an uncertain environment based on the change of uncertain information. When a robot moves, it can collect information along the path to direct further exploration [18]. For each searching step, Burns computes $I(D|i)$ with the assumption that the step is reachable. Burns prefers to minimize entropy distribution of the candidate paths [18].

The shortage of the entropy based algorithm is the local minimal trap, because the lack of the global measurement.

3.5.4 The Maximum Collision-free Probability Planner

We design a path planner, called as the MaxProb Planner, which only finds the path with the highest collision-free probability in uncertain environments. The MaxProb planned path l can be defined as:

$$l = \underset{l}{\operatorname{argmax}} \left\{ \prod_{m_{x,y} \in l} [1 - p(m_{x,y})] \right\} = \underset{l}{\operatorname{argmin}} \left\{ - \sum_{m_{x,y} \in l} \log[1 - p(m_{x,y})] \right\} \tag{21}$$

The MaxProb Planner has the same searching process as the Dijkstra algorithm, except that it uses the collision-free probability (ignore the path length) to evaluate the candidate paths. For numerical stability, the MaxProb Planner finds the path which minimizes the negative log likelihood, as shown in (21).

There are three ideas to count the collision-free probability of the diagonal move, in the MaxProb planner. For example, in Fig. 19, the robot moves from the cell 4 to the cell 2. One idea is to multiply the collision-free probabilities of these four cells (in this case no diagonal movement will be planned). The second idea is to multiply the collision-free probabilities of the cell 1, cell 2, and the average collision-free probabilities of two neighbour cells (cell 1 and cell 3). The third idea is to ignore the two neighbour cells' occupancy probability. Here, we define the MaxProb planner with the third idea.

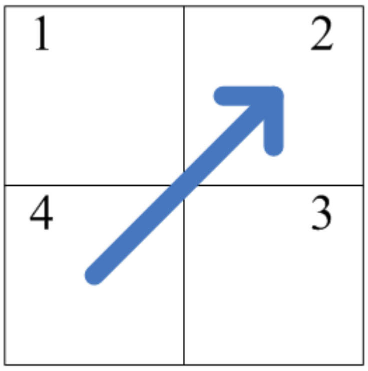


Fig. 19 The MaxProb diagonal movement example.

The reason that the MaxProb Planner does not count the cell 1 and the cell 3 is: the PD Planner does not count the neighbour cells' clusters for the diagonal movement in the PD Map. The possible shortest paths clustering in these two neighbour cells can be seen as the candidate possible shortest re-plan paths to overcome the obstacles along this diagonal movement. But the MaxProb Planner does not count the re-plans.

When we design the PD Planner, we find that a path l has a probability $p(l)$ to be a collision-free path and a probability $p_{opt}(l)$ to be the shortest path in the real environment (although we do not know the cells' states of the real environment). These two probabilities have the relation: $p(l) \geq p_{opt}(l)$. A path has the highest collision-free probability may have the lower probability to be the shortest path. Therefore, if the highest cluster area in the PD Map is far away to the highest collision-free path, the MaxProb Planner will not as good as the PD Planner. For example, the Fig. 23 shows the high clustering areas and the MaxProb Planner planned path shown in Fig. 24 is not lying in the high cluster areas.

Additionally, the MaxProb Planner ignores the path length and does not count the re-plan situations. These reasons make the different performance between the MaxProb Planner and the PD Planner.

3.5.5 Local Re-plans Strategy in Experiment

The PD Planner is a global path planning algorithm, but it can be used in local re-planning (plan a path based on the sub-map around the robot current position to overcome the local obstacle) as well. The local re-plan is a strategy to re-plan a path faster, suitable for all algorithms described in this thesis. It plans a path based on the sub-map around the robot position, in order to re-connect with the former planned path from the blocked point. It assumes that the last planned path is just blocked by a small obstacle. Therefore, the local re-plan cannot guarantee to find the global shortest path.

The local PD Planner re-planning uses a smaller OC map which can be applied for the cases that the clustering channel is not blocked completely. For example, in Fig. 20, the red cloudy area shows a local re-plan area, which is a small part of the map. The blue solid line A-C-D-F-E-B is the planned path. The local re-plan assumes the planned path is blocked locally, and its planning aims at re-connecting with the planned path. If the path is blocked at D-F, we can make a re-plan within the red cloud area, where the "stream" of alternative paths is available to reconnect with the path segment E-B. The blue dash line D-G-E-B is the local re-planned path in Fig. 20.

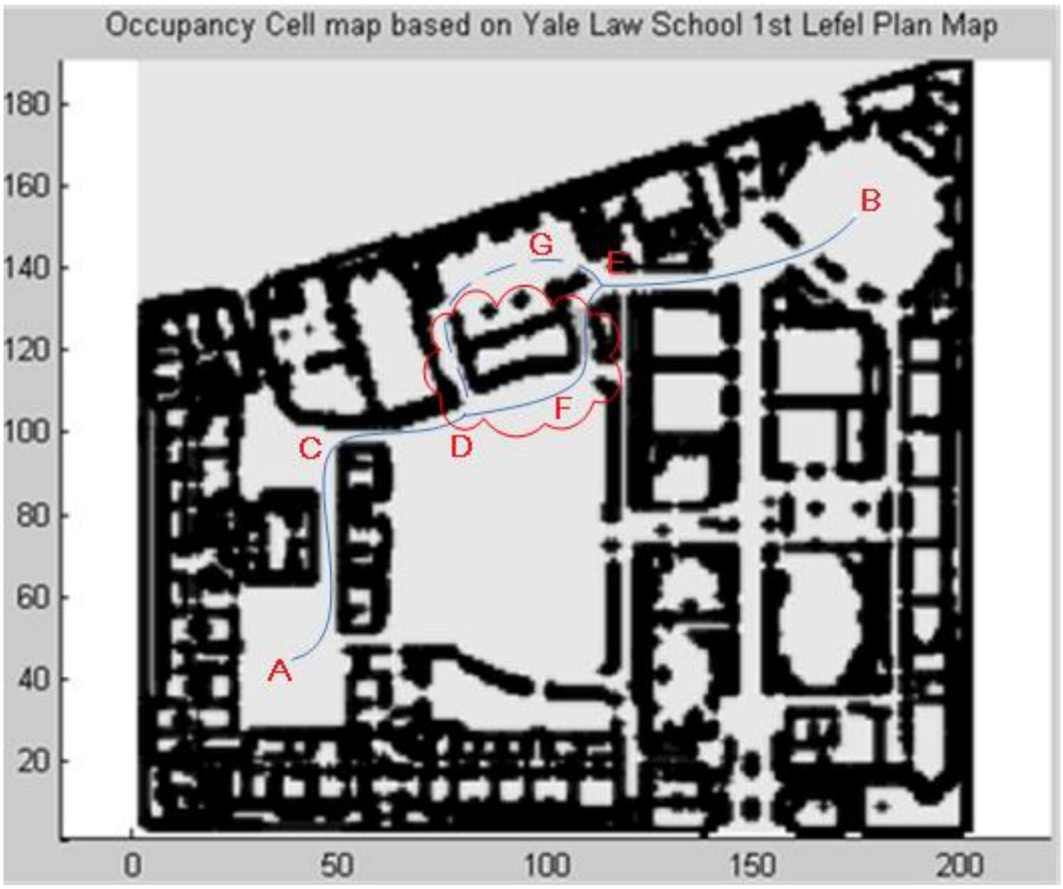

Occupancy Cell map based on Yale Law School 1st Lefel Plan Map
180
160
140
120
100
80
60
40
20
0
50
100
150
200
A
B
C
D
E
F
G

Fig. 20 The solid line: the planned path. The dash line: re-planned path.

3.5.6 The Experiment Results of the Monte Carlo Test

The test scenario consists of the following. First, a planner was presented with a nominal OC map which was constructed using prior information. The planner computed an initial path. The platform then executes its mission. The robot flies to each waypoint in the planned path. As it fly, it uses an idealized mapping algorithm: the sensors detect the nearest obstacles around the platform within a short distance (currently is 5 cells, and it can be changed to any integer in our simulator), and the sensor data is used to update the sub-map around the robot. Therefore, the global map M is updated gradually as the robot moves. When the platform encounters a cell which blocks the currently planned path, the path planner is executed again using the updated map information. If the path planner has the local re-plan interface, the simulator will employ the local re-plan function first. If the local re-plan cannot find a path, the simulator will call the global re-plan function of different planners. Currently, we have not implemented the re-plan strategy based on the new gap in the updating OC Map, which is one of the possible future research interests.

We compared the performance of three algorithms: the A* algorithm using a threshold OC map [80] (as explained in section 3.5.2), the entropy based path planning algorithm [18] (as explained in section 3.5.3), and the PD Planner. The metrics we recorded were: the time required to complete a scenario, the travel length, and the number of re-plans. The performance of each algorithm was assessed in terms of average path length, computational time, and probability of reaching the target. These results were computed over 3000 Monte Carlo runs, in which each run has a randomly sampled true map (the obstacle positions' probabilities obey the OC Map) for simulator to decide the robot collision correctly. The robot uses different algorithms to plan the path based on its OC Map. The shortest paths in these sampled true map instances are randomly distributed, and the average length of these shortest paths is 215.46.

In Fig. 21*,* we show the cumulative probabilities of different algorithms when the robot reaches target at each travelled length. We can see that, in the most of the cases, the PD Planner causes the platform to travel a significantly shorter path than the entropy algorithm, and the entropy algorithm produces a shorter path than A* for all choices of the threshold parameter. The reason is that the PD Planner exploits the global uncertainty information; the entropy algorithm only exploits local uncertainty information, and the A* algorithm exploits no uncertainty information at all. These results are validated in Table 1, which shows the average and standard deviation of the travel times and also shows that the PD Planner has a significantly smaller standard deviation.

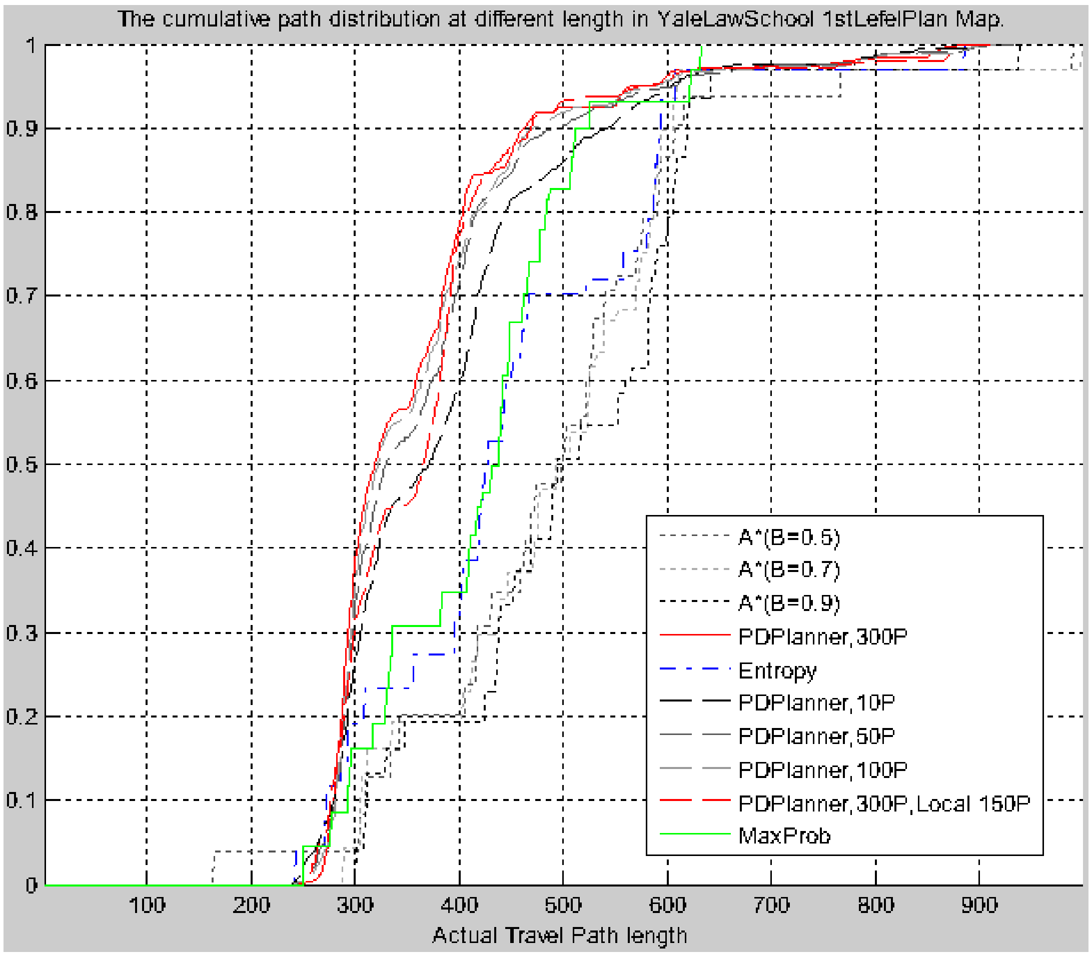


Fig. 21 The cumulative probabilities of different algorithms at each path length.

We expect the robot to travel the shortest length when it reaches the target. The significance of the probability to reach target can be considered at each travelled length. The shortest possible path length is 183, shown in Fig. 22. But, this shortest path going through a low path distribution passage, see the OC Map in Fig. 23 (the low path distribution area is below the point D). Its location indicates that it is not the shortest path in most possible instances of the real environment. That is the reason why the A* planner robot can get about 0.04% probability to reach target at this path length. However, if the path is blocked, the robot has to fly back down the shortest path and pursue the next shortest path. This will lead to the large jump in the path length.

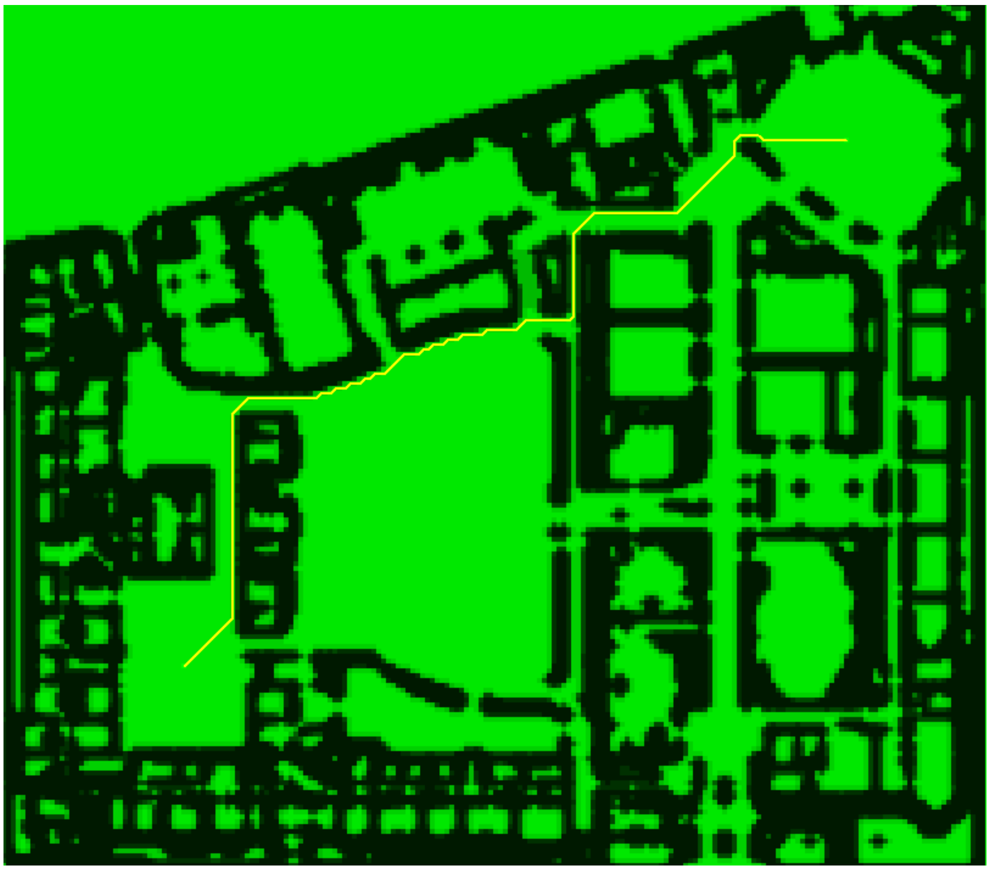


Fig. 22 The shortest path whose length is 183.

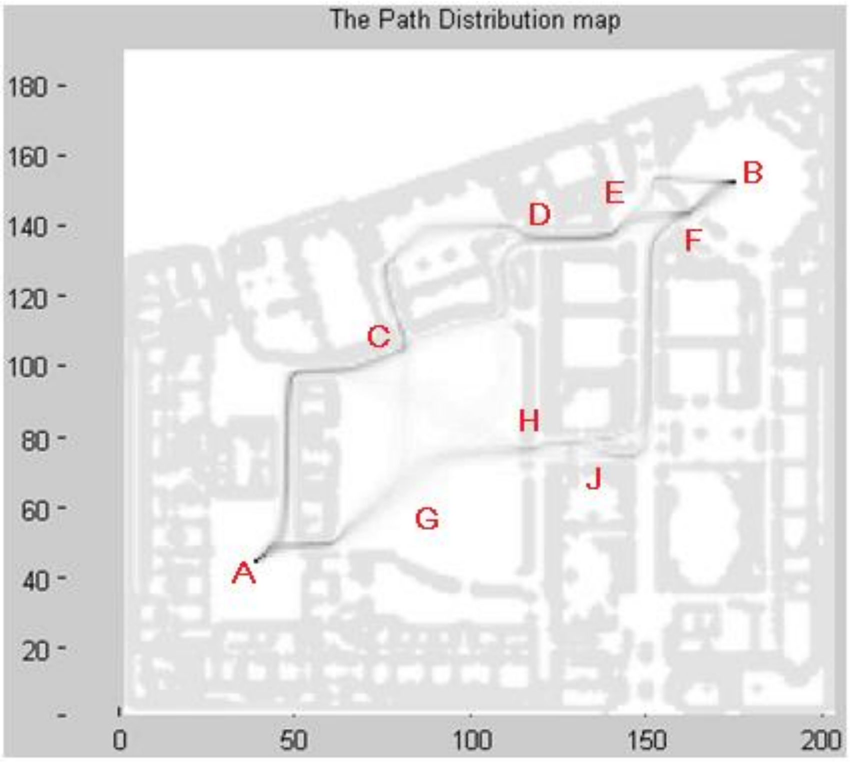


Fig. 23 The low Path Distribution below point D, (background is OC Map)

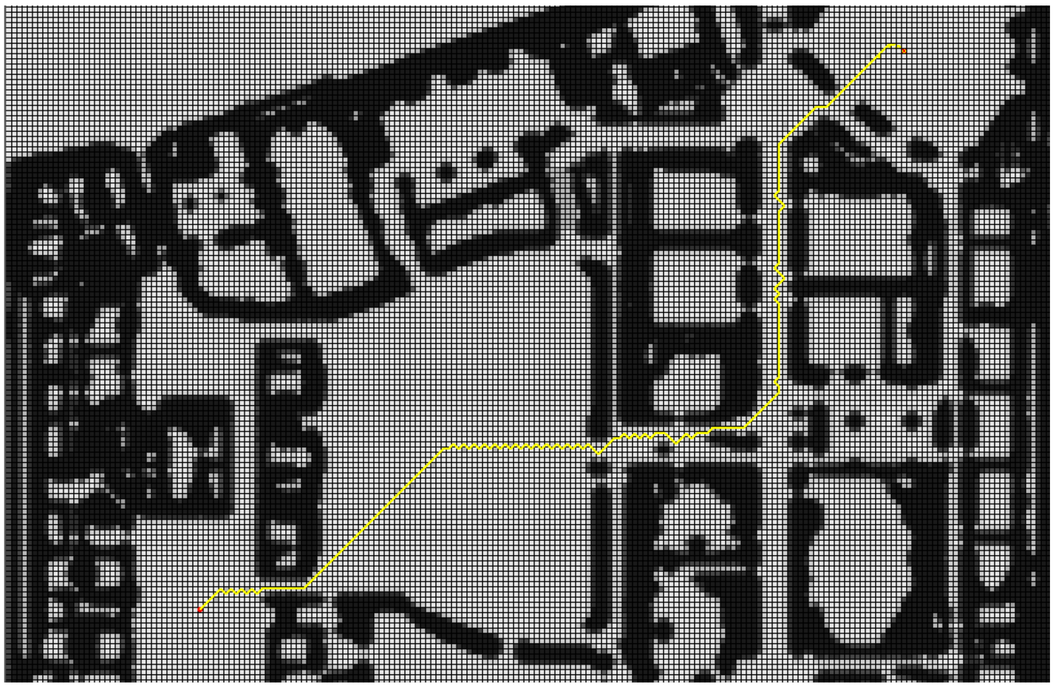


Fig. 24 The MaxProb planned path is not lying in the high cluster areas of the PD Map (Fig. 23).

Table 1 The average length and standard deviation of Fig. 21. “Local 150P” means that we use 150 particles for local re-planning.

Algorithms	Avg Length	Standard Deviation
A*, $\beta = 0.9$	509.2773	131.0747
A*, $\beta = 0.7$	492.162	135.2005
A*, $\beta = 0.5$	490.148	149.0865
Entropy	466.0447	135.0181
PDPlanner 10Particle	386.9380	115.6397
PDPlanner 50Particle	371.0313	112.8375
PDPlanner 100Particle	365.8473	111.6878
PDPlanner 300Particle	361.1553	109.3952
PDPlanner 300P,Local 150P	371.4227	110.5571
MaxProb	418.3893	95.0450

The highest cluster area in the PD map may only cover 40% of all possible map instances, while second high cluster area covers 30% and other cluster areas cover 30% all-together. The bigger particle can estimate the PD Map more accurate. This is the reason why the deviation in Table 1 decreases as the number of particle increases.

The analysis so far has hypothesizedthat the nominal and actual occupancy maps are the same. However, it is hard to compute the occupancy probabilities accurately. To examine the impact of these errors, we repeated the experiment. However, the actual occupancy map was computed by perturbing the nominal occupancy map. Specifically, the actual occupancy map is given by

$$p_{actual}(m_{x,y} = 1) = p_{nom}(m_{x,y} = 1) + \gamma(U_{x,y} - 0.5) \tag{22}$$

where $U_{x,y}$ is a uniform random variable drawn in the range [0,1]. The value of $p_{actual}(m_{x,y} = 1)$ was clamped to lie in the range [0,1]. We investigated the results for the cases when $\gamma = 0.3$ and $\gamma = 0.5$. The results are shown in Fig. 25 and Fig. 26 respectively. Despite the large errors in the nominal occupancy map, the PD Planner still produces the shortest path. Table 1, Table 2 and Table 3 show the average and standard deviation of the lines in Fig. 21, Fig. 25 and Fig. 26, respectively. We can conclude that the PD planner can get a higher probability to reach target within a more condensed travel length than others, although the average path length of PD Planner increases as the occupancy probability error increases.

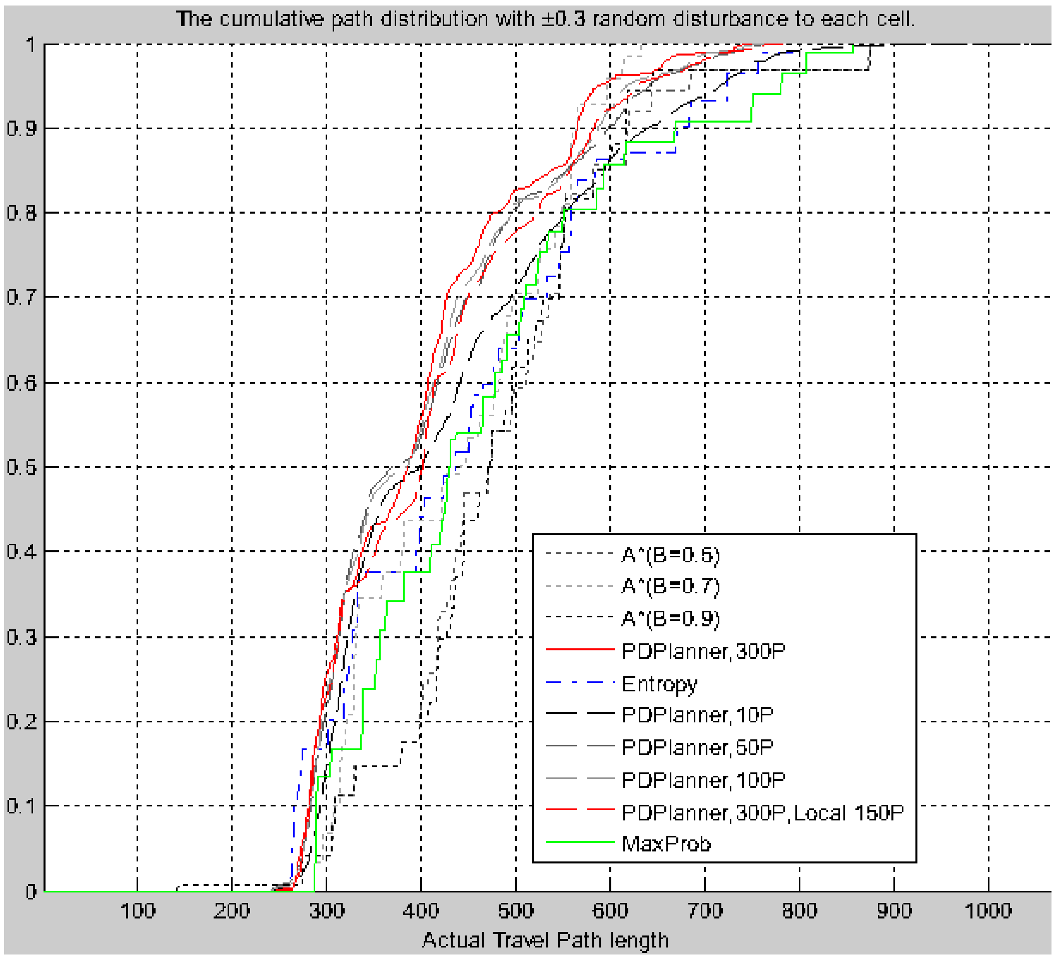


Fig. 25 : The cumulative probability of the actual travel path length for $\gamma = 0.3$.

Table 2 The average length and standard deviation of the paths in Fig. 25

Algorithms	Average Length	Standard Deviation
A*, $\beta = 0.9$	480.0795	120.4837
A*, $\beta = 0.7$	434.7765	104.1231
A*, $\beta = 0.5$	480.7590	123.9443
Entropy	445.5380	144.4400
PDPlanner 10Particle	431.9455	139.4256
PDPlanner 50Particle	405.2715	120.1786
PDPlanner 100Particle	402.6400	116.6115
PD Planner 300Particle	397.9580	109.9135
PDPlanner 300P, Local 150P	410.6500	118.2655
MaxProb	461.9740	143.3752

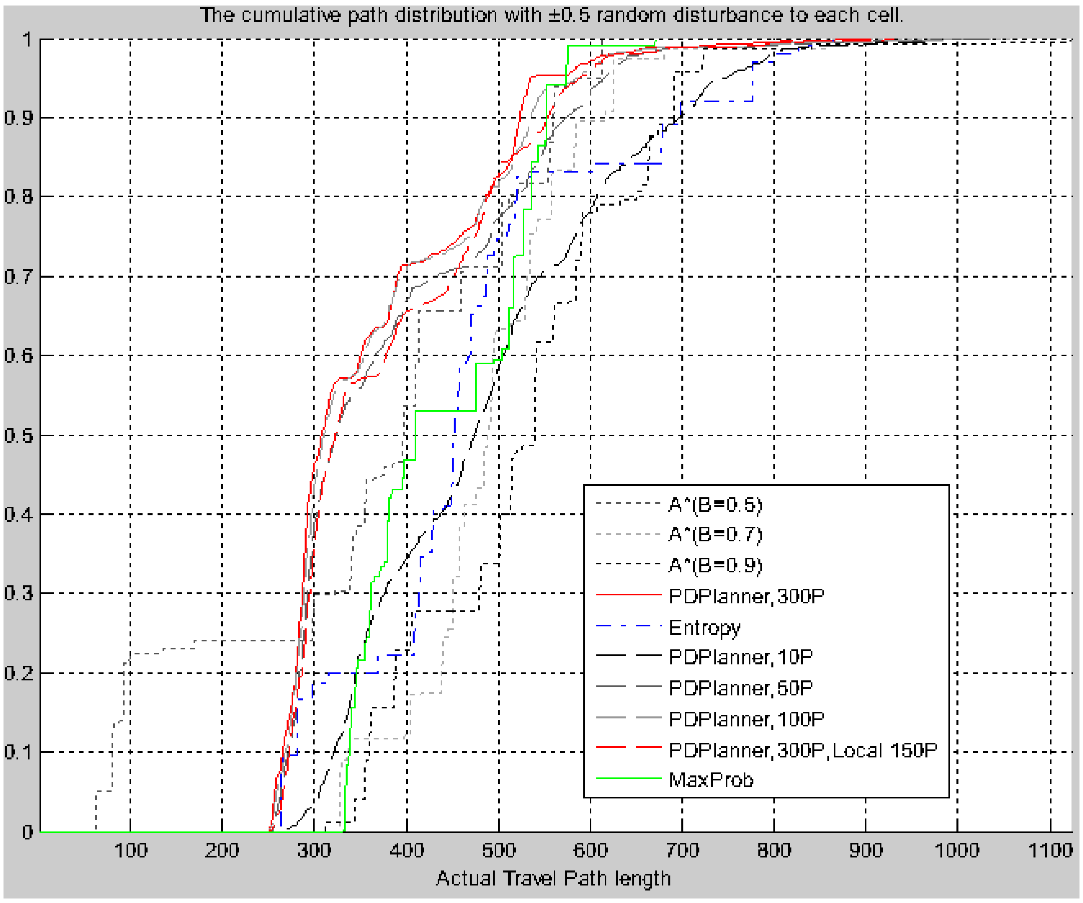


Fig. 26 The cumulative probability of the actual travel path length for $\gamma = 0.5$

Table 3 The average length and standard deviation of the paths in Fig. 26

Algorithms	Average Length	Standard Deviation
A*, $\beta = 0.9$	525.9680	124.1112
A*, $\beta = 0.7$	490.9960	152.7526
A*, $\beta = 0.5$	358.7960	173.6503
Entropy	466.6490	144.464
PDPlanner 10Particle	487.1890	142.3330
PDPlanner 50Particle	385.9250	127.7427
PDPlanner 100Particle	370.4900	118.0775
PD Planner 300Particle	365.4680	111.9697
PDPlanner 300P, Local 150P	381.2160	117.7380
MaxProb	442.4190	88.3116

The average number of times a path has to be re-planned is shown in Table 4. Re-planning occurs when a path is blocked. On average, the PD Planner has to re-compute paths significantly less frequently than the other algorithms.

Table 4 The average re-plan numbers of different algorithms when the nominal and actual occupancy maps are the same, and when they differ

Algorithms	Original	$\gamma = 0.3$	$\gamma = 0.5$
A*, $\beta = 0.9$	65.664	54.093	68.630
A*, $\beta = 0.7$	52.194	52.763	59.791
A*, $\beta = 0.5$	46.531	50.930	54.725
Entropy	48.560	52.310	56.568
PDPlanner 10Particle	38.429	48.455	58.383
PDPlanner 50Particle	35.702	41.416	42.145
PDPlanner 100Particle	35.004	40.596	39.422
PD Planner 300Particle	34.434	40.419	38.300
PDPlanner 300P, Local 150P	36.567	43.578	43.981
MaxProb	30.0760	32.8505	36.0370

3.5.7 The effect of Particles for PD Planner

We look at sampled true maps as the points in the Monte Carlo Sampling Space. So, each sample is a particle in the space. More particles provide a more accurate PD map for the PD planner to find the path, but it will also increase the computational cost. The number of particles is linearly related to the computational cost, see Fig. 27. In the following experiment, we use four different particle sizes (10, 50, 100, 300) for PD Planner with different occupancy probability errors in OC map.

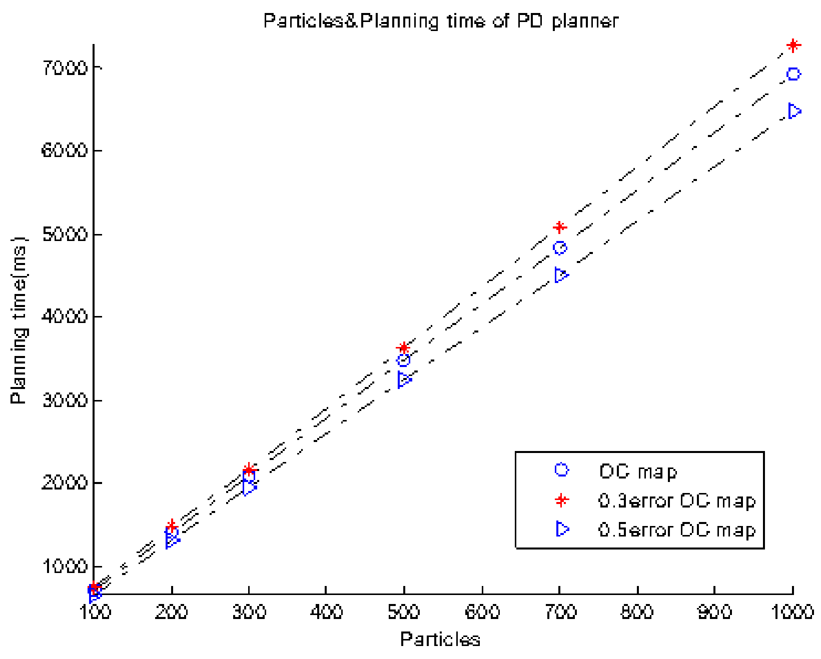


Fig. 27 Running time for the PD Planner with different numbers of particles. The $0.3/0.5$ error means: the occupancy probabilities of cells in OC Map for PD planner have a $\pm 0.3/0.5$ error.

To sample one map instance, the states of all N grid cells in OC Map are sampled based on their occupancy probabilities, respectively. We run A* to find the shortest path in each map instance, whose complexity obeys $O(N^2)$. The PD Planner uses s particles to estimate the PD map. To plan a path based on the PD Map can be seen as a Dijkstra searching process, in which the evaluation g is replaced by the path distribution value $P_d(l)$. In the worst case, Dijkstra will visit all nodes in the map to construct the searching tree. So, its complexity obeys $O(N)$. Therefore, the PD Planner calculation cost can be rewrite as

$$O(PD\ Planner) = s[O(N) + O(N^2)] + O(N) = sO(N^2) \tag{23}$$

3.6 Summary

In this chapter, we assume that there is only one target for one robot to go. We focus on how to plan a path to travel efficiently in uncertain environments. It is the basis for our later research.

The PD Planner is an approach to solve the path planning problem in an uncertain environment. The clustering value is the path distribution probability of the cells in PD map, which indicate where the paths are likely to go. The exact path distribution map is very difficult to calculate, so we estimate the PD map instead.

We compare the actual travelled length of the robot with the PD Planner and other typical algorithms in uncertain environments.

One important shortage of PD Planner is that its computational cost $sO(N^2)$ is higher than other algorithms. In the next chapter, we will analyse the computation cost of the PD Planner, which will increase fast when the particle number increases. Therefore, we introduce two ways to reduce the computational cost. In Chapter 6, we will extend PD Planner to the scenario of one robot to visit multiple targets, meanwhile the multiple robots visit multiple targets.

4 Reducing the Computational Cost of PD Planner

4.1 Introduction

In order to estimate the Clustering Value (CV) Map, we need to sample many possible map instances and find the POS in each of these map instances. This operation job requires lots of time to compute the result. In this chapter, we will develop two strategies to reduce the computational cost of PD Planner. We analyse where the extremely expensive computations come from and propose two strategies. The first, called Sampling in Planning Process (SiPP), performs lazy sampling within the planning algorithm of itself. It is a very simple strategy, but it has a surprising impact on the performance of the algorithm. We call the second the Hierarchal PD Planner that performs dimensionality reduction by decomposing the environment into homogeneous regions.

The PD Planner will be used as an implementation example to illustrate how these two strategies work. We show that these approaches can reduce the computational costs with minimal loss of performance.

The PD Planner shows that the more particles s we sample, the more accurate the PD Map and the better the performance will be. However, more samples require more calculations. If s samples are drawn and we use A* to find paths in sampled map instances, the computational cost is

$$s[O(N)+O(N^2)]+O(N)=sO(N)+sO(N^2)+O(N) \tag{24}$$

This equation is the same as (23). The first $O(N)$ term arises from the sampling of M_{oc} to get M^i. $O(N^2)$ is the cost of the worst case in A* algorithm finding a path in M^i [38]. The final $O(N)$ term arises from planning with the PD Map.

Since s is typically large, the computational cost is dominated by the first two terms. This analysis suggests that two strategies should be pursued – the first is to reduce the computational cost of sampling the map instances and the second is to reduce the overall dimension of the map.

To test the time budget of each part, we use the PD Planner to do the following experiment. We select the OC Maps with different sizes, from 50*50 to 300*300 grid cells. For each size, we randomly generate 5 different OC Maps (randomly generate a blurred map just as we did in Chapter 4). The start and target cells are located at top-left and bottom-right corner, or at top-right and bottom-left corner. The PD Planner uses 300 particles to estimate the PD Map. We run 1000 times for each map to get the average time budget (sampling, running A*, planning with PD Map) of the original PD Planner. The experimental result is shown in Fig. 28, in which the "Original" means no strategy to decrease the computational cost of PD Planner. Here, "OriginalSample" corresponds $sO(N)$, "OriginalAStar" corresponds $sO(N^2)$, "OriginalPlanner" corresponds $O(N)$, and "OriginalTotal" corresponds (24).

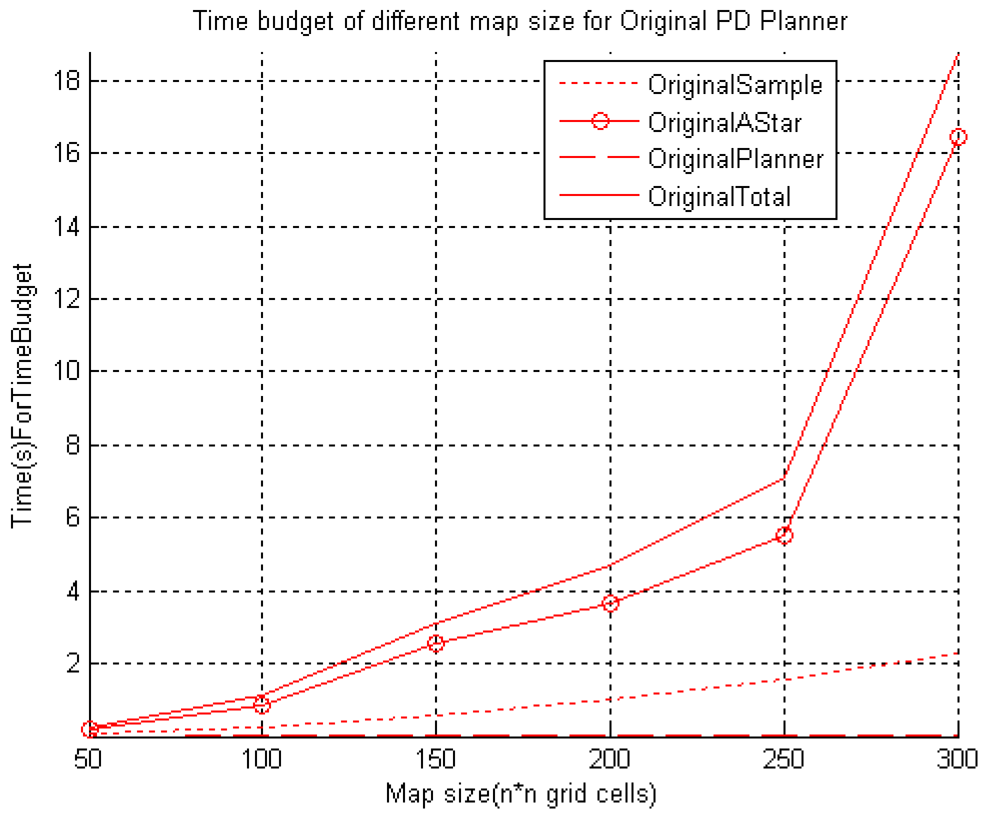


Fig. 28 The time budget parts of PD Planner

We can see that the main time budget of PD Planner lies in the sampling operation (“OriginalSample”) and finding paths in sampled map instances (“OriginalAStar”).

4.2 Sampling in Planning Process (SiPP)

The previous implementation of the map sampling step sampled all the cells in the path. However, it turns out that the MDP planning algorithm (like Dijkstra, A*, RRT, etc.) rarely checks all the grid cells in the planning process. To test the number of cells visited, we randomly generate different size maps (10000 instances for each size) and compared the number of cells having been visited. The results, shown in Fig. 29, suggest that empirically less than 50% of the cells are visited. This suggests a surprisingly simple heuristic which we call Sampling in Planning Process (SiPP): rather than sample the entire map state at the beginning of a Monte Carlo run, it is possible to embed the sampling in the A* algorithm directly.

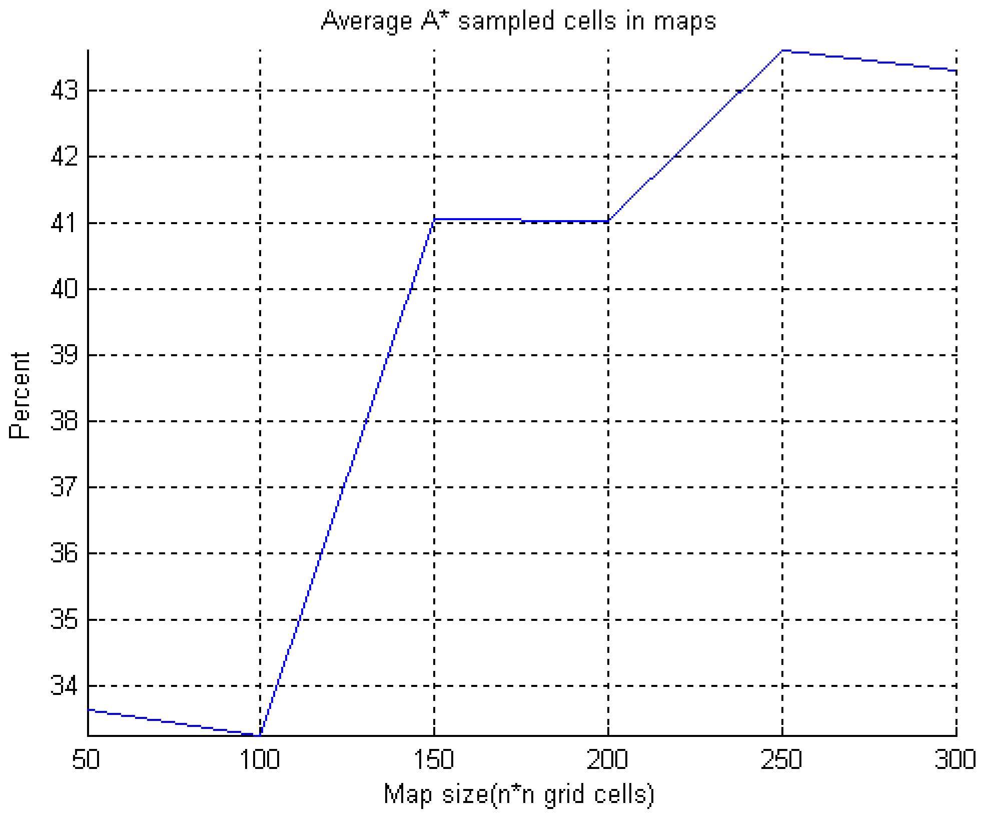


Fig. 29 The average A* checked cells in different size Maps (10000 instances for each size).

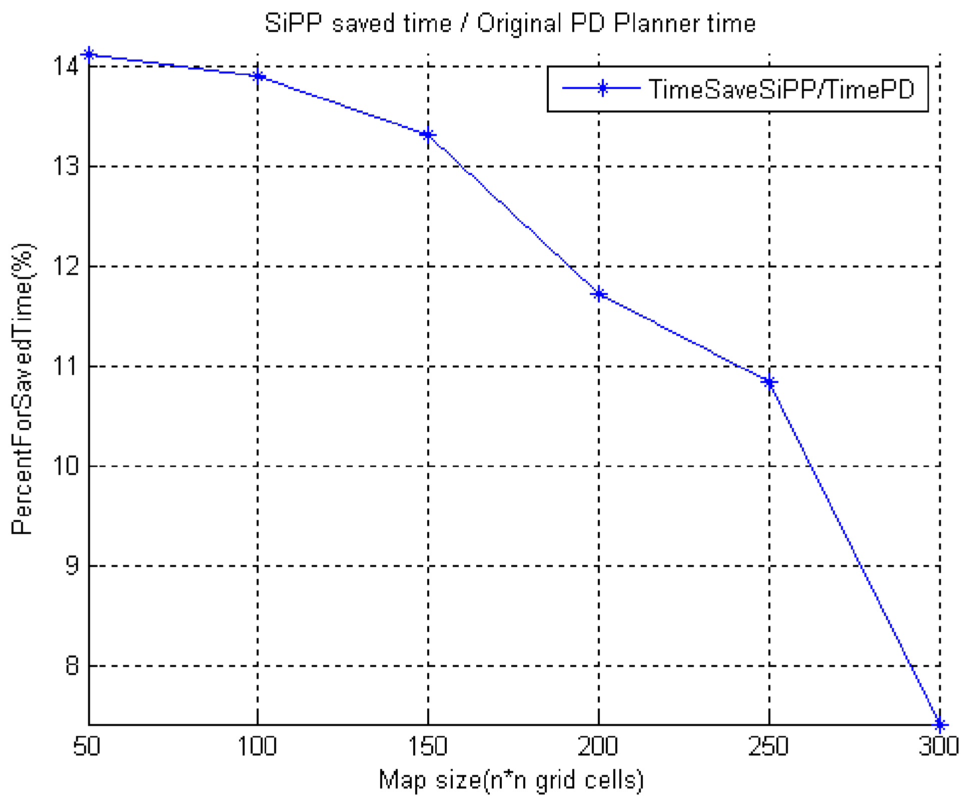


Fig. 30 The average SiPP saved time based on completely random OC maps

The SiPP strategy is designed to sample the states of those grid cells that checked by the searching process of the MDP algorithm. Therefore, in each map instance M_i, only those grid cells already checked by the searching process (like Dijkstra, A*, etc.) have a sampled and certain state: empty or blocked, and the states of other grid cells are unknown (un-sampled). Those grid cells having not been visited by the searching process will not affect the planning process, so the SiPP strategy will not affect the location of the planned path.

In Fig. 30, we show that the SiPP saved time that might otherwise be used for the percentage of the original PD Planner, based on the completely random generated OC maps (no typical structures, like walls or rooms, and no idea heuristic available). Since the start and target position located at the corner of the map, the searching process (like A* algorithm) always checks most part of the map to find the shortest path for bigger size map.

In order to implement SiPP, we insert the sampling operations into the searching process where grid cells' states are required. The pseudo code of SiPP function, based on A* algorithm, OC_A, with an OC Map M_{oc} is shown in Pseudo-code 5.

Pseudo-code 5 The adjusted A* for estimating the PD map of PD Planner.

```
function OC_A*(start, target, M_oc)
1   init closedset as empty,and openset as {start}
2   init back-pointer[] array
3   g_score[start] = 0        //lowest cost from start to current node.
4   f_score[start] = g_score[start] + heuristic(start, target)

5   while openset is not empty    //some nodes are waiting to extend
6   {   cur = the node in openset having the lowest f_score[] value
7       if cur = target
8          return reconstruct_path(start, target)

9       remove cur from openset
10      add cur to closedset
11      for each neighbor of node cur
12      {   tentative_g=g_score[cur]+dist_between(cur ,neighbor, M_oc)
13          if neighbor in closedset
14             if tentative_g>= g_score[neighbor]
15                continue

16          if neighbor not in openset or tentative_g < g_score[neighbor]
17          {   back_pointer[neighbor] = cur
18              g_score[neighbor]= tentative_g
19              f_score[neighbor]=g_score[neighbor]+heuristic(neighbor, target)
20              if neighbor not in openset and tentative_g < infinity
21                 add neighbor to openset
22          }
23      }
24  }
25  return failure
```

The sampling operation is defined in the $function\ dis\_between(cur, target, M_{oc})$, in line 12. This function also makes sure that each grid cell is sampled only once in one round. The pseudo code of function $dis\_between(cur, target, M_{oc})$ is shown in Pseudo-code 6:

Pseudo-code 6 The dis_function for inserting the sampling process into the planning process

```
function dis_between(cur, target, $M_{oc}$)
if target is sampled
    target_state = recall sampled state
else
{   target_state = random_state(occupancy probability of $M_{oc}$)
    record sampled state of target for late use
}

if target_state is empty
    return  Euclidean_distance(cur, target)
else
    return infinity
```

The sampling efficiency depends on the effectiveness of the A* heuristic function in the given OC Map. But, as we analysed and shown in the experiments, the SiPP is highly effective for low dimensions and low effective for high dimensions. This suggests that the methods which reduce the overall dimensions of the problem will have a significant impact on the performance of the algorithm.

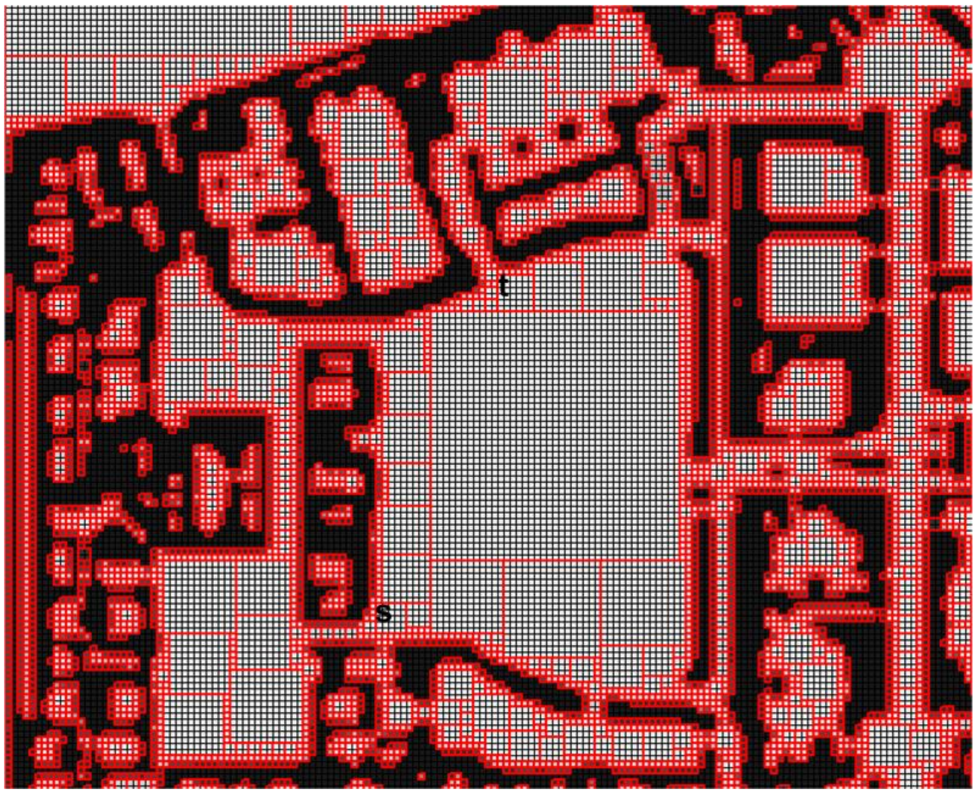
Therefore, the SiPP can save more time for small size map corresponding to the smaller number of nodes in the searching tree. It does not save much time when N increases. But there are less grid cells for small size map between the start and target position, so the searching process will check less to find the shortest path.

In the next section, we propose the Hierarchical PD Planner.

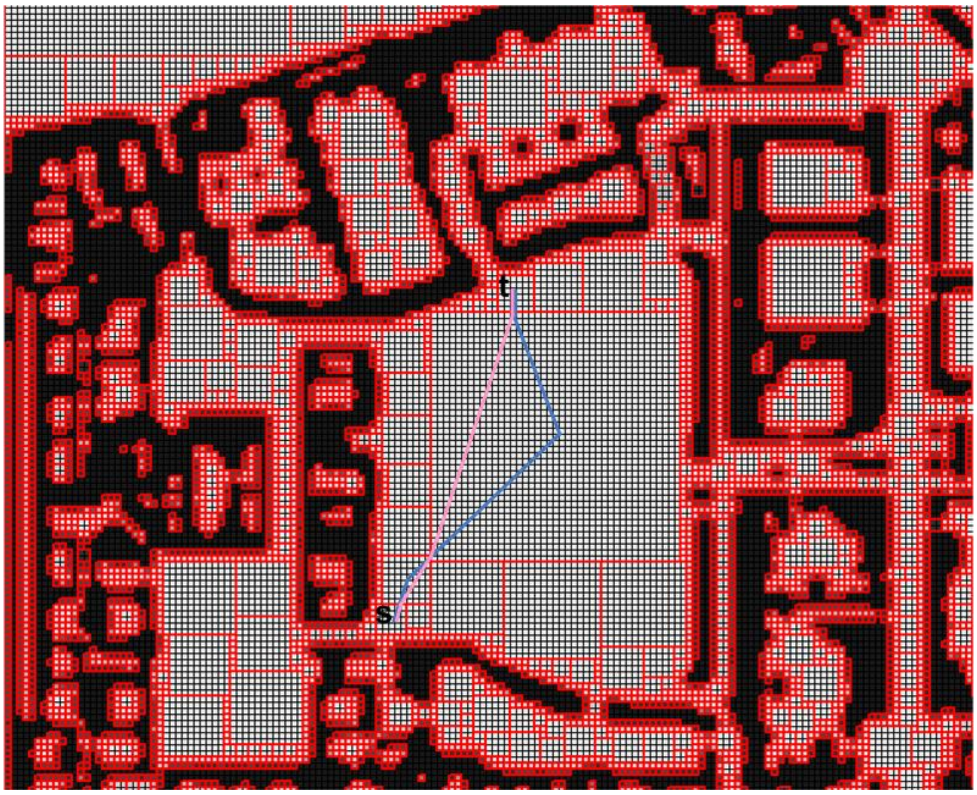
4.3 The Hierarchal PD Planner

The second challenge is to reduce the dimensionality of the environment. In this section, we describe a method called the Hierarchal PD (HPD) Planner. The main idea of the HPD Planner is to hierarchically decompose the map into a small set of homogeneous regions, then find a path based on this high level map, and restore the path to the original map. The properties of each region can be cached in advance. As a result, the PD Planner can be viewed as running over a graph representation of the environment, where each vertex describes the properties of a region, and each edge encodes the links between regions. The essential idea is to represent the grid cells with same occupancy probability as one region. So, the node of the searching space for path planning can be decreased.

Given the raw occupancy map Fig. 15, areas of similar probability are merged to create the region map, as shown in Fig. 31(a), where the robot seeks to move from the start position “s” to the target position “t”. In Fig. 31 (a), we use the red lines to show the constructed Region Map, where the grid cells in the same region have the same occupancy probability. In Fig. 31 (b), we draw the region path with a blue line, and the pink line shows the restored grid cell path based on the region path.



(a)



(b)

Fig. 31 The red squares are the regions in an OC Map. A robot moves from “s” to “t” with different path strategies in (b).

The Hierarchical PD Planner (HPD Planner) has three steps: 1, Construct a Region Map; 2, Find the clustering of paths in regions by running the PD Planner on just the Region Map; 3, Represent the path in OC Map grid cells.

When we run the PD Planner in Region Map, we estimate the path distribution in Region Map (paths cluster in regions). The Region Map decreased the dimension of OC Map, so the PD Planner in Region Map will be faster. The SiPP strategy, which is more effective at lower dimensions, can also be applied here.

4.3.1 Constructing the Region Map

We define each region as a square or rectangle sub-map, containing grid cells having the same (or similar) occupancy probabilities. Zou [93] uses a Sub-Region to plan a path. Lingelbach [94] looks the entire map as one cell and decompose it into smaller grids to plan a path. Both of them plan a path by decreasing the searching space. Comparing with former researchers, the main difference of our region map is that we transfer the occupancy probability to different regions and use it to find the passable probability of each region.

To construct the Region Map, we define a region by its top-left and bottom-right grid cells in the OC Map. We scan the OC Map from its top-left corner to bottom-right corner, as shown in Fig. 32, where TL is top-left grid cell of the region, BR is the bottom-right grid cell of the current constructing region, New1 and New2 are top-left grid cells added into wait list after we construct the current region.

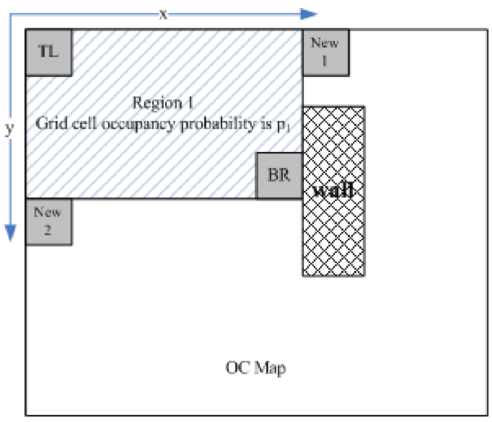


Fig. 32 Region constructing example

A waiting list is used to record the top-left grid cell coordinate for finding a new region. The pseudo code for finding regions based on an OC Map is shown in Pseudo-code 7.

Pseudo-code 7 The region construct function for estimating CV map (PD map of PD Planner)

RegionConstructFunc(OC Map:M_{oc})

```
1   Init waiting grid cell list S={cell(1,1)}
2   while S is not empty
3       take a cell $m_{x,y}$ from S, and
4       define the current region's top-left cell $m_{x,y}^{TL} = m_{x,y}$
4       set current region's bottom-right cell $m_{x,y}^{BR} = m_{x+1,y+1}$
5       while (CheckRegionLimitation($m_{x,y}^{TL}$, $m_{x,y}^{BR}$) is true)
6           $m_{x,y}^{BR} = m_{x+1,y+1}^{BR}$    //for square, expand x/y first for rectangle
7       end while
8   $m_{x,y}^{BR}=m_{x-1,y-1}^{BR}$, and add current region to region map Mr
9   add cell{X($m_{x,y}^{BR}$)+1, Y($m_{x,y}^{TL}$)} and cell{X($m_{x,y}^{TL}$), Y($m_{x,y}^{BR}$)+1} into S
10  end while
11  return Mr
```

The function "CheckRegionLimitation" is defined to check a region is permitted or not, and its pseudo code is shown in Pseudo-code 8.

Pseudo-code 8 The region check function of constructing a region

```
CheckRegionLimitation(top-left cell:$m_{x,y}^{TL}$, bottom-right cell:$m_{x,y}^{BR}$)
$p_{average}$= sum of p($m_{x,y}$) in region / cell count;
Δp = 0.0     //the max difference permited
for new added grid cells in this region {$m_{x,y}^{TL}$, $m_{x,y}^{BR}$}
   if (one cell's p($m_{x,y}$) > $p_{average}$ + Δp)
      or (one cell's p($m_{x,y}$) < $p_{average}$ - Δp)
      return false;    //some cell's p cannot be accepted
   end if
 if (one cell in region{$m_{x,y}^{TL}$, $m_{x,y}^{BR}$} is included in another region)
    return false;
end if
return true;
```

The region shape can be defined based on the characteristics of the OC Map, if they are pre-known. Here, we find a square region first, and then we try to expand the new found square region to rectangular region by expanding the region along the x-axis or y-axis.

We use $p_{average} \pm \Delta p$ to define the occupancy probability similarity in one region, where p_average is the average occupancy probability of grid cells in the region. If $\Delta p \neq 0$, we will accept that some grid cells in one region have different but similar occupancy probabilities. In following part, we use $\Delta p = 0$ to construct a region map.

Once all the regions have been found, it is easy to find their connections with the top-left and bottom-right grid cell coordinates of the region. The region map can be looked as a graph, where each node is a region and the edge shows the connection of two closest neighbour regions (two regions have a segment of same boundary).

For the worst case, each grid cell of the map will be visited twice: to set an extension of a region and to set an initial region, except the top-left and the bottom-right grid cells. Therefore, in the worst case, the computational cost for constructing the region map is $O(N^2)$, where N is the grid cell number of the grid cell map. This occurs when each region consists of a single cell, which is unlikely to occur in practice.

4.3.2 Computing Travel Length and Occupancy Probability of Regions

The PD Planner uses the occupancy probabilities and the travel cost of moving from one grid cell to another. The Hierarchical PD Planner requires these quantities for each region. In this subsection, we develop methods to compute the occupancy of the travel length between two neighbouring regions and the occupancy probability of a region.

In the Region Map, the path from region A to the neighbouring region B is defined in terms of moving from the centre of region A to the centre of region B, passing through the middle grid cells of the same border between the region A and the region B.

Since all the grid cells in the same region have the same occupancy probability, the four cells shown in Fig. 33 (Line AB is the same border of Region A and B) are sufficient to define the path. These are:

m_1 : the centre grid cell of region A

m_2 : the middle grid cell of line AB, in region A

m_3 : the middle grid cell of line AB, in region B

m_4 : the centre grid cell of region B

The path length from region A to region B is the Euclidean Distance from m_1 to m_2, then to m_3, then to m_4.

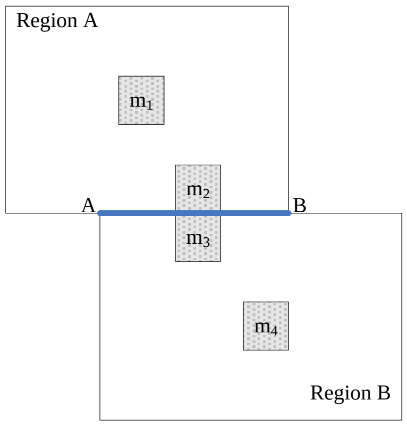


Fig. 33 The region connection is described with four grid cells.

The region occupancy probability p_{region} specifies the probability that a robot, when entering a cell, is not able to get to the centre of that region. This is similar as the path un-exist probability in 2-dimensional percolation theory [101]. The difference is that we want the robot to reach the target position, while the percolation theory wants the robot to reach any point on the bottom boundary of the region.

This region occupancy probability p_{region} is a function of the region size S_{region}, occupancy probability p_{oc} of the region's cells, and the target cell is the region centre, written as

$$p_{region} = func\{S_{region}, p_{oc}\} \tag{25}$$

We also define the p_{succ} as the crossing probability that the robot can reach the region centre cell successfully. We have $p_{succ} + p_{region} = 1$. For the same uncertain environment, we can use the OC Maps with different number of grid cells to represent it. If one grid cell represents a smaller area, the OC Map will have more grid cells. In the other case, the OC Map will have less grid cells. In these different size maps, the robot should have the same probability to reach its target position from the same start position. This is same for the region map with different number of regions. Therefore, we should transfer the occupancy probability from grid cell to region.

We design an experiment to see the changes of p_{succ}, when an OC map (with different S_{region} and different p_{oc}) is transferred to a one region map M_{region} (only contains one region). For example, we have an OC Map M_{21*21} (21*21 grid cells) and transfer it to a region map M_{region} (contains only one region, where the region size is $S_{region} = 21$). The occupancy probability of the grid cells in M_{21*21} is $p_{oc} = 0.3$. The robot has the same probability p_{succ} to reach the centre point from the boundary point via the shortest path, no matter it moves in M_{21*21} or in M_{region}. This process is similar with the percolation theory, which discusses the probability of an open shortest path existing between a centre point and a boundary of the environment [106].

Specifically, a set of different OC Maps of different sizes were selected and a range of occupancy probabilities $p_{oc} = \{0.1 \sim 0.9\}$ were used. We sample each OC Map 1 million instances to find the number of instances in which the shortest paths exist, in order to estimate p_{succ}. For example, we have an OC map with 101*101 grid cells and $p_{oc} = 0.4$. Its possible shortest paths clustering areas are shown in Fig. 34, where the brighter and higher points have more paths clustering. We get p_{succ} is about 0.1 in this example. The changes of p_{succ} for different S_{region} and p_{oc} are shown in Fig. 35, where the x-axis is the p_{oc} and the y-axis is the p_{succ}. We can see that the crossing probability p_{succ} decreases exponentially as the map size increase, which is the same as Menshikov's percolation theory in [105]. Aizenman [106] proved that this crossing probability is the same for conformal map (one grid cell has more than two neighbours), which means that we can calculate the crossing probability for the hexagon cell map and the octagon cell map in the same way.

Our experiment results show that the p_{succ} has three parts:

1, When $p_{oc} \leq 0.3$, the p_{succ} is almost a linear function of p_{oc}, which can be estimated as $p_{succ} = 0.985 - 1.85 * p_{oc}$

2, When $p_{oc} \geq 0.7$, the p_{succ} is close to 0 for different size maps when the occupancy probability.

3, When $0.3 < p_{oc} < 0.7$, the p_{succ} decreases exponentially based on S_{region} and p_{oc}.

For the 3rd case, we can assume the crossing probability p_{succ} based on the percolation theory [105][106], which looks like:

$$p_{succ} = a_1 * \left(p_{oc} * S_{region}\right)^{a_2} + a_3 * (p_{oc})^{a_4} + a_5 * \left(S_{region}\right)^{a_6}$$

$$+a_7 * \sin\left(\left(p_{oc} * S_{region}\right)^{a_8}\right) + a_9 * \exp((p_{oc})^{a_{10}}) + \mathrm{a}_{11} * \cos\left(\left(S_{region}\right)^{a_{12}}\right) + \mathrm{a}_{13} \tag{26}$$

In our experiment, we have S_{region}, p_{oc} and the result p_{succ}, as illustrated in Table 5. Based on these values, we can use the Matlab function “lsqcurvefit” to estimate the parameter matrix $[a_i], where\ i \in \{1,2,3, \dots, 13\}$.

Table 5 Experiment results of p_{succ} for different S_{region} and p_{oc}

S_{region}	$p_{oc} = 0.3$	$p_{oc} = 0.4$	$p_{oc} = 0.5$	$p_{oc} = 0.6$	$p_{oc} = 0.7$
5	0.4426	0.2827	0.1593	0.0769	0.0307
11	0.4216	0.2022	0.0636	0.0134	0.0018
21	0.4291	0.1601	0.0201	0.0011	0
41	0.4358	0.1292	0.0028	0	0
101	0.4354	0.1038	0	0	0
301	0.4360	0.0935	0	0	0

The estimated result of $[a_i]$ is $[-9.7432, -8.9417, 0.2617, -1.1183, 0.8947, -0.5693, 0.6703,\ -1.9497,\ -0.3504, 0.1604, -0.5430, 0.2366, -0.0979]$. We show the estimated p_{succ} with red lines in Fig. 35.

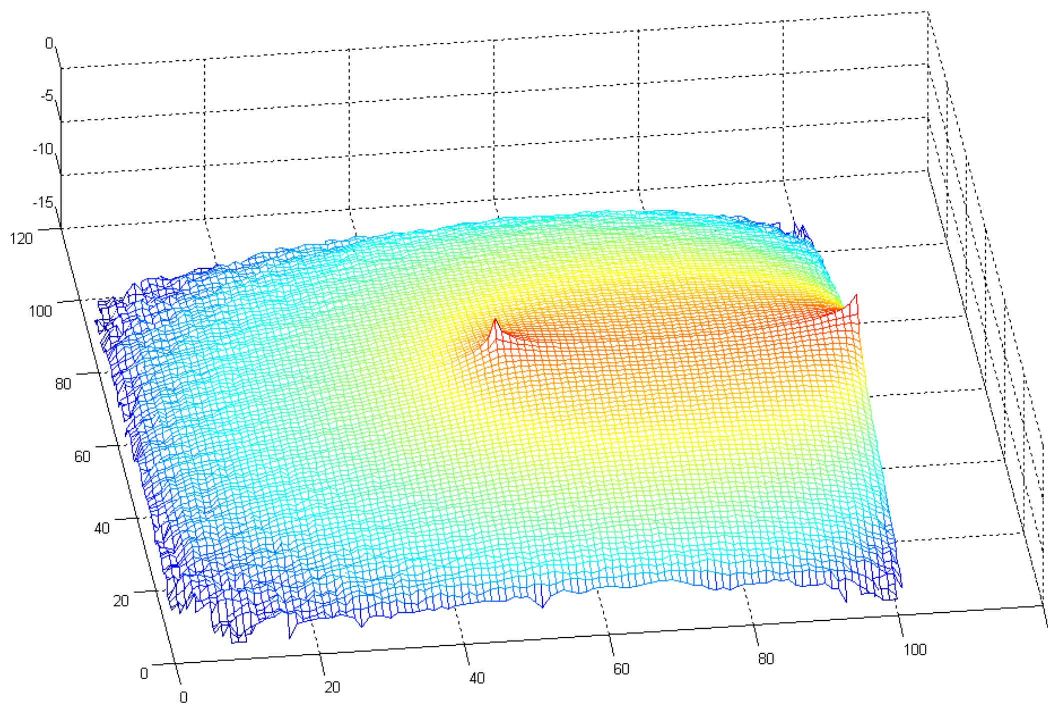


Fig. 34 The height is the cluster value of the possible shortest paths in a region ($p_{oc} = 0.4$). The path links from region centre to mid-point of the region border. The higher and brighter points have higher clustering value.

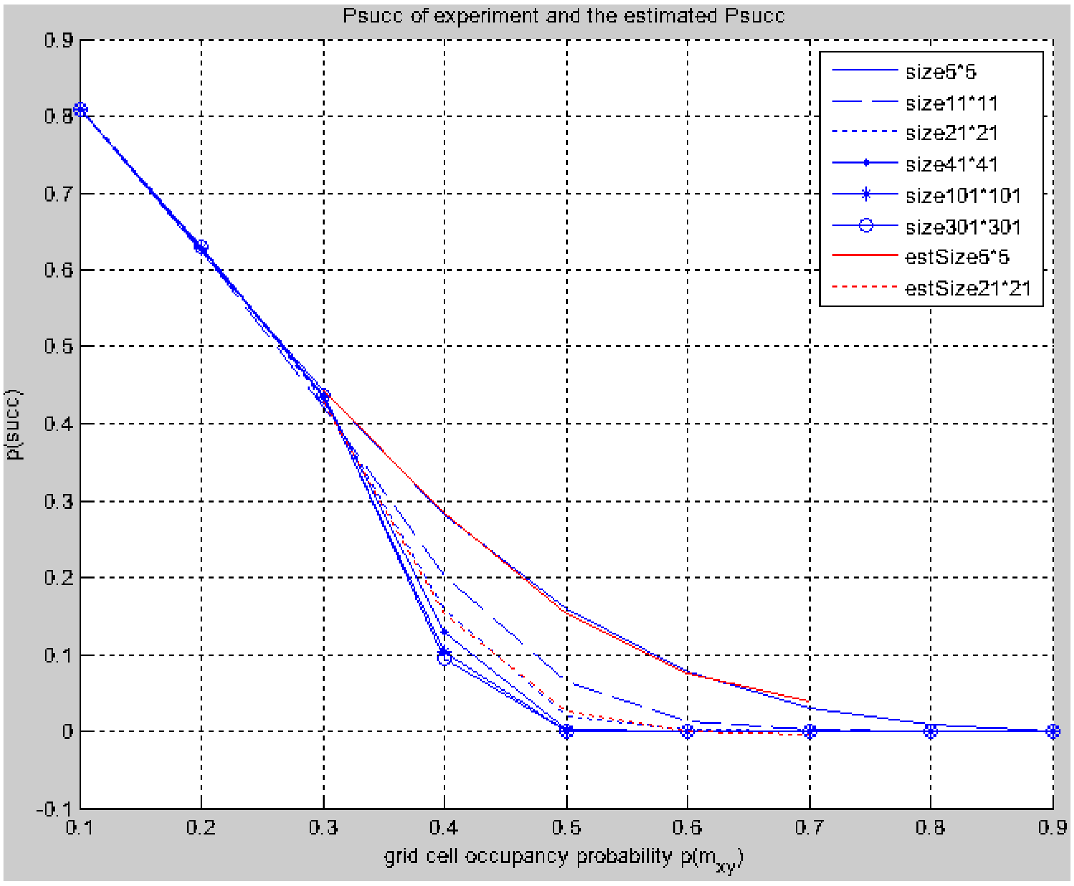


Fig. 35 Probability to reach a region centre with different $\boldsymbol{p(m)}$ and map sizes. Two estimate probabilities are shown for $\mathbf{0.3} \leq \boldsymbol{p(m)} \leq \mathbf{0.7}$.

In the HPD Planner's 1st step, we construct a region map and calculate the region occupancy probability $p_{region} = 1 - p_{succ}$ for each region (containing more than one grid cell) based on (26). This operation will affect the operations in the 2nd and the 3rd step of the HDP Planner.

Since we have the crossing probability, the region path can be smoothed to directly link the cells of entering and leaving a region, when we restore a region path to a grid cell path (then, the path may not go through the region centre) in the 3rd step of HDP Planner.

4.4 Experiment Results and Discussion

In the Hierarchical PD (HPD) Planner experiments, we use the OC Map Fig. 15 (used in PD Planner experiment) as the representation of the uncertain environment. We will compare the difference of HPD Planner and PD Planner.

In our experiment, we ran the original PD Planner, the PD Planner with SiPP strategy, and the hierarchical PD Planner with SiPP strategy with the same start and target cell in the same OC Map. Our simulator samples 3,000 possible maps based on M_{oc}. The robot can detect the environment within a 5-cell distance to the robot location and update the OC Map for re-planning when the current path is blocked. We record the total length travelled by the robot in these 3000 true maps. A Region Map example is shown in Fig. 37, where we marked out high clustering value regions. The experiment result is shown in Fig. 36.

Each algorithm was executed 1000 times to collect the average calculating times for finding a path. We select 3 pairs of start and target cells randomly. The experiment results are shown in Table 6, (the HPD Planner includes the Region Map constructing time). There are building structures (e.g. walls, rooms, corridors) that the A* heuristic may use, in the OC map Fig. 15. Therefore, the A* can check less grid cells to find the shortest path in map instances and the SiPP can save more time.

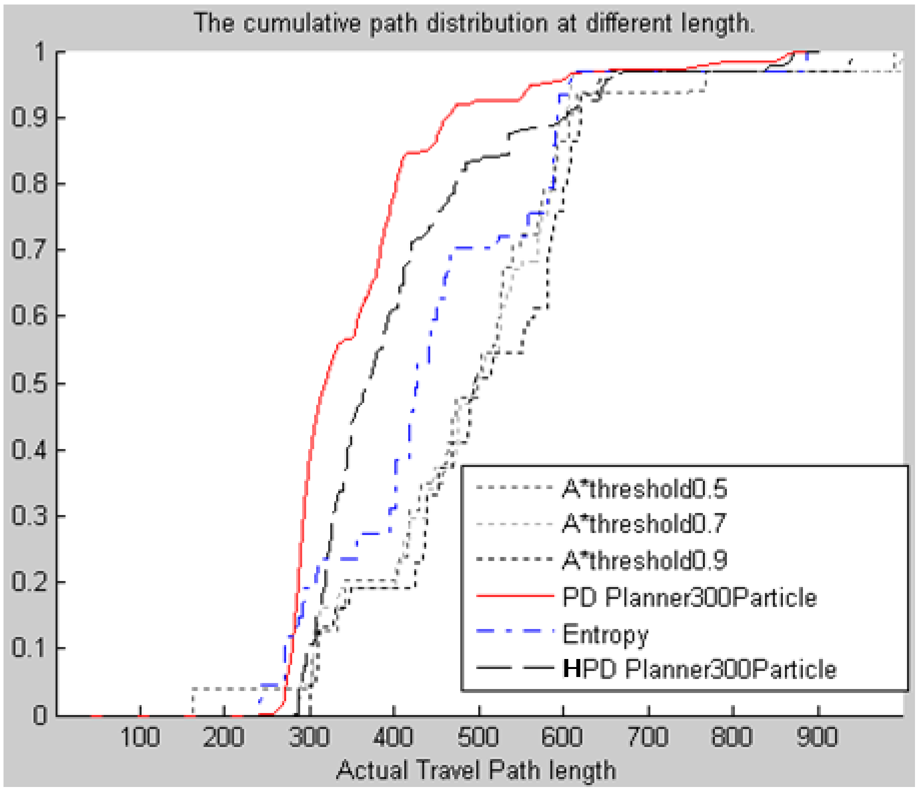


Fig. 36 The cumulative probability of reaching target

Table 6 Average planning time of different strategies.

	cell pair 1	Replan	cell pair 2	Replan	cell pair 3	Replan
A* $\beta = 0.9$	0.008s	0.115s	0.011s	0.114s	0.007s	0.117s
Original PD	2.237s	34.434	2.797s	33.857	1.908s	34.923
SiPP	1.465s	34.434	1.396s	33.857	1.299s	34.923
HPD Planner	0.975s	39.573	1.003s	38.733	1.092s	39.957

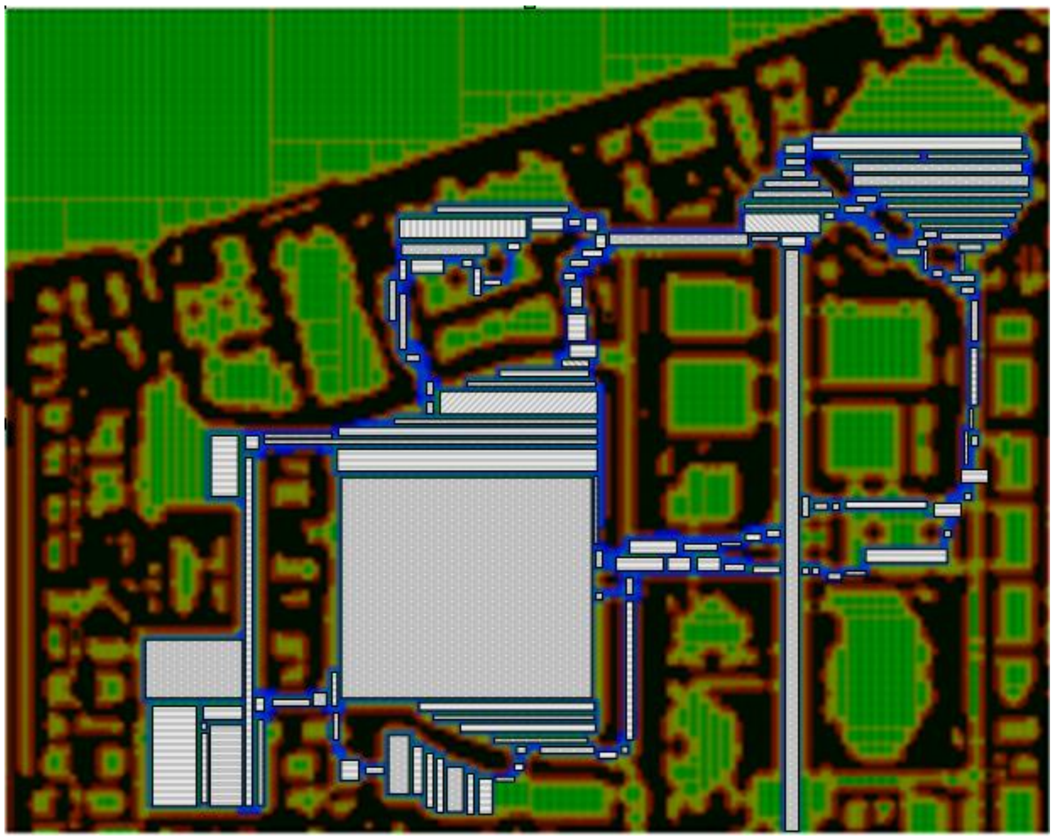


Fig. 37 Regions are marked with blue lines, darker means higher clustering value

4.5 Summary

In this chapter, we have discussed the problem of reducing the computational costs of the Monte Carlo sampling calculation through the use of two strategies – lazy sampling and hierarchical decomposition. We implement these strategies to reduce the computational cost of the PD Planner.

Lazy sampling is a very simple heuristic but is surprisingly effective. If the OC map has some typical precondition structures for A* to define a specific heuristic function, it can reduce even further. Furthermore, it is guaranteed that it will have no impact on the overall results achieved.

The hierarchical approach decomposes the map into a set of regions in which the occupancy probability is nearly constant. The properties of these regions have been cached in advance. The result is to reduce the overall dimension of the problem which reduces the computational cost in the A* planner. The resulting algorithm is about two times faster than original PD Planner. The shortest path passing a region may not be a straight line, because of the obstacles in the region. Therefore, the region path cannot describe the details as meticulous as the grid cell path planned by PD Planner. That is the reason why the performance of the HDP Planner is not as good as the original PD Planner. In our experiment, the HDP Planner increases the re-plan number about 15%.

Currently, the region path is planned by linking a region centre to its closest neighbour region centre. There are several avenues to extend the region types and the path of passing a region for future research. First, the Region Map currently uses just simple square and rectangle structures which might poorly describe the geometry of the environment. Therefore, we might define other structures, like an "S" arch or "T" corridor, to construct the Region Map in the future.

Second, the situations of $\Delta p > 0$ for constructing regions could be researched in the future, which can decrease the dimension of the searching space to a deeper extent (but increase the complexity of transferring occupancy probability to region occupancy probability).

Thirdly, although we have developed strategies to reduce the computational cost of each sampled map instance, we have not developed methods to reduce the overall number of map instances required for estimating PD map. Since the purpose of sampling is to develop a better characterization of the PD Map. One possible research interest is to analyse each POS based on OC Map structures (like a long corridor, a big room with several doors, etc.) and adjust POS's probability for estimating the PD Map. If we can increase the POS's probability for estimating the CV Map, by finding that it is lying in a higher clustering area, we will be able to estimate the PD Map with fewer particles.

5 Targets Allocation Planning for a Group of robots

5.1 Introduction

Chapter 3 discussed the issue of how to plan the trajectory for a single robot which has to move between two specified points in an uncertain environment. However, the rescue team could set multiple targets in the damaged building and ask a robot group to investigate all of these targets as soon as possible. In this chapter, we will consider the problem of how to make a plan for a robot group to search multiple targets in an uncertain environment. Here, the robot group contains more than one robot.

We will define the assignment and the allocation for the robot group in this chapter, by extending the PD Planner to the scenario of one robot visiting several targets, and multiple robots visiting multiple targets.

The PD Planner can plan a path, linking two points in an uncertain environment. For one robot to visit multiple targets, we will plan an assignment, which includes multiple paths for one robot to visit all targets one by one. We call the planner as the Assignment Distribution (AD) Planner. We want the actual travelled routes to be the shortest when all targets are visited.

Secondly, we extend the planner for the cases that the robot group has several robots. The planned result is an allocation, which includes the assignment for each robot in the group. We call the planner as the Group Allocation Distribution (GAD) Planner, which will decide how to assign multiple targets to each robot and plan the assignment for each robot to visit its targets. When all targets are visited, one robot will have the longest travelled route in the robot group. We want this longest travelled route to be the shortest. In fact, the GAD Planner is also suitable for the cases that there is only one robot in the group to visit several targets, or there is only one robot to visit one target. We introduce our research step by step to show the inside details of the clustering algorithms.

In the following part, we will introduce the multiple robot POMDP model. Then, we review the existing targets assignment algorithms, which are suitable for one or more robots. After we discuss their shortages in an uncertain environment, we introduce our proposals: the AD Planner and the GAD Planner.

5.2 POMDP Model for Multi-robot

The most complex scenario of our research is the situation of multiple robots visiting multiple targets. It is carried out in the scope of the Decentralized POMDP (Dec-POMDP) framework, which is an extension of POMDPs to the case of multiple robots. The Dec-POMDP is a very powerful extension of POMDP, and it enables us to model the interaction of two (or more) autonomous robots, which have to take individual actions based on their individual observations and common knowledge about the uncertain environment [36]. These individual observations may not represent the world accurately and actions may fail to have the intended effects. Our simulator supports the scenarios for multiple robots to visit multiple targets, which can be modelled with Dec-POMDP.

The Dec-POMDP is formally defined as the tuple $\{\mathbb{D}, X, \mathcal{U}, T, \mathbb{O}, R, h, b_0\}$ [36], where

$\mathbb{D}$ is a finite set of N robots

X is a finite set of possible states of the environment

$\mathcal{U} = \times_i U_i = U_1 \times U_2 \times U_3 \dots U_n$ is a finite set of possible joint actions for n agents, which is the Cartesian product of individual actions. u_i is the finite set of individual actions for agent i

T is the transition function, a mapping from states and joint actions to probability distributions over next states $T: X \times \mathcal{U} \longmapsto p(S)$

$\mathbb{O} = \times_i O_i = O_1 \times O_2 \times O_3 \dots O_n$ is a finite set of possible joint observations for n agents. O_i is the finite set of individual observations for agent i

O is the observation function, a mapping from the joint actions and the successor states to probability distributions over joint observations $O: \mathcal{U} \times X \longmapsto p(\mathbb{O})$

R is the reward function, a mapping from joint actions and current state to real number $R: X \times \mathcal{U} \longmapsto \mathbb{R}$

h is the horizon, the number of timesteps considered

b_0 is an initial belief (probability distribution over states) $b_0 \in p(X)$

At the beginning $(t = 0)$, the environment is at the state x_0. x_o is unknown to all robots, what all robots know is the prior belief b_0. At each step t $(t \geq 0)$, each robot i individually performs an action u_i^t which is unknown to the other robots.

The main difference from the single robot model is that each robot will take the other robots' states and actions into its own decision process. If the other robots' observations and actions are not available, one robot should use the estimate about other robots' actions instead. So, we have the following version of the reward equation for a multi-robot model.

Given the current state x^t and the multi-robot joint action $u^t = \{u_1^t, \dots, u_N^t\}$, the environment changes to a new state x^{t+1}, which is drawn from the distribution $T(x^t, u^t) \in p(X)$.The new state x^{t+1} remains hidden to all robots, but a multi-robot joint observation $o^{t+1} = \{o_1^{t+1}, \dots, o_N^{t+1}\}$ is drawn from the distribution $O(u^t, x^{t+1}) \in p(\mathbb{O})$. Each robot i receives its individual observation o_i^{t+1}, which is a part of o^{t+1}, and is unknown to the other robots. All robots receive their own rewards $r_i^t = R(x^t, u^t)$. At the next step, each robot i performs its individual action u_i^{t+1} and the whole process is repeated until step $t = h$, where h is the terminate condition. At this stage, the sum

$$r = \sum_{t=0,i=1}^{t<h,i=N} r_i^t \tag{27}$$

is calculated to represents the total reward obtained in the process, in which the sequence $\{x^0, u^0, x^1, u^1, \dots, x^{h-1}, u^{h-1}\}$ has occurred.

The goal of the Dec-POMDP planning algorithms is to assign each robot i an individual policy π_i, so that the multi-robot joint policy $\pi = \{\pi_1, \dots, \pi_N\}$ maximizes the total reward on average for each possible sequence $\{x^0, u^0, x^1, u^1, \dots, x^{h-1}, u^{h-1}\}$ [36].

The Multi-agent A* (MAA*) is a typical algorithm for solving Dec-POMDPs [37]. Its policy search tree is a joint (or partially joint) of the robots' policy searching trees. Each robot policy search tree is extended independently to find the shortest policy sequence for that robot using the A* heuristic function, but the evaluation of each search step includes all the policies of all the robots. In order to run faster, the author provides a sub-shortest strategy which only extends a robot's policy set if a child node has the same evaluation as its parent. In this case, other children nodes will not be checked in each search step. The computational costs of the A* search process and the heuristic for a single policy searching tree is the same as algorithms for POMDP.

5.3 Robot Group Allocation Algorithms Review

In our scenario, we have some target cells where the missing persons might be. The robot group allocation algorithm should decide how to allocate these targets to each robot, the targets' visiting sequence, and the shortest path to visit these targets.

The planning problem for one robot is similar as the Travelling Salesman Problem (TSP). The planning problem for multiple robots is similar to the multiple Travelling Salesman Problem (mTSP) [114]. However, there are typical differences in our planning problem scenario: 1) the sales men can start from any target location in the planning process, but we cannot change our robots' start points in one planning enquiry, 2) we are planning in uncertain environments where the actual travel paths are not known in advance.

If the robot group has more than one robot, we will focus on the robot with the longest path.

The robot group allocation research theories proposed by Furukawa and colleagues are introduced in a series of papers [83] [84] [85] [86]. An allocation includes the assignments for all robots in the group. The assignment for one robot contains several paths linking the robot position and targets assigned to the robot. Therefore, group shortest allocation has these characteristics:

- The shortest time to complete the mission is the longest time it takes any one robot to complete its assigned tasks (visit all its allocated targets).
- If we remove the robot with the longest paths and its allocated targets, the rest parts of the robot group also has the shortest allocation.

To find the shortest allocation, the path lengths of all possible allocations should be measured and compared.

Undeger [87] proposed an on-line allocation search algorithm (Real-Time Edge Follow - RTEF) in grid-type environments. RTEF [87][88] uses a heuristic algorithm (RTEF-ARM) to evaluate the global environmental information to decide which allocation segment, or direction, has greater probability to reach a target. Bachrach presented an online, forward-search algorithm for planning under uncertainty by representing the robot's belief of each target's pose as a multimodal Gaussian in

[89]. He exploited a parametric belief representation to directly compute the distribution of posterior beliefs after the actions are taken and enabled the robot to search more deeply by considering policies composed of multi-step action sequences [89].

The former allocation researchers do not take these cases into account:

- The actually travelled route in an uncertain environment between two points is different from the initial planned path, as we have shown in Chapter 3.
- When we make a plan, there might be one shortest clustering branch and several sub-shortest clustering branches between two points in an uncertain environment. The clustering branches among targets may be overlapped in some areas to increase the probability that the shortest allocation lying in the area, and shift the sub-shortest allocations to the shortest allocations.

In our search planning problem, we will make a plan for a robot group, $\mathrm{Q} = \{\mathrm{Q_i}\}, \mathrm{i} = \{1, \dots, \mathrm{N_Q}\}$, to visit several targets, $\mathrm{A} = \{\mathrm{A_j}\}, \mathrm{j} = \{1, \dots, \mathrm{N_A}\}$, where $\mathrm{N_Q}$ is the number of robots, $\mathrm{N_A}$ is the number of targets, and we have $1 \leq \mathrm{N_Q} \leq \mathrm{N_A}$. The uncertain environment includes both obstacles and free-space probabilities, described by an OC map $\mathrm{M_{oc}}$. Here, A_j corresponds a cell $m_{x,y}$ in an OC map M_{oc}. The probability of finding a missing person at A_j is $p(A_j)$.

In our simulator, we use a 2.5-dimensional environment representation and one belief node can hold several robots, because that they can fly at different altitudes. The rescue team members can give the targets based on different damages and their experiences, because the catastrophic event is unpredictable and meanwhile the effects of its damage are very hard to formulate exactly [1][22][24][26][27][28].

The path l links two destination cells. We define the shortest path as l^* that has the lowest cost $\mathrm{L}_{min}(l)$. In this chapter, we use the *path length* to evaluate the cost of a path (as we used in former chapters).

The targets allocated to robot Q_i is $\mathrm{S}_{\mathrm{Q_i,A}}$. The assignment for robot Q_i is defined as the collection of joined l^* to visit all targets in $\mathrm{S}_{\mathrm{Q_i,A}}$ in sequence, written as $\mathrm{L}(\mathrm{S}_{\mathrm{Q_i,A}}) = \{l_1^*, l_2^*, \dots\}$. Different assignment sequences have different lengths. The shortest assignment has the shortest length for Q_i to visit all its targets $\mathrm{S}_{\mathrm{Q_i,A}}$, written as

$$\mathbf{L}_{\boldsymbol{min}}(\mathbf{S}_{\boldsymbol{Q_i,A}}) = \mathbf{argmin}_{\mathrm{L}(\mathbf{S}_{\boldsymbol{Q_i,A}})} \left[\sum\nolimits_{\boldsymbol{l_i^*} \in \mathbf{L}(\mathbf{S}_{\boldsymbol{Q_i,A}})} \mathbf{L}_{\boldsymbol{min}}(\boldsymbol{l_i^*})\right] \tag{28}$$

The target assignment planning is specified as follows:

1, We exclusively and exhaustively assign targets to one robot in different orders.

2, We use the shortest paths (planned by PD Planner) to link these target one by one to construct the routes for different assignments. The shortest assignment is the one with the shortest route length among all possible assignments.

The group allocation which assigns all the targets to Q, is $G(Q,A) = \left\{\mathrm{L}(\mathrm{S}_{\mathrm{Q_1,A}}), \mathrm{L}(\mathrm{S}_{\mathrm{Q_2,A}}), \dots, \mathrm{L}\left(\mathrm{S}_{\mathrm{Q_{N_Q},A}}\right)\right\}$. The allocation cost is the max cost of the assignments in $G(Q,A)$, because the mission completion is decided by the robot with the max assignment length in an allocation. The shortest allocation $G^*(Q,A)$ has the lowest cost:

$$\mathrm{L}_{min}[\mathrm{G(Q,A)}] = \underset{\mathrm{G(Q,A)}}{\mathrm{argmin}} \left\{ \underset{\mathrm{L_{min}(S_{Q_i,A}) \in G(Q,A)}}{\mathrm{argmax}} \left[\mathrm{L_{min}}(\mathrm{S}_{\mathrm{Q_i,A}})\right] \right\} \tag{29}$$

The target allocation planning is specified as follows:

1, We exclusive and exhaustively allocate targets to the robot group. There will be many different possible allocations. Each target will only be allocated to one robot and the union of all assignments $Q_{i,A}$ must cover the targets set A.

2, Each robot is allocated several targets in one possible allocation, and we find the shortest assignment for each robot (using the assignment planner).

3, The allocation length is the longest length of the shortest assignments in the allocation. The shortest allocation is the allocation with the shortest length among all possible allocations.

All the robots start positions can be set by the user in the simulator. They may be launched from the same cell $m_{x,y}^{start}$, or different cells in an OC Map. The cooperation is unpredictable, because it is hard to define the wireless communication model in an uncertain environment [72]. Therefore, we use a simple strategy in our simulator: a single robot does not exchange information with other robots if it makes a local re-plan; the robot group will exchange information and make a new assignment plan if one robot requires a global re-plan.

The total travelled paths of robots are affected by the searching sequence of the targets and the detected obstacles when the robots are moving. The main challenge of our research is that the robot actual travelled route from its initial start position to the target position is a combination of the initial plan and several re-plans. Our research aims at the shortest actual travelled route for the Complete Mission Process (CMP).

Here, we illustrate this problem for a known environment first, then we introduce the research problem in an uncertain (partially known) environment.

5.3.1 Search Planning in a Known Environment

The simplest scenario is the search planning problem in which everything is known exactly [81][82]. For example, in the two-dimension map shown in Fig. 38, the obstacles, targets and robot start positions are known exactly, including the positions of the obstacles and the targets. In this figure, the black squares are obstacles, three targets are marked with a cross (labelled “A_1” “A_2” “A_3”), and the start position for the robots is marked with an arrow. The search planning problem is to find the shortest assignment so that each target is visited by at least one robot, and the time to visit all targets is. If we assume that the robot moves with a constant velocity, the minimum time equals to the shortest path length.

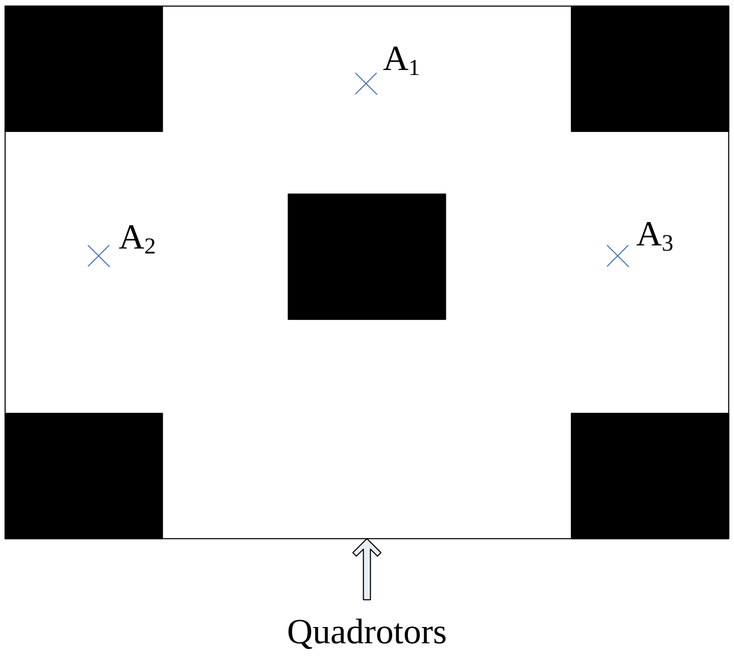


Fig. 38 An example of everything is known exactly.

The number of all possible assignments is exponential in the number of targets and robots, ${N_A}^{N_Q}$. By evaluating all possible assignments, we can find the shortest solution. However, this is an NP-complete problem [90] and we cannot exhaust all possible assignments in most cases.

The group allocation algorithms are suitable for the case that the group has only one robot. So, we do not introduce the one robot group algorithms review separately.

One of the earliest approaches is a naïve greedy method to assign the targets, called the Target Equalization (TE) method [91]. The algorithm attempts to assign the same number of targets to each robot. The algorithm pseudo-code is shown in Pseudo-code 9.

Pseudo-code 9 Target Equalization function

```
TargetEqualization_Func(TargetSet: $A$, UAV Set: $Q$)
1  Init the assignment $S_{Q,A}$ as an empty assignment
2  while $A$ is not empty
3      Assign each UAV $Q_i$ an available target which is nearest to $Q_i$.
4      Resolve conflicts (one target is assigned to several UAVs) in
favour of the quadrotor with lower cost.
5      Update $S_{Q,A}$ and quadrotors' positions to their last assigned targets
6      Remove these assigned targets from the target set $A$.
7  end while
8  return the assignment $S_{Q,A}$
```

Its main loop will be executed $\frac{N_A}{N_Q}$ times. TE addresses this problem in a greedy way: the robot(s) will always move to the closest target. It is clear that this method is a greedy assignment and will not guarantee to find the best solution. For example, in Fig. 39, the best assignment is: the robot Q_2 goes to the fastest target A_3 and the robot Q_1 goes to the closer two targets (A_1 A_2); but the TE will assign all targets to Q_1. However, it can be used to provide a baseline to evaluate other methods.

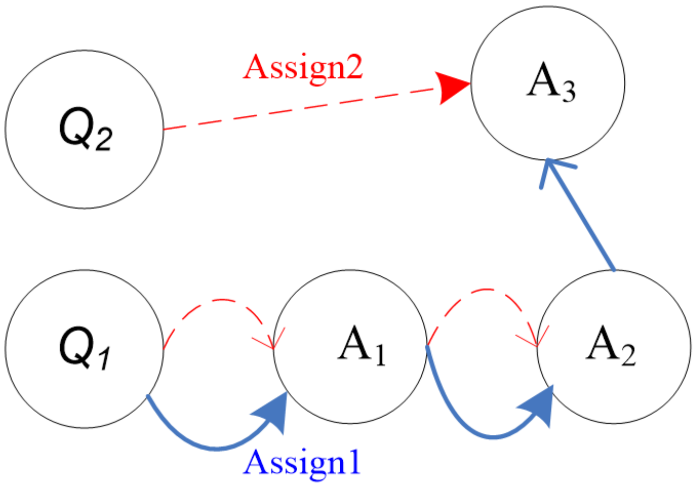


Fig. 39 The TE assignment is Assign1, the best assignment is Assign2.

The Path Equalization (PE) method takes the path length comparison into the assignment process to avoid the case in which some robots have a very long path while others end up with very short paths. Its process pseudo-code is shown in Pseudo-code 10.

Pseudo-code 10 Path Equalization function

```
PathEqualization_Func(TargetSet: $A$, UAV Set: $Q$)
1  Init the assignment $S_{Q,A}$: assign one target to each UAV like TE method.
2  while $A$ is not empty
3      Choose the UAV $Q_i$ with the shortest cumulative path $L_{min}(S_{Qi,A})$ so far
4      Assign the closest available and free target to $Q_i$, and remove the
target from $A$.
5      Update the cumulative path length for the $Q_i$.
6  end while
7  return $S_{Q,A}$
```

This method chooses the next robot for assignment based on a current assignment, rather than in a fixed order through all robots. It attempts to balance the path length among the robots [91]. Its main loop will be executed N_A times. However, all assignment decisions are made locally without a global view to assign targets [90].

Assume that we have 3 robots and 30 targets. When using the PE method, we find that the options available to robots decrease with the number of available targets, as the assignment process goes on. This is not a problem when the number of available targets is still large, but in the following stages of the assignment process, it can lead to a situation where a robot with the shortest cumulative path at the time of selecting the next target has much poorer options than another robot with a somewhat longer cumulative path [90]. For example, we have three robots: Q_1, Q_2, Q_3. The targets assigned to Q_1, Q_2 mean that the nearest target to Q_3 forces it to dramatically and irreversibly increase its path length so that the assignment leads to a drastic and irreversible imbalance in path length.

In [90], a two-stage assignment process is proposed to solve the PE problem. A time-varying parameter $\xi(t)$ is defined at step t, as: $\xi(t) \equiv \frac{N_{available}(t)}{N_Q}$, where

$N_{available}(t)$ is the number of targets still unassigned at step t. The two-stages are governed by a threshold θ_{ξ}.

Stage I: When $\xi(t) \geq \theta_{\xi}$, any selected robot is likely to have many options for its next assignment, and the PE algorithm is followed.

Stage II: When $\xi(t) < \theta_{\xi}$, many robots are likely to have very limited choices, and the simple PE approach could lead to poor assignments. In this case, the assignment pseudo-code is shown in Pseudo-code 11.

Pseudo-code 11 Function of 2 stage assignment algorithm

```
StageII_Func(TargetSet: $A$, UAV Set: $Q$)
1   $\lambda^*=Z$, where $Z$ is a very large number; $i^*=1$
2   Until an assignment is accepted or $A$ is empty
3      Pick the $Q_i \in Q$ with the shortest cumulative path length, $\lambda_i(t)$.
4         Calculate the Euclidean distance of the nearest available target
          for current assignment of $Q_i$, and calculate the resulting
          updated path length, $\lambda_i^+(t)$, if $\lambda_i^+(t)<\lambda^*$, set $i^*=i$.
5         If $\lambda_i^+(t)<max_{j\in Q}\lambda_j(t)$, i.e., the resulting updated path would not
                exceed the longest current path,
6            accept the assignment, go to line 11
7         else
8            remove $Q_i$ from $Q$
9         end if
10     Repeat to line 3, until $Q$ is empty
11     if (no assignment was made in line 3~10)
12        assign the nearest available target to $Q_{i^*}$ and update $max_{j\in Q}\lambda_j(t)$
13     end if
14  Repeat to line 2
15  return the assignment
```

The process can be seen as a combination of assigning many targets to robots (limiting the search choices) and assign several targets to robots (increase the search choices). It is better and computationally more feasible to consider several — even all — robots for each update rather than myopically picking out the one with the shortest path so far [90]. If no good choice is available, the least bad choice is made on line 3 of pseudo code of the stage II (see above). The planned assignment is acceptable because the longest path is shorter than the threshold Z.

Maddula [90] proposed a refinement algorithm to improve the assignment solution. He defined a concept called a Potential Point of Exchange (PPE) as follows. Suppose path P_i is currently assigned to Q_i and P_j is currently assigned to Q_j. If targets $v_1 \in P_i$ and $v_2 \in P_j$ are such that the distance between them is less than a threshold, $\|v_1 - v_2\| < \theta_e$, we term $[v_1, v_2]$ a potential point for exchanging targets between the robots [90]. Based on this, he defined four exchange operators:

- Operator 1: The sub-paths of P_i and P_j starting at v_i and v_j linking their last targets, respectively, are exchanged.
- Operator 2: The sub-paths of P_i and P_j, including the segments terminating at v_i and v_j linking their start positions, respectively, are exchanged.
- Operator 3: The segments of P_i and P_j starting at v_i and v_j, respectively, are exchanged.
- Operator 4: The segments of P_i and P_j ending at v_i and v_j, respectively, are exchanged.

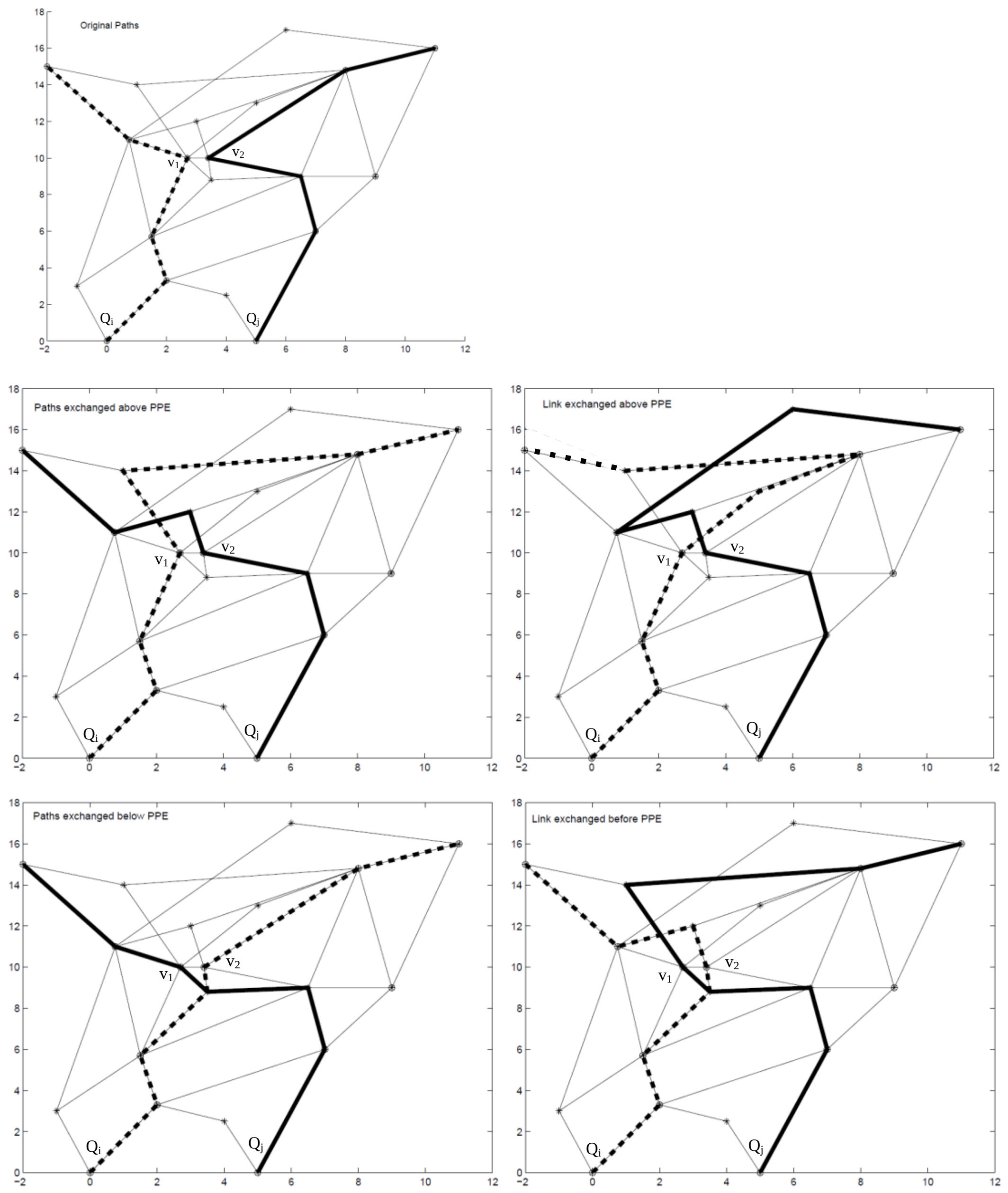

Original Paths
Paths exchanged above PPE
Link exchanged above PPE
Paths exchanged below PPE
Link exchanged before PPE
v_1
v_2
Q_i
Q_j

Fig. 40 The initial assigned paths and their exchanges[90].

The exchange operators are illustrated in Fig. 40, where the first graph shows two assigned paths (one dot line and one solid line) and the potential routes are shown as the grey lines. The next four graphs show the effect of applying each of the exchange operators to the original paths. Not all operators produce better options, so worse assignments than the original one will be rejected. Operator 1 and Operator 2 are the main assignment searchers, and they exchange whole sub-paths. Operators 3 and 4 mainly act to remove "kinks" in existing pairs of paths, where the paths cross and then cross again within a short distance [90]. The paths exchanging requires a local search on the map. If the exchange does not work (because the path segments cannot be reconnected), it will be rejected automatically.

The procedure begins with the assignment produced by one of the above algorithms, and iterates over the following procedure until a stopping criterion is met. The exchange operation pseudocode is shown in Pseudo-code 12.

Pseudo-code 12 The Potential Point of Exchange function

```
PPE_func(CandidatePaths: $P_i$, $P_j$)

[$v_i$, $v_j$] = find Potential Point of Exchange(PPE) of $P_i$ and $P_j$
init new path [$NewP_i$, $NewP_j$] = [$P_i$, $P_j$]
for(i=0; i<4; i++)                    //try four exchange operators
    [$P'_i$, $P'_j$] = exchange_operator(i, $P_i$  $P_j$)
    if ([$P'_i$, $P'_j$] better than [$NewP_i$, $NewP_j$]
        [$NewP_i$, $NewP_j$] = [$P'_i$, $P'_j$]
    end if
end for
[$P_i$, $P_j$] = [$NewP_i$, $NewP_j$]           //record the exchanged new paths
```

In a word, the algorithms described above do not compare all possible assignments. If we could select the PPE algorithm carefully, the assignments closer to the shortest assignment will be evaluated in the planning process. They define an algorithm to refine the basic assignment and using stopping criteria, or a limit on the number of iterations, or a limit calculation time, to finish the refine operations.

5.3.2 Search Planning in Uncertain Environment

We hope to solve the more general problem that the targets lying in an uncertain environment with a probability to find a missing person (survivor), as illustrated in Fig. 41. In this case, the targets are marked by clouds, which indicating the probability to find a survivor. The robots need to go through the uncertain areas (the occupancy probability is shown with cloud patterns) and visit the targets to locate the survivor(s). Generally, each search target has a probability of containing survivor(s), $p(A_i), i \in \{1,2,\dots,N_A\}$.

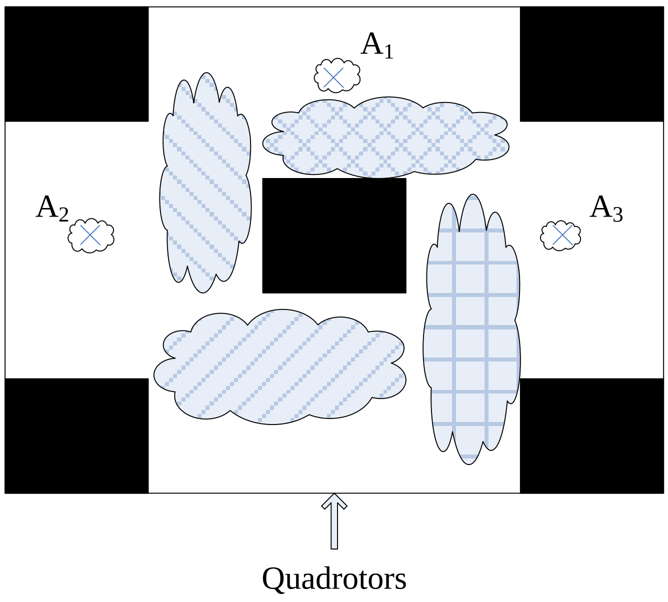


Fig. 41 The targets' positions are uncertain.

The search planning objective is to allocate the targets to the robots, which is similar to the discussion above. The robot group should visit all targets with the shortest travelled path length.

Lau [92] proposed an algorithm which aimed to minimize the average time for finding a survivor given an unknown number of targets, by deciding the search sequence of the target areas based on the value iteration function. He models the environment as a graph, where the nodes are the search areas and the edges are the paths linking two immediate neighbour searching areas together. The algorithm normalized the probability, $p_i = \frac{p(A_i)}{\sum_{j=1}^{N_A} p(A_j)}$, where $i \in \{1,2,\dots,N_A\}$. We have $\sum_{i=1}^{N_A} p_i = 1$ and p_i can be looked as the normalized probability of the remaining targets in the search area A_i.

Lau uses Dynamic Programming Equations (DPEs) to evaluate which area is the best for searching next. The algorithm has three steps, and the pseudo-code is shown in Pseudo-code 13.

Pseudo-code 13 Dynamic Program Equation function

```
DynamicProgramEquation_Func()
1  Initialize the value function $V_0$, for example choose
$V_0\{x,p_1,p_2,\ldots,p_{Na}\}=0$ for $x, p_1, p_2, \ldots, p_{Na}$
2  Compute function $V_{i+1}$ using the dynamic programming (DP)
recursion for $(p_1,p_2,\ldots,p_{Na})\neq(0,0,\ldots,0)$, using the iteration function to
calculate $V_{i+1}\{x,p_1,p_2,\ldots,p_{Na}\}=f\_iteration$
3  Stop when $V_{i+1}\{x,p_1,p_2,\ldots,p_{Na}\}=V_i\{x,p_1,p_2,\ldots,p_{Na}\}$, for all
$x,p_1,p_2,\ldots,p_{Na}$
```

The iteration function in line 2 is shown bellow:

$$V_{i+1}\{x,p_1,p_2,\ldots,p_{N_A}\} = f\_iteration$$

$$= \min[T_x + q_x V_i\left(x,\frac{p_1}{q_x},\ldots,\frac{p_{x-1}}{q_x},0,\frac{p_{x+1}}{q_x},\ldots,\frac{p_{N_A}}{q_x}\right), \min_{j\in E(x)}\{V_{xj} + V_i(j,p_1,p_2,\ldots,p_{N_A})\}] \tag{30}$$

where $q_x = 1 - p_x$,

$E(x)$ is the set of all direct adjoining areas to A_x

T_x is the time to search area A_x

V_{xj} is the evaluation that robot move from area A_x to area A_j.

The final function $V_i\{x,p_1,p_2,\ldots,p_{N_A}\}$ satisfies the form of a Dynamic Programming Equation (DPE) and is the value function that we seek.

The main calculation of the algorithm is step 2, where the minimal expected average time is equal to the lesser of (i) the minimal expected time if the robot chooses to search in the area where it is, equalling the search time T_x plus the minimal expected time calculated from when the search is finished, and (ii) the minimal time for the robot to move to any area $j \in E(x)$ plus the minimal expected average time calculated from when the robot is in that new area A_j [92]. The limitation of this algorithm is that it has the exponential scaling as before. But there is a further cost associated with the searching regions: (i) the choice of x will increase as the number of searching areas increases; (ii) the set of all direct adjoining areas $E(x)$ becomes bigger when the connectivity of the search areas increases.

This algorithm can be used to find the shortest search sequence for a single robot when a target assignment is given to that robot. Based on the result, uncertain target search planning can be compared with different target assignments to find out the shortest assignment for the robot team. In reality, we can use the PE, or two-stage algorithm to generate the baseline, and the refinement algorithm can provide the key search areas for DPE to find the shortest sequence faster.

5.4 Assignment Planning

One important limitation of the algorithms described above is that they expect the actually travelled route to be the same as planned, which is almost impossible in an uncertain environment, as shown in Chapter 3. The partially known environment cannot guarantee that the single planned assignment can be executed without collision. After the robots visited all targets, the actually travelled routes are the combination of many assignment plans. Therefore, the algorithms in section 5.2 cannot guarantee that the robots actually travelled routes are the shortest.

In this section, we will introduce how to undertake target assignment planning, after we know how to make a plan for moving between any two points in an uncertain environment. The assignment planning should take the path planning between any two points and the probability of finding targets into account. In an uncertain environment, the planned assignment should combine some different assignments to deal with the uncertainty. The travel cost and the search efficiency are balanced in the planned assignment for robot to execute.

To solve the assignment planning problem, we will find where the possible shortest assignments tend to cluster and plan the assignment based on the clustering. This means that the assignment is evaluated based on the clustering of many possible shortest assignments. Compared with other algorithms, the clustering also leads to decrease the re-plan probability. Our aim is to make the actual travel length has the highest probability to be the shortest one.

In this section, we introduce the algorithm for single vehicle multiple target planning. This is the special case that there is only one robot Q_1 in the robot group. The planner must find the shortest assignment $\mathrm{L}_{\min}(S_{\mathrm{Q}_1,\mathrm{A}})$ which consists of the order that the targets are to be visited, and the trajectory the robot will take to visit these targets. We call the algorithm the Assignment Distribution (AD) Planner.

Similar to the PD Planner, the AD Planner has two parts: calculating/estimating the Assignment Distribution (AD) Map, and using this map to find the shortest assignment. However, the assignment includes the targets' order, and some waypoints/grid cells may be revisited in an assignment. Therefore, the CV in AD Planner is more complex than the CV in PD Planner.

5.4.1 Calculating/Estimating Assignment Distribution (AD) Map

The AD Map is designed for single vehicle multiple target planning, which provides the spatial distribution of the set of shortest assignments.

Given M_{oc} with N grid cells, there are many possible map instances $M^i, i \in \{1,2,3,\dots,2^N\}$. There is only one robot in the group, so the target assignment is $S_{\mathrm{Q,A}}$. The shortest assignment in M^i is written as $\mathrm{L}_{\min}(S_{\mathrm{Q}_1,\mathrm{A}}|M^i)$, which is found by applying an MDP assignment planning algorithm. Each $\mathrm{L}_{\min}(S_{\mathrm{Q}_1,\mathrm{A}}|M_\mathrm{i})$ is a Possible Optimal Assignment (POA).

We use A* to find all shortest paths between any two points among robot initial point and targets in M^i. Then, we evaluate all possible different orders of targets for a robot to move. Each order of targets may involve different path length. The planned shortest assignment is the one that the robot will travel the shortest distance to visit all targets.

Each $\mathrm{L}_{\min}(S_{\mathrm{Q,A}}|M^i)$ is a POA for a single robot to visit multiple targets. The spatial distribution over many POAs contains a high degree of structure, which can be exploited for planning. The areas where the POAs tend to cluster are where the shortest assignment is the least sensitive to the uncertainties tends to coincide. The set of all POAs is superimposed onto a single map, which is called as the Assignment Distribution (AD) map M_{AD}. Each grid cell in M_{AD} records the probability that the shortest assignment will actually pass through that cell. Therefore, we plan an assignment which maximizes the probability that it will stay in the regions where most POAs cluster. Marginalising over map instances, the probability of the shortest assignment passing through the cell $m_{x,y}$ is the POA clustering value to the cell, given by

$$P_{\mathrm{AD}}(m_{x,y}) = \sum_{M^i \in M} \{p[m_{x,y} \in \mathrm{L}_{\min}(\mathrm{S}_{\mathrm{Q}_1,\mathrm{A}}|M^i)]P(M^i)\} \tag{31}$$

Generally, it is impossible to exhaustively compute this over all possible map instances, and so we use a Monte Carlo sampling strategy (the same as we do with PD Planner). A large number of map instances are randomly drawn, an assignment planner in certain environments is executed, and the AD Map computed empirically from (31).

The PD Planner clusters the paths to each cell when planning a single path. In the PD Planner, one planned path cannot visit one grid cell more than once. The AD Planner extends this by clustering the POAs to each cell for planning an assignment. The paths in one POA indicate where the POA is lying. Therefore, we count the POAs that pass one cell and do not count the paths that passing through the cell in AD Map. Although, the shortest assignment $\mathrm{L}_{\min}(\mathrm{S}_{\mathrm{Q,A}}|M^i)$ can visit a grid cell $m_{x,y}$ several times.

Fig. 42 overlays three POAs computed from three different realizations of the M_{oc}. The AD map for the scenario in Fig. 42 is shown in Fig. 43. Despite the complexity of the original environment, the AD map is relatively sparse and contains just a few well-defined paths. The AD Planner uses these well-defined paths to plan its shortest path.

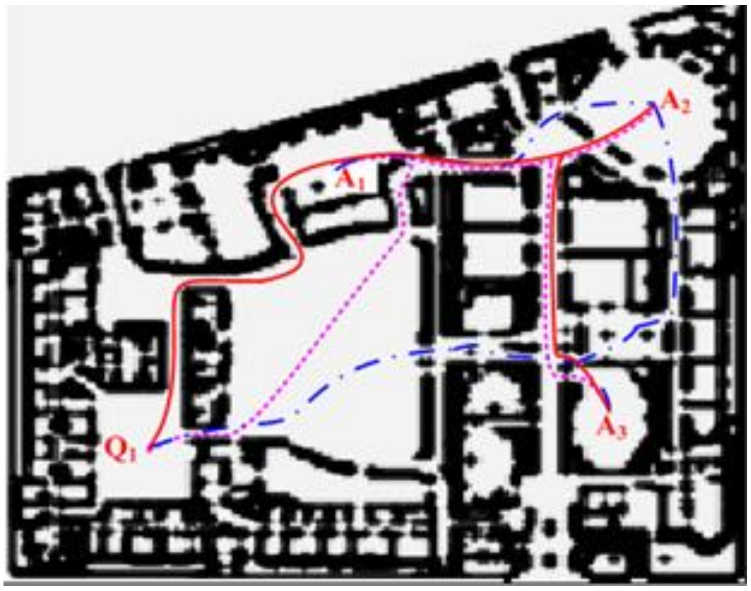


Fig. 42 Three POAs arise from different realizations of the OC map. The robot starts from Q_1 and must visit $\{A_1, A_2, A_3\}$. The POAs are shown in red, pink and blue. The OC Map is the background. Darker cells have higher occupancy probability.

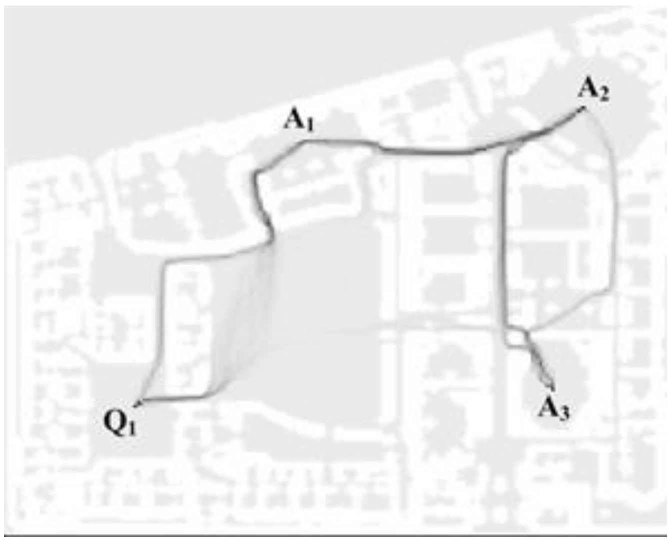


Fig. 43 An example of AD Map (1 robot and 3 targets), with OC Map as the background. The darker cells have higher P_{AD}, where POAs tend to cluster..

Here, we construct a two-floor OC Map as another example to show what the AD Map looks like. In Fig. 44, there are two floors: (a) is the upper floor and (b) is the lower floor, and the four targets are marked as black cells. There are three connections between these floors. This two-floor plan was chosen because the building has many narrow corridors, and the alternative routes, including travel from one floor to another floor when we make an assignment plan. The grid cells' occupancy probabilities are shown in green colour (the darker the higher occupancy probability). The red dots in the grid cells are where the assignments cluster. The pink line is the planned shortest assignment. We will describe the algorithm of assignment planning with AD Map in the next section.

We show the clustering value of an AD Map in Fig. 45, where the brighter area has the higher clustering value. Despite of the complexity of the environment, the brighter lines illustrate that there is only a relatively small number of cells that the paths cluster around.

In Fig. 46, two targets marked with "1" and "2" for a robot to visit. One possible shortest assignment is $b_{start} \rightarrow b_2 \rightarrow b_1 \rightarrow 1 \rightarrow b_1 \rightarrow 2$, which revisit the node b_1. Another possible shortest assignment is $b_{start} \rightarrow b_2 \rightarrow b_1 \rightarrow 1 \rightarrow b_3 \rightarrow 2$, which has the same segment $b_{start} \rightarrow b_2 \rightarrow b_1 \rightarrow 1$ with the former assignment.

An assignment in M^i is constructed with N_A shortest paths. As shown in Pseudo 14, all paths are saved in the cost matrix S_{path} and we use the dynamic programming [115] to make sure that each path is planned once. L_{min} is the shortest assignment among those exhausted assignments. We use L_{min} as the upper bound of branches to cut hopeless branches [116]. The searching process is a typical mTSP recursive function to force exhaust all possible assignment branches. L_{cur} is the exhausting assignment branch. The computational cost is $O(n^2 2^n)$ [116] (where n is the number of targets). In line 6 of Pseudo 14, if l can connect the last target in L_{cur} to another target not in L_{cur}, we extend L_{cur} by adding l to L_{cur}. In line 8, if L_{min} is longer than L_{cur}, we call this recursive function to extend L_{cur} further. This recursive process repeats until all targets are assigned, and we replace L_{min}

with the new-found assignment L_{cur} (line 1~4). So, L_{min} will be the global shortest assignment among all exhausted assignments when the recursion process finishes.

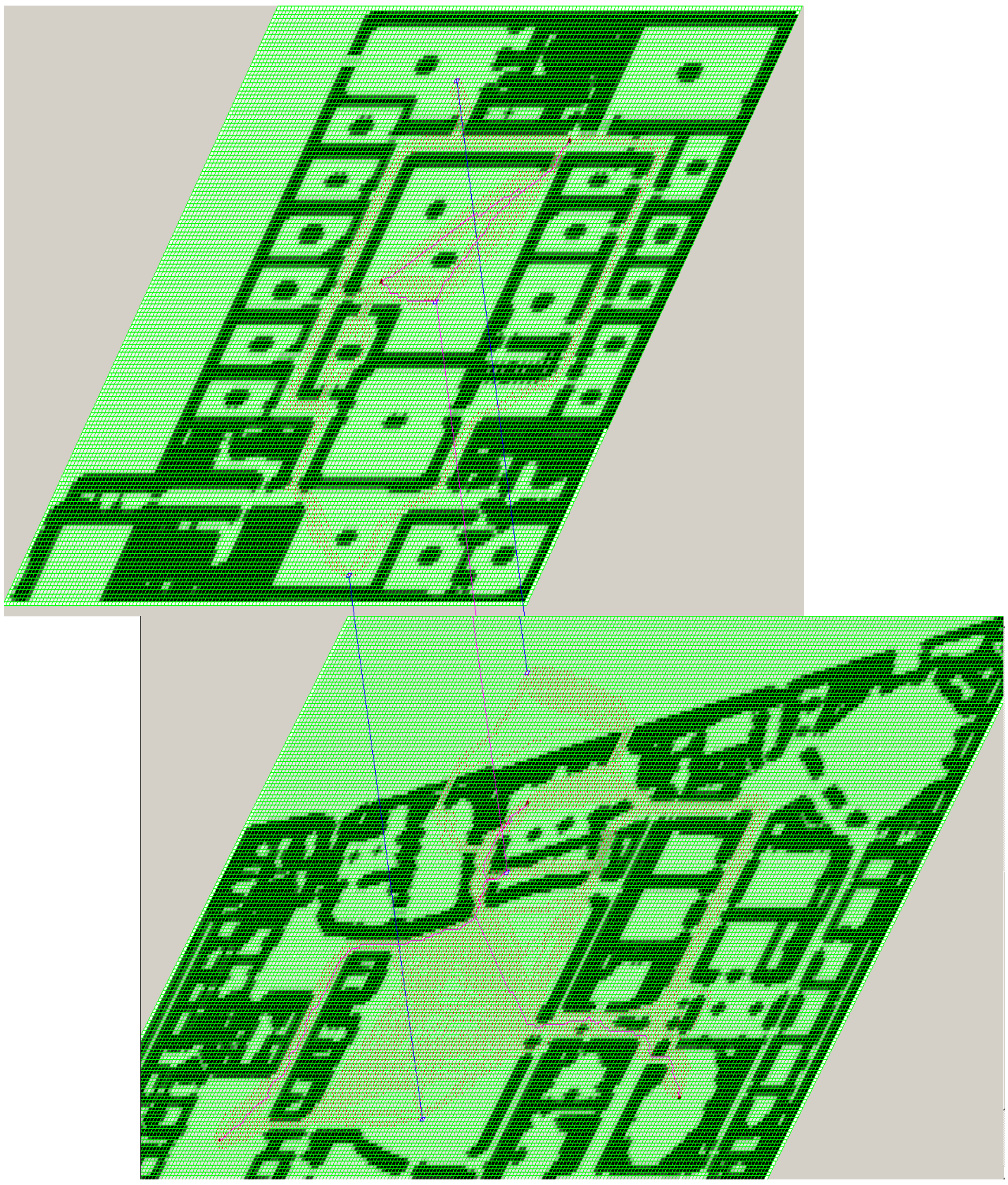


Fig. 44 The Assignment Distribution in a two-floor OC Map.

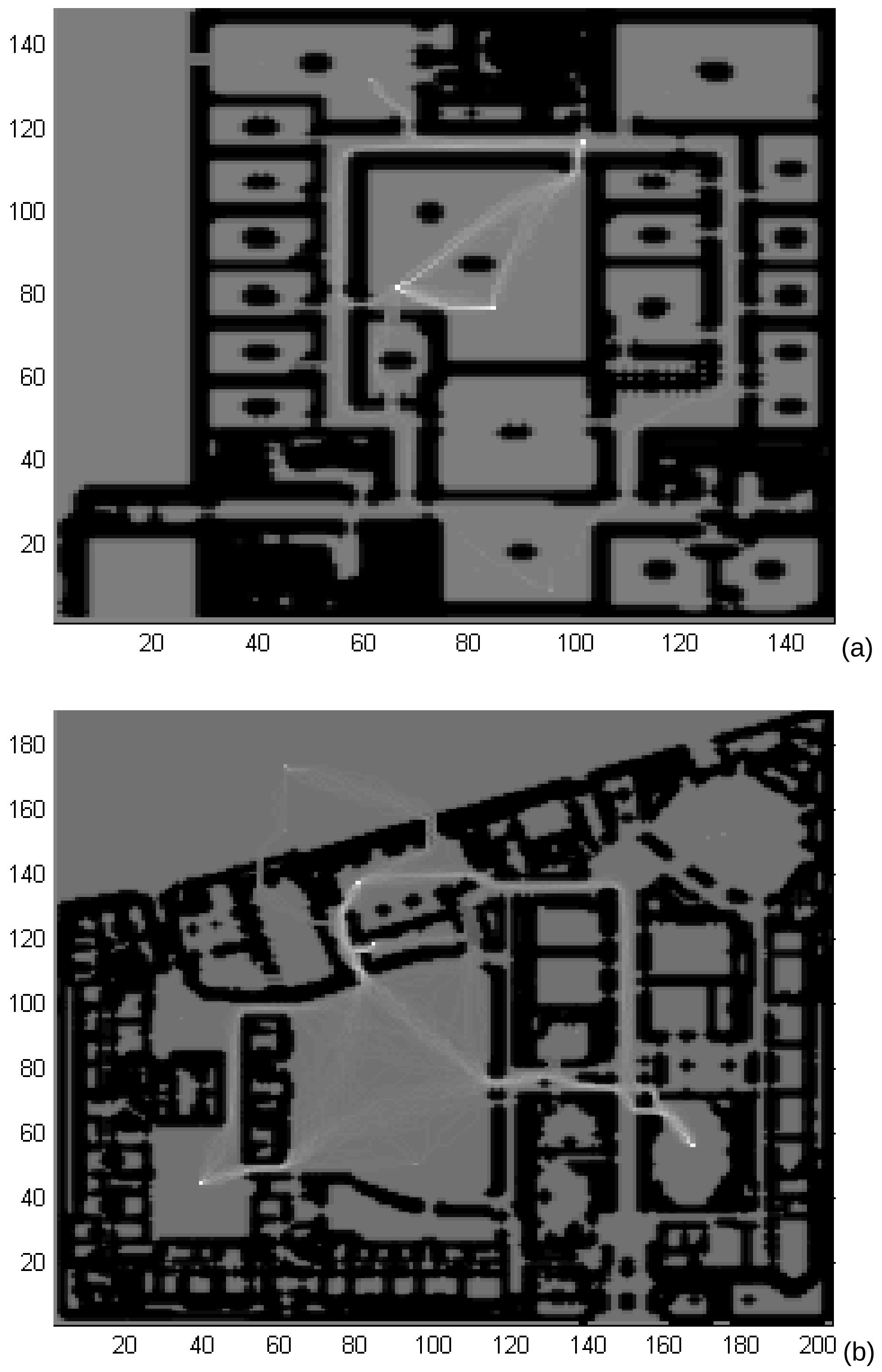


Fig. 45 The Assignment Distribution(AD) Map example

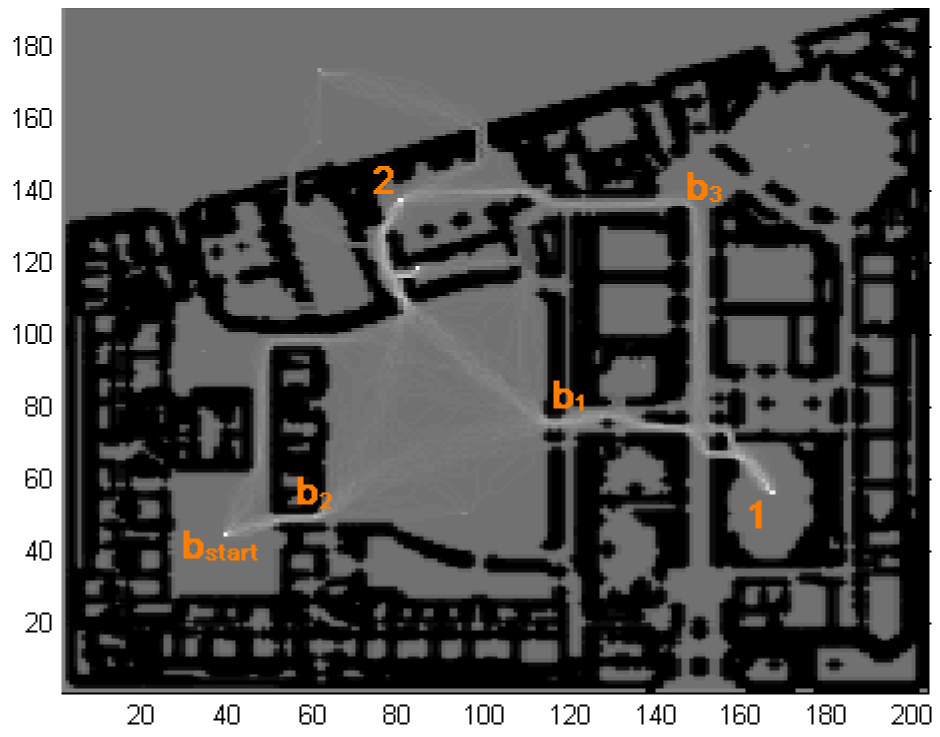


Fig. 46 The assignment clustering example with OC Map as background. The POAs go through the brighter cells more than the dark cells. Three key nodes b_1, b_2, b_3 are used to mark the clustering routes.

Pseudo 14 The recursive function "OptAssignment" to find the shortest assignment with all paths set.

OptAssignment(PathSet:S_{path}, Targets:A, CurAssignment:L_{cur}, CurOpt:L_{min})

```
1   if ($L_{cur}$ has $N_A$ paths)    //all targets are visited by $N_A$ paths in assignment $L_{cur}$
2       $L_{min} = L_{cur}$
3       return;
4   end if
5   for($l$ = path in $S_{path}$,)
6       if ($l$ connects the last target in $L_{cur}$ and one target not visited in $L_{cur}$)
7           add $l$ to $L_{cur}$
8           if ($L_{min}$ longer than $L_{cur}$)    //skip assignment $L_{cur}$ if it is longer than $L_{min}$
9               call OptAssignment($S_{path}$, A, $L_{cur}$, $L_{min}$)
10          end if
11          remove $l$ from $L_{cur}$
12      end if
13  end for
```

5.4.2 Planning the Assignment Path with the AD Map

Similar to the PD Map, the AD Map describes where the POAs are more likely to be lying, and where a few POAs are lying. Our AD Planner uses this AD Map to evaluate the candidate assignments, and pick out the one that passes through where most assignments cluster.

More formally, the goal is to find the assignment for a single robot to move, which can be written as a path going through all targets. The assignment decides the target order visited by the path, and some waypoints (grid cells of the map) may be visited more than once. We evaluate the assignment path with the Clustering Value based on the AD Map.

To find the shortest assignment, we repeat the assignment search process introduced in Pseudo 14 by replacing the path length in S_{path} with the path clustering $P_{AD}(l^*)$ defined as:

$$\boldsymbol{P_{AD}(l^*) = \prod_{m_{x,y} \in l^*} P_{AD}(m_{x,y})} \tag{32}$$

The cluster evaluation of an assignment is

$$\boldsymbol{P_{AD}[\mathrm{L}(S_{Q_1,A}|\mathrm{M}_{AD})] = \prod_{l_i^* \in \mathrm{L}(S_{Q_1,A}|\mathrm{M}_{AD})} P_{AD}(l_i^*)} \tag{33}$$

The AD Planner seeks the assignment $\mathrm{L_{AD}(S_{Q_1,A}|M_{AD})}$ which the highest evaluation:

$$\boldsymbol{\mathrm{L}_{AD}(S_{Q_1,A}) = \underset{\mathrm{L}(S_{Q_1,A}|\mathrm{M}_{AD})}{\mathrm{argmax}}\; P_{AD}[\mathrm{L}(S_{Q_1,A}|\mathrm{M}_{AD})]} \tag{34}$$

In the AD Map, a grid cell $m_{x,y}$ has a Clustering Value $P_{AD}(m_{x,y}) \leq 1$. So $P_{AD}[\mathrm{L(S_{Q,A}|M_{AD})}]$ will be smaller and smaller when we extend the assignment length in the searching process. For numerical stability, we calculate and compare the negative log likelihood of $P_{AD}[\mathrm{L(S_{Q,A}|M_{AD})}]$ for candidate assignments in the searching process. Therefore, in our approach, the AD Planner will find the minimal evaluation to be:

$$\mathrm{L_{AD}}\left(\mathrm{S_{Q,A}}\middle|\mathrm{M_{AD}}\right) = \underset{\mathrm{L}\left(\mathrm{S_{Q,A}}\middle|\mathrm{M_{AD}}\right)}{\operatorname{argmin}} \left\{-\sum_{i=1}^{\mathrm{L}_{en}} \log\left[P_{AD}\left(m_{x,y}^{i}\right)\right]\right\} \tag{35}$$

The difference between the AD Planner and the PD Planner is that the grid cells may be revisited several times. So, one grid cell may be counted more than once when we evaluate a candidate assignment. In practice, we separate the assignment into several segments and each segment links between two targets (or links between start point and a target). Then, the candidate assignment $\mathrm{L}\left(\mathrm{S_{Q,A}}\middle|\mathrm{M_{AD}}\right)$ with N_{seg} segments can be written as the connection of segments:

$$\mathrm{L}\left(\mathrm{S_{Q,A}}\middle|\mathrm{M_{AD}}\right) = \{l_1, l_2, \dots, l_{N_{seg}}\} \tag{36}$$

In the searching process, we can evaluate each segment as a path, which is the same as evaluate a PD Planner path:

$$-\log[P_{AD}(l)] = -\sum_{i=1}^{|l|} \log\left[P_{AD}\left(m_{x,y}^{i}\middle|m_{x,y}^{i} \in l\right)\right] \tag{37}$$

In fact, when we search the segment path l, we guarantee that it has the lowest evaluation (37).

Then, we evaluate the Clustering Value of the candidate assignment as:

$$-\log\left[\mathrm{L}\left(\mathrm{S_{Q,A}}\middle|\mathrm{M_{AD}}\right)\right] = -\sum_{i=1}^{N_{seg}} \log[P_{AD}(l_i)]$$

Therefore, the AD Planner planned assignment in our implementation is found by:

$$\mathrm{L_{AD}}\left(\mathrm{S_{Q,A}}\middle|\mathrm{M_{AD}}\right) = \underset{\mathrm{L}\left(\mathrm{S_{Q,A}}\middle|\mathrm{M_{AD}}\right)}{\operatorname{argmin}} \left\{-\sum_{i=1}^{N_{seg}} \log\left\{P_{AD}\left[l_i \in \mathrm{L}\left(\mathrm{S_{Q,A}}\middle|\mathrm{M_{AD}}\right)\right]\right\}\right\} \tag{38}$$

The pseudo code of the AD Planner is shown in *Pseudo 15*, where we sample s instances and estimate M_{AD} (line 1~7). In line 4, $\mathrm{L_{min}}$ is initialized with a greedy order $\{\mathrm{A}_1 \rightarrow \mathrm{A}_2 \rightarrow \cdots \rightarrow \mathrm{A}_{\mathrm{N_A}}\}$ (select the shortest path from a previous target to next target). N_{all} paths with the lowest value $-\log[P_{AD}(l^*)]$ are found in the AD Map (means the highest $P_{AD}(l^*)$) and recorded in S_{path} (line 8). OptAssignment (Pseudo 14) uses it as the path cost to compare the different assignments.

Pseudo 15 The AD Planning Algorithm.

```
AssignmentPlanner(OC Map: $M_{oc}$, Robot: $Q_1$, Targets: $A$, Particle:s)
1  for (i=1; i<s; i++)
2      sample map instance $M_i$ based on $M_{oc}$
3      Find all paths set $S_{path}$ between any two points among $Q_1$ and targets (A*)
4      init $L_{cur}$ for robot to visit all targets in order of $\{A_1\text{->}A_2\text{->}A_3\text{->}\dots\}$
5      Find the optimal assignment $L_{min}(S_{Q1,A}|M_i)$   //call OptAssignment function
6      update AD Map $M_{AD}$ using $L_{min}(S_{Q1,A}|M_i)$
7   end for
8  Find all paths with the lowest value $-\log[P_{AD}(l)]$ in $M_{AD}$, using A*, between
any two points among $Q_1$ and targets, recorded in $S_{path}$
9  Call OptAssignment to find $L_{AD}(S_{Q1,A}|M_{AD})$, the path length is replaced with
$-\log[P_{AD}(l)]$ as the evaluation.
4  return the optimal assignment $L_{AD}(S_{Q1,A}|M_{AD})$
```

5.4.3 AD Planner Experiment Results

In our simulator system, we do not limit the upper boundary of N_A, N_Q and the map size. However, we cannot spend unlimited time to do the experiments. Therefore, we design the experiment scenario for AD Planner as: one robot to visit four targets, to test the AD Planner.

The experiment map is a two-floor OC Map, Fig. 44, with one robot and four targets. We compare our AD Planner with the assignment algorithm based on threshold A* and entropy. The main difference between these algorithms is how to evaluate the candidate assignments in the planning process: AD Planner evaluates the clustering value of possible shortest assignments, threshold A* evaluates the threshold path length and the entropy evaluates the entropy.

Our robot moves from the start cell to all four targets in the OC Map. The simulator sampled 1500 map instances based on the OC Map as the true map for robot to travel. Our robot configuration is the same as the experiment for PD Planner, except that the robot can move from one floor to another floor. When the current path is blocked, the robot will make a new assignment for all unvisited targets based on the updated OC Map and begins travelling along this new path of the assignment. This overall process is repeated until the robot either visits all targets or determines that no further targets to be visited. We also add the AD planner based on a region map (described in Chapter 4) in our experiment, to see how the region affected the planned result.

Similar to the PD Planner experiment, we record all the paths travelled by the robot using the different algorithms. The experiment results are shown in Fig. 47. We can see that the robot with AD Planner has a higher cumulative probability (the biggest probability advantage is 20%) to visit all targets when travelling the same distance, comparing with the entropy and threshold A* based assignment algorithm. The AD Planner with higher particle number has better performance.

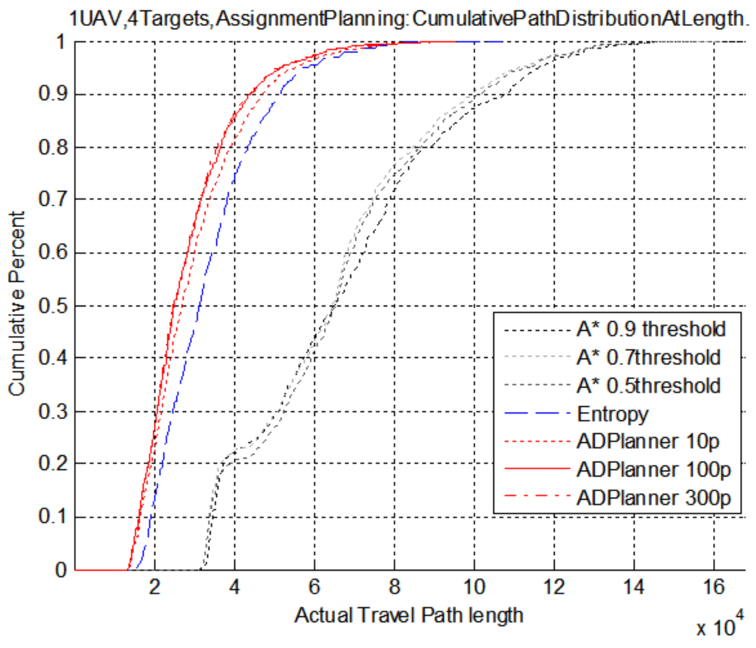


Fig. 47 The cumulative probability of different algorithms at each travel length

5.5 Group Allocation Distribution Planner

In this section, we consider the AD planner to the case where the group consists of more than a single robot. We introduce the Group Assignment Distribution (GAD)

Planner which will make a plan for several robots to visit many targets in an uncertain environment.

5.5.1 The Group Allocation Distribution (GAD) Map

The difference between AD Planner and GAD Planner is that we have one robot, or multiple robots, works in a group. Group allocation problem is an example of mTSP [115].

The group robots have $\mathrm{N_A}$ robots. The shortest group allocation $G_i = G_i(M^i)$ in MDP instance M^i is found by applying an MDP allocation planning algorithm. Here, the shortest allocation G_i in MDP instance M^i includes the shortest assignment for each robot: $G_i = \left\{\mathrm{L_{min}}\left(\mathrm{S_{Q_j,A}}|M^i\right)\right\}$, where $\mathrm{L_{min}}\left(\mathrm{S_{Q_j,A}}|M^i\right), j \in \{1,2,\dots,N_r\}$ is the shortest assignment for the j^{th} robot to visit the assigned targets $\mathrm{S_{Q_j,A}}$ in M^i.

Three steps are required to get the shortest allocation G_i:

1, Sample grid cells to generate a set of map instances M^i.

2, For each M_i, find all shortest paths between any two points among robots' positions and targets (exclude the point pairs between any two robots) using A*, and save them in the path cost matrix.

3, Exhaust all possible allocations by recursively calling the function "OptAllocation", which will return the shortest allocation with shortest path length for the robot group. The pseudo code of the recursive function "OptAllocation" is as shown in Pseudo 16. The assignments $\mathrm{L_{min}}\left(\mathrm{S_{Q_j,A}}|M^i\right)$ in a group allocation are found by calling "AssignmentPlanner" in line 2. In different allocations, the targets allocated to the same robot may be different. So, we have to plan the shortest assignments in each allocation.

Pseudo 16 The "OptAllocation" to exhaust all allocations and find the shortest allocation recursively [115].

OptAllocation(PathSet:S_{path}, Robots:Q, Targets:A, CurAllocate:G_{cur}, CurOpt:G_{min}, Index:i)

```
if (target index i > target number $N_A$)   //all targets are allocated?
   find the optimal assignments in $G_{cur}$ :call "AssignmentPlanner" for each assignment in $G_{cur}$
   if (current allocation $G_{cur}$ shorter than current optimal allocation $G_{cur}$)
      $G_{min} = G_{cur}$
   end if
   return;
end if
for(j = 1; j< $N_Q$; j++)   //allocate the i-th target to the j-th robot
    allocate the i-th target to the j-th robot assignment $S_{Qj,A}$, and update $G_{cur}$
    call OptAllocation($S_{path}$, Q, A, $G_{cur}$, $G_{min}$, i+1)
    remove the i-th target from $S_{Qj,A}$, and update $G_{cur}$
end for
```

In line 3, if the longest assignments of two allocations have the same length, we compare their second longest assignments, until we find the shorter one as the better allocation. The shortest allocation G_i satisfies:

$$\boldsymbol{G_i(Q,A) = \underset{G}{\operatorname{argmin}}\{L[G(Q,A,M^i)]\} = \underset{G}{\operatorname{argmin}}\left\{\max_{S_{Q_j,A}\in G(Q,A,M_i)}\left[L_{min}\left(S_{Q_j,A}|M^i\right)\right]\right\}} \quad (39)$$

In each allocation $G(Q,A,M^i)$, we can sort the robots' paths base on their length. If the longest paths of two assignments have the same length, we will compare their second longest assignment, until we find the shorter one as the better group allocation.

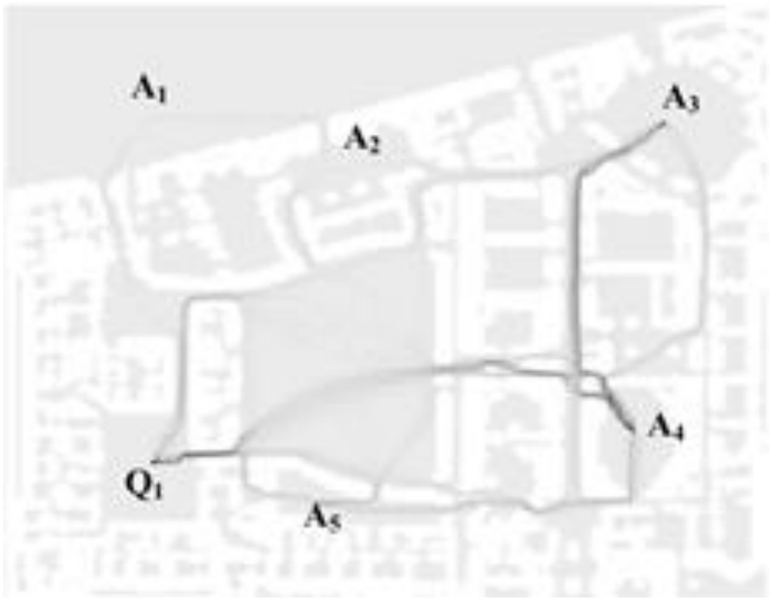
In section 5.3, we saw that the possible shortest assignments of one robot are clustering to some areas. When we have several robots, we also find that those possible shortest group allocations G_i also tend to cluster to some areas. Therefore, we define the Group Allocation Distribution (GAD) Map to describe where the possible shortest group assignments are clustering.

The GAD Map M_{GAD} is the set of AD maps for each robot, $M_{GAD} = \{M_{AD}^1, M_{AD}^2, \dots, M_{AD}^{N_Q}\}$. Each grid cell in M_{AD}^j records the probability that the j^{th} robot in the shortest allocation will actually pass through that cell. Please note that the targets allocated to each robot can be changed in each sampled map instance. Therefore, it is different with the AD Map in section 5.4.1, which contains the paths to visit all targets (the single robot group has to visit all targets by one robot). In the GAD Map, each robot may be assigned different targets which make the GAD Map of the robot containing the paths to visit different targets in different MDP instances.

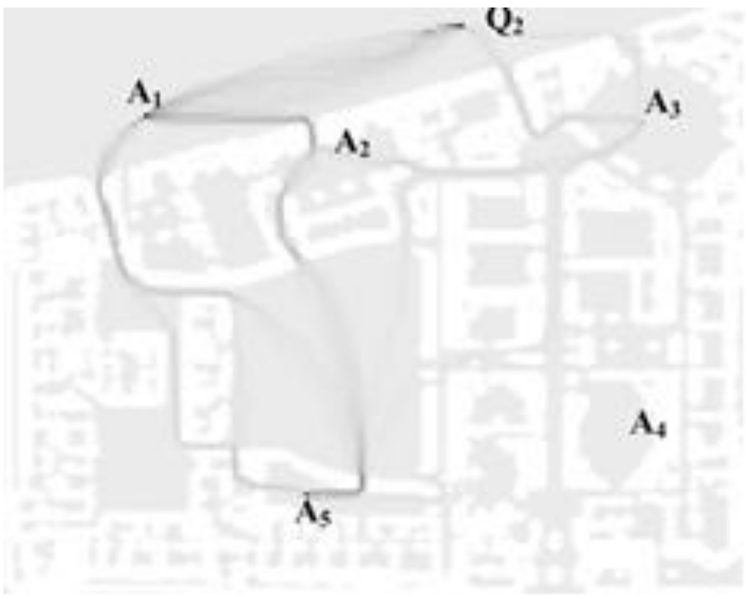
Marginalising over map instances, the probability of a group shortest allocation passing through the cell $m_{x,y}$ for the j^{th} robot is given by

$$\mathrm{P}_{\mathrm{GAD}}^{\mathrm{j}}(m_{x,y}) = \sum_{M^i \in M} \left\{ p\left[m_{x,y} \in \mathrm{S}_{\mathrm{Q_j,A}} | \mathrm{S}_{\mathrm{Q_j,A}} \in G_i(Q, A, M^i) \right] P(M^i) \right\} \tag{40}$$

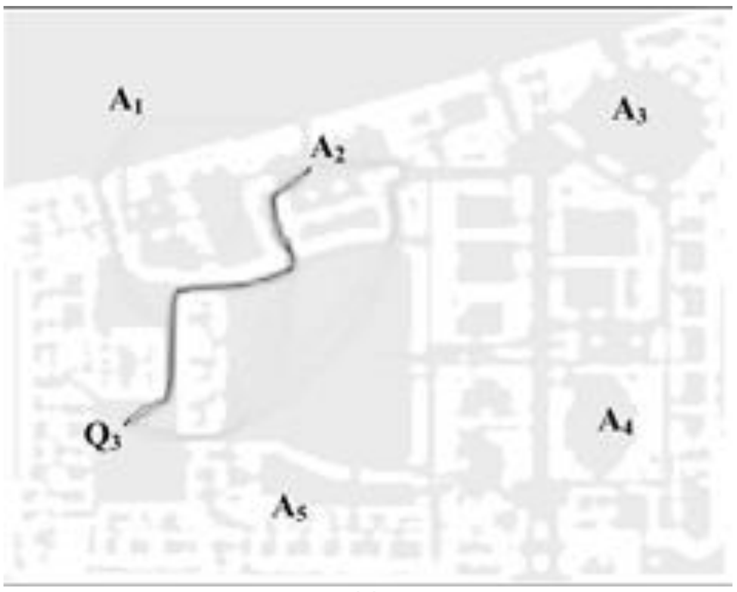
Generally, it is impossible to exhaustively compute all possible map instances to calculate the GAD Map. So, we use a Monte Carlo sampling strategy (the same as we do with PD Planner) to estimate it. A large number of map instances are drawn randomly, a group shortest assignment planner in certain environments is executed, and the GAD Map is computed empirically from (40) for each robot. Therefore, each robot in the robot group has a GAD Map, looks like Fig. 45.



(a)



(b)



(c)

Fig. 48 The GAD map example for 3 robot group. (a)(b)(c) indicate the AD Map for Robot-1, Robot-2 and Robot-3, respectively.

Fig. 48 shows an example of the GAD Map with three robots and five targets. Each robot has a different starting position. M_{GAD} shows that different robots have different clustering areas, depending on the shortest allocation in the map instances. For example, targets $\{A_3, A_4, A_5\}$ are not linked by the clustering areas in M_{AD}^3, because they are not always allocated to Robot 3.

The pseudo code to estimate the GAD Map is shown in Pseudo-code 17. Just like the AD Planner, we find N_{GAD} shortest paths once (line 4) and use them to find the shortest allocation. Each robot has a path linking a target and we need the paths to link among targets.

Pseudo-code 17   Estimate the GAD Map function

EstimateGAD(OC Map:M_{oc}, Particle:s, Robots:Q, RobotNum:N_Q, Targets:A)

```
1  Init GAD Map: $M_{GAD}=\{M^1_{GAD}, M^2_{GAD}, \ldots, M^{N_Q}_{GAD}\}$
2  for(i=0; i<$s$; i++)
3      Sample a map instance $M_i$ based on $M_{oc}$
4      Find all shortest paths $S_{path}$ between any two points among robots and targets
(exclude the point pairs of two robots)
5      init $G_{min}$ for robots having similar lengths (PathEqual algorithm)
6      optimal allocation $G_i$ = OptAllocation($S_{path}$, $Q$, A, $G_{cur}$, $G_{min}$, $1$)
5      update GAD Map $M^i_{GAD}$ based on $G_i$
6  end for
7  $M_{GAD}$=-log($M_{GAD}$)      //Calculate the negative log value
8  return $M_{GAD}$
```

The "FindOptimalGroupAssignment" is a function to find the shortest Group Allocation in a certain environment M^i. The pseudo code of this function is shown in Pseudo-code 18.

Pseudo-code 18   The function to find the shortest group assignment

FindOptimalGroupAssignment(Map:M^i, UAVs:Q, UAV Num:N_r, Targets:A)

```
1   Init optimal group assignment $G_{opt}$
2   try all possible group assignment $G_{cur}$, where $G_{cur}=\{S_{Q\_1,A}, S_{Q\_2,A}, \ldots, S_{Q\_Nr,A}\}$
3       for(j=0; j<$N_r$; j++)   //for each UAV, find its opt assignment in certain map $M^i$
4           Len($S_{Q\_j,A}$)=find shortest path for jth UAV in $\{M^i, S_{Q\_j,A}\}$
5           update Len($G_{cur}$) based on Len($S_{Q\_j,A}$)
6           if Len($G_{cur}$) < Len($G_{opt}$)
7               goto 2            //$G_{cur}$ has a path longer than $G_{opt}$, so try next possible $G_{cur}$
8           end if
9       end for
10      If current group assignment Len($G_{cur}$) < Len($G_{opt}$)
11          $G_{opt} = G_{cur}$
12      end if
13  end try
14  retrun $G_{opt}$
```

5.5.2 Group Allocation Planning based on the GAD Map

The GAD Map describes where the possible shortest group allocations are most likely to go in the OC Map. Our Group Allocation Distribution (GAD) Planner uses this map to find the Group Allocation G that pass through the cells where most group assignments cluster. So, we cluster allocation to each cell for planning an allocation, and do not count the assignments in one allocation that pass through a cell in GAD Map.

The Group Allocation G based on the OC Map includes the target assignments for each robot, written as: $G = \{\mathrm{S}_{\mathrm{Q}_1,\mathrm{A}}, \mathrm{S}_{\mathrm{Q}_2,\mathrm{A}}, \dots, \mathrm{S}_{\mathrm{Q}_{\mathrm{N}_\mathrm{r}},\mathrm{A}}\}$. The assignment for the j^{th} robot is $\mathrm{S}_{\mathrm{Q}_\mathrm{j},\mathrm{A}}, j \in \{1,2,\dots,N_r\}$ and the path for j^{th} robot is $\mathrm{L}(\mathrm{S}_{\mathrm{Q}_\mathrm{j},\mathrm{A}})$, which is defined as the AD Planned assignment in section 5.4.

Specifically, a candidate allocation is evaluated using

$$\begin{aligned}\boldsymbol{L_{GAD}}[\boldsymbol{G}(\boldsymbol{Q},\boldsymbol{A})] &= \underset{\boldsymbol{G}(\boldsymbol{Q},\boldsymbol{A},\boldsymbol{M_{GAD}})}{\mathbf{argmax}} \; \boldsymbol{C_v}[\boldsymbol{G}(\boldsymbol{Q},\boldsymbol{A},\boldsymbol{M_{GAD}})] \\ &= \underset{\boldsymbol{G}(\boldsymbol{Q},\boldsymbol{A},\boldsymbol{M_{GAD}})}{\mathbf{argmax}} \left\{ \underset{\mathrm{L}\left(\boldsymbol{S_{Q_j,A}}\right)\in\boldsymbol{G}(\boldsymbol{Q},\boldsymbol{A},\boldsymbol{M_{GAD}})}{\mathbf{argmax}} \left[\mathbf{L}_{\boldsymbol{AD}}\left(\boldsymbol{S_{Q_j,A}}\middle|\boldsymbol{M^j_{GAD}}\right)\right] \right\} \end{aligned} \tag{41}$$

where $\mathrm{C_v}(.)$ means the cluster value of possible shortest allocations, $\mathrm{L_{AD}}\left(\mathrm{S}_{\mathrm{Q}_\mathrm{j},\mathrm{A}}|\mathrm{M}^\mathrm{j}_\mathrm{GAD}\right)$ is the shortest assignment found using the AD Planner on map $\mathrm{M}^\mathrm{j}_\mathrm{GAD}$:

$$\mathbf{L}_{\boldsymbol{AD}}\left(\boldsymbol{S_{Q_j,A}}|\boldsymbol{M^j_{GAD}}\right) = \underset{\mathrm{L}\left(\boldsymbol{S_{Q_j,A}}|\boldsymbol{M^j_{GAD}}\right)}{\mathbf{argmax}} \; \boldsymbol{C_v}\left[\mathbf{L}\left(\boldsymbol{S_{Q_j,A}}|\boldsymbol{M^j_{GAD}}\right)\right] \tag{42}$$

The robot group needs to work together, so the goal is to find the Group Allocation which maximises the candidate Group Allocation Clustering Value $\mathrm{C_v}[\mathrm{G}(\mathrm{Q},\mathrm{A})]$.

In practice, we can evaluate $\mathrm{L}\left(\mathrm{S}_{\mathrm{Q}_\mathrm{j},\mathrm{A}}\right)$ in M^j_{GAD} using the negative log likelihood of its grid cells' cluster values for the numerical stability:

$$
\begin{aligned}
&\mathrm{L_{AD}}\left(\mathrm{S_{Q_j,A}}|M_{GAD}^{j}\right)\\
&= \underset{\mathrm{L}\left(\mathrm{S_{Q_j,A}}|\mathrm{M_{GAD}^{j}}\right)}{\operatorname{argmin}} -\log\left\{C_v\left[L\left(S_{Q_j,A}|\mathrm{M_{GAD}^{j}}\right)\right]\right\}\\
&= -\sum_{i=1}^{|\mathrm{S_{Q_j,A}}|} \log\left\{P_{GAD}^{j}\left[m_{x,y} \in L\left(S_{Q_j,A}\right)|M_{GAD}^{j}\right]\right\}
\end{aligned} \tag{43}
$$

The GAD Planner searches and compares candidate group allocations to maximize $\mathrm{C_v}[\mathrm{G}(\mathrm{Q},\mathrm{A})]$. The pseudo code of GAD Planner is shown in Pseudo-code 19, where the line 4 uses the same process in Pseudo 16 to find all possible allocations.

Pseudo-code 19 the GAD Planner function

```
GADPlanner(OC Map:$M_{oc}$, Particle:s, Robots:$Q$, RobotNum:$N_Q$, Targets:$A$)
1   $M_{GAD}$=EstimateGAD($M_{oc}$, $s$, $Q$, $N_Q$, $A$)
2   Init $G_{opt}$ allocate targets to robots with similar GAD evaluations
3   try all possible group allocation $G_{cur}$, where $G_{cur}=\{S_{Q\_1,A}, S_{Q\_2,A}, \ldots, S_{QN_Q,A}\}$
4       for(j=0; j<$N_Q$; j++)  //for each robot,find its optimal assignment in $M_{GAD}$
5           $C_v(S_{Q\_j,A})$=find optimal assignment for jth robot based on $\{M^j_{GAD}, S_{Q\_j,A}\}$
6           update $C_v(G_{cur})$ based on $C_v(S_{Q\_j,A})$
7           if $C_v(G_{cur}) < C_v(G_{opt})$
8               goto 3              //$G_{cur}$ is not better than $G_{opt}$, so try next possible $G_{cur}$
9           end if
10      end for
11      If current group allocation $C_v(G_{cur}) < C_v(G_{opt})$
12            $G_{opt} = G_{cur}$
13      end if
14  end try
15  retrun $G_{opt}$
```

5.5.3 Calculation cost of AD and GAD Planner

In this section, we will use A* as the MDP planner to evaluate the calculation cost of the AD Planner and the GAD Planner.

In the AD Planner, we will find the paths from the robot to each target, and the paths between any two targets. There are $\mathrm{N_A}$ targets for a single robot to visit. The planned path number in each MDP instance is:

$$N_{AD} = \mathrm{N_A} + (\mathrm{N_A} - 1) + (\mathrm{N_A} - 2) + \cdots + 1 = \frac{\mathrm{N_A} + 1}{2}\mathrm{N_A} \tag{44}$$

Each sampled MDP instance has N nodes (grid cells). To estimate the CV Map for AD Planner, we use the particle s. The AD Planner will find $\mathrm{N_A}$ paths from N_{AD} paths to construct the shortest assignment path in each MDP instance. Then, the calculation cost of estimating CV Map for AD Planner is:

$$s\left[O(N) + O(N^2)N_{AD} + O\left(P_{N_{AD}}^{N_A}\right)\right] \tag{45}$$

where

$$P_{N_{AD}}^{N_A} = N_{AD}(N_{AD} - 1)(N_{AD} - 2)\ldots(N_{AD} - N_A - 1) = \prod_{i=0}^{N_A - 1}(N_{AD} - i) \tag{46}$$

The AD Planner plans a path in CV Map is the same as planning a path in a sampled MDP instance, but without the sampling operation. Therefore, the calculation cost of AD Planner is:

$$s\left[O(N) + O(N^2)N_{AD} + O\left(P_{N_{AD}}^{N_A}\right)\right] + O(N^2)N_{AD} + O\left(P_{N_{AD}}^{N_A}\right) \tag{47}$$

The GAD Planner has two main parts: estimating the GAD Map and finding the group allocation for multiple robots. Therefore, it is more complex than AD Planner.

In GAD Planner, we will find the paths from each robot to each target, and the paths between any two targets. We have $\mathrm{N_Q}$ robots and $\mathrm{N_A}$ targets. The planned path number in each MDP instance is:

$$N_{GAD} = \mathrm{N_A}\mathrm{N_Q} + (\mathrm{N_A} - 1) + (\mathrm{N_A} - 2) + \cdots + 1 = \mathrm{N_A}\mathrm{N_Q} + \frac{\mathrm{N_A}}{2}(\mathrm{N_A} - 1) \tag{48}$$

To estimate the GAD Map for GAD Planner, we use the particle s. The GAD Planner will find $\mathrm{N_A}$ paths from N_{GAD} paths to construct the shortest Group Allocation (the shortest assignment path for each robot). The upper bound of branch can provide a discount $\gamma \leq 1$ to the calculation cost, which depends on the scenario of the uncertain environment. Therefore, the calculation cost of estimating M_{GAD} is:

Then, the calculation cost of estimating CV Map for GAD Planner is:

$$O(M_{GAD}) = O(sN) + O(sN^2)N_{GAD} + O\left(s\gamma C_{N_{GAD}}^{N_A}\right) \tag{49}$$

where $O(N)$ is cost to sample M_i, $O(N^2)$ is the cost for A* to find the shortest path in M^i, $O\left(C_{N_{GAD}}^{N_A}\right)$ is the cost to find the shortest allocation in M_i. $C_{N_{GAD}}^{N_A}$ can be written as:

$$\boldsymbol{O\left(C_{N_{GAD}}^{N_A}\right) = O\left(N_{GAD}(N_{GAD} - 1) \dots (N_{GAD} - N_A - 1)\right) = O\left((N_{GAD})^{N_A}\right)} \tag{50}$$

The planning process to find the GAD allocation in GAD Map has the similar cost as planning a group allocation. The only difference is that the GAD allocation in M_{GAD} has not the sampling operation. Therefore, the calculation cost of GAD Planner is:

$$O(M_{GAD}) + O(N^2)N_{GAD} + O((N_{GAD})^{N_A}) \tag{51}$$

The main cost of GAD is $O(M_{GAD})$, which is the clustering of possible shortest allocations.

In application, we do not need to try all possible allocations GAD Planner, when we plan the shortest assignment/allocation in an MDP instance. We use three ideas that are used to make the planning process faster in pseudo codes.

The first idea is that a greedy assignment (like the PE assignment) can provide an allocation $G_{opt}(Q, A)$ as a baseline, which can reject the possible allocations worse than it. For example, when we find that one robot's path of the current allocation $\mathrm{L}[G_{cur}(Q, A)]$ is longer than $\mathrm{L}\left[G_{opt}(Q, A)\right]$, the current allocation $G_{cur}(Q, A)$ can be rejected before all targets are allocated to robots in the planning process.

The second idea is that we extend the current Group Allocation by linking a new target to one of the robots, in order to make sure that all targets are allocated and visited by one robot at least. Those assignments that some targets are connected with path segments but not visited by any robot will be rejected in the searching process.

Thirdly, we record the last target allocated to the robot whose assignment is the longest one in the allocation. In recursive process, we return to that robot directly and change its assignment. This process can reject the allocations with the same longest assignment at once.

In this way, the $O\left(C_{N_{GAD}}^{N_A}\right)$ in the calculation cost equations (47) and (51) will become smaller in our implementation.

5.5.4 Experiment Results

We simulate the robot moving with constant velocity. For the trials here, we consider the case with $\{\mathrm{N_Q} = 3, \mathrm{N_A} = 5\}$. These numbers are sufficiently large to illustrate the behaviours of the different algorithms which were implemented on a low end machine (IntelCore2Duo CPU 2.93GHz with 4GB of memory).

We use a three-floor OC Map, see Fig. 49, to do the experiment. In this map, we add a free space around the building (the blue boundary grid cells), where the robot can move freely. In addition, we add a punishment for robot to fly from one floor to another floor. We transfer this punishment to the path length: flying up will spend more energy (fly 1.5 times longer in length) and flying down will spend less energy (0.8 time shorter in length). There are two variables to record these punishments for robot moving from one floor to another floor, and they are used for all assignment planner algorithms in our experiment.

The robots start from their initial cells, and follows the planned allocation to visit their allocated targets. The robot can observe obstacles within a fixed distance (line-of-sight) and update its own OC Map. However, the robot cannot detect the status of a grid cell behind an obstacle.

A simple communication strategy is used: robots can exchange the updated OC Maps to make the global replan when a single robot is blocked by an obstacle. This overall process is repeated until all targets are visited.

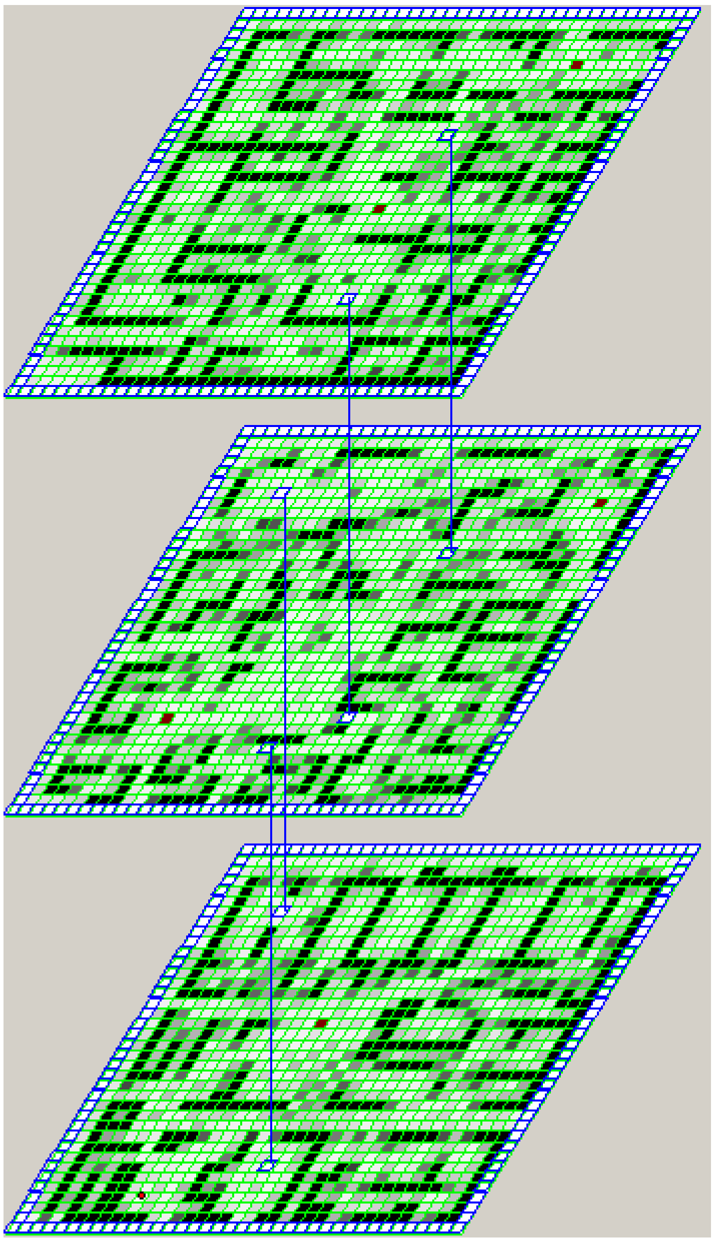


Fig. 49 The OC Map for Group Allocation experiment

We compare the GAD Planner with the Group Allocation Planner based on threshold A* and entropy in the experiment. The robot(s) start from the same cell to visit 5 targets in three-floor building. The robot group uses different Group Allocation Algorithms to make a plan for each robot. If one robot is blocked by an obstacle, the robot group will exchange their updated OC Maps and make a global re-plan, until all targets are visited or make sure that no targets are reachable.

Our experiment includes one robot, two robots and three robots to visit these five targets. The simulator sampled 1500 true maps as the true map instances based on M_{oc}. These instances are used to test the performance of different planners. The robots do not know these true maps, and they plan and re-plan with their updated M_{oc}. Only the simulator does know all obstacles and empty spaces in these instances. All results were computed over 1500 Monte Carlo trials.

We record all the paths travelled by each robot using different algorithms. We select the longest travelled route in the robot group of different algorithms. We get the travelled length of different algorithms for each instance. The average length and the quantile 0.5 (half of all paths' lengths are shorter than the length of the quantile 0.5) of these lengths are shown in Fig. 50, where the length of the GAD Planner (500p) is decreased by about 30% than others.

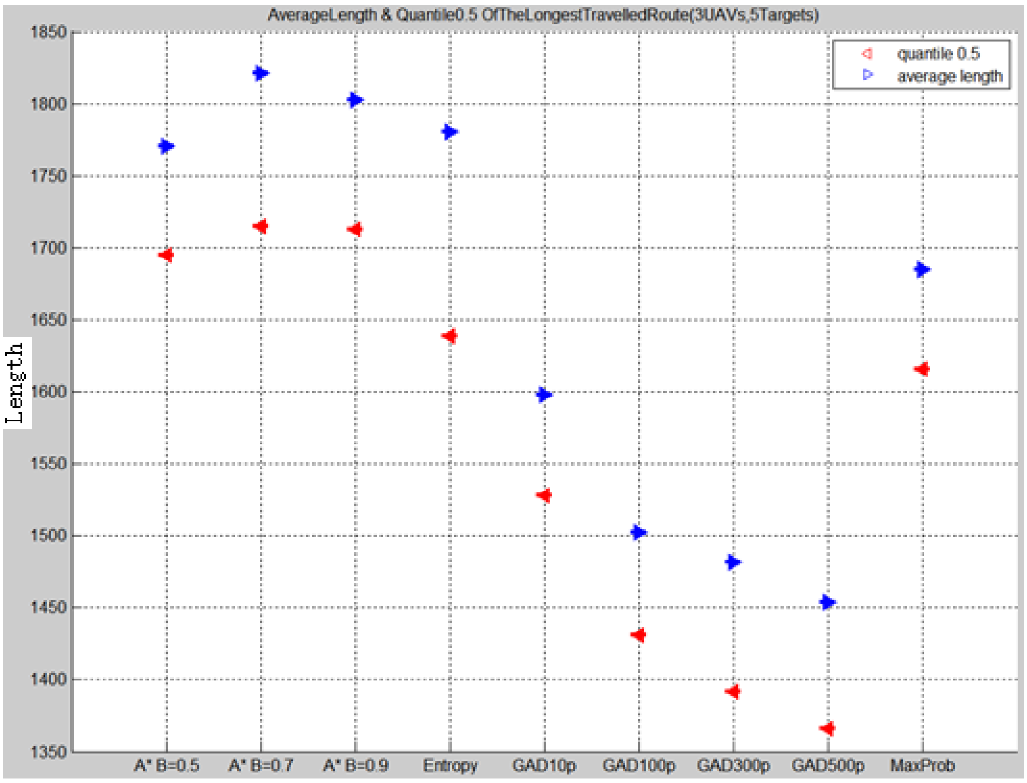


Fig. 50 The average length and quantile 0.5.

Fig. 51 shows the cumulative distribution of the longest path lengths for different algorithms. The results show that the GAD outperforms the other algorithms – for any given travel length, the probabilities that the robots will visit all targets is less than that of all other algorithms The reason is that the GAD Planner exploits global uncertainty information; the entropy planner only exploits local uncertainty information, and the A* planner uses the free space assumption that states in the unvisited environment are open. The figure also shows that the increasing number of particles improves the performance of the GAD. This reflects the fact that it computes a progressively more accurate approximation of the M_{GAD}. However, the marginal improvement declines with the number of samples, suggesting that the particle number 500 is close to the highest performance in the experiment OC Map.

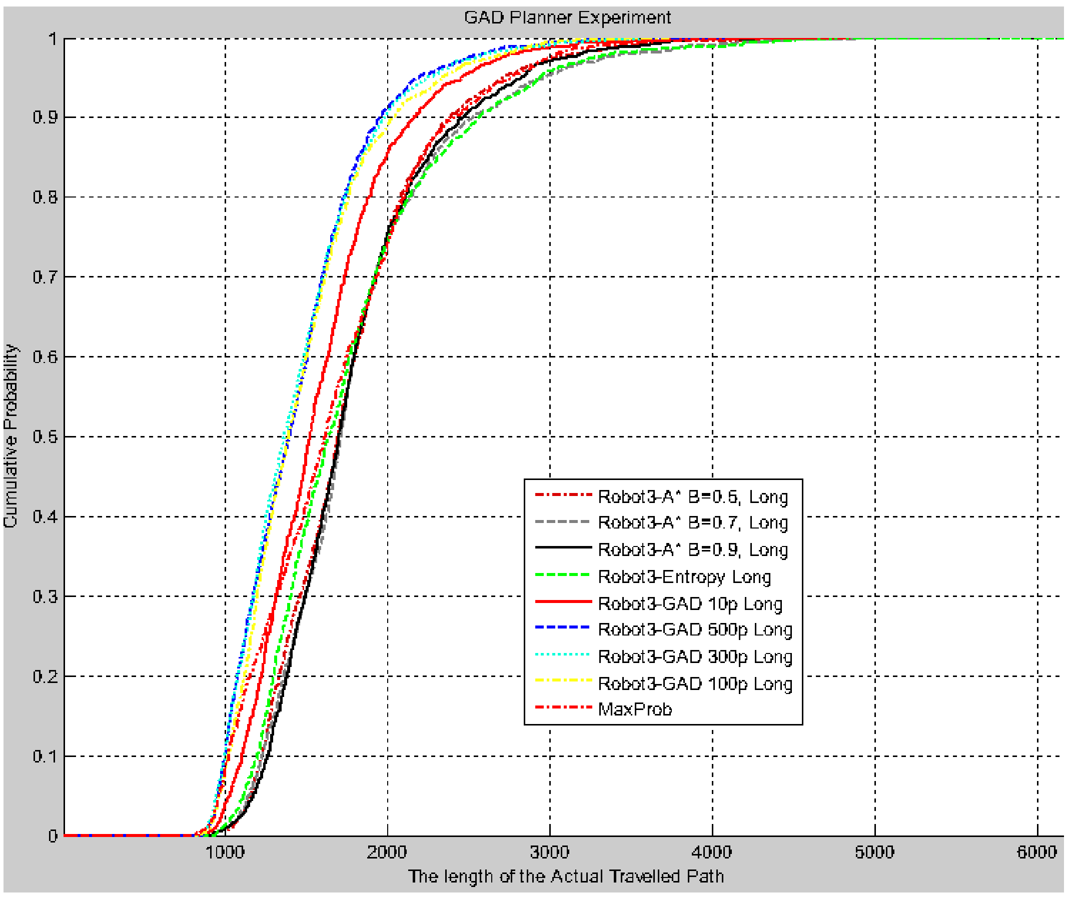


Fig. 51 Cumulative distribution of travel lengths for different allocation algorithms.

The histograms of the longest travel route in the robot group using different algorithms are shown in Fig. 52. We can see that the crest of GAD planner is the highest, and its peak is at x-coordinate 2000. Other algorithms peaks are located on its right side. Therefore, the cumulative probability of the robot travelled route based on GAD planner increases faster than other algorithms.

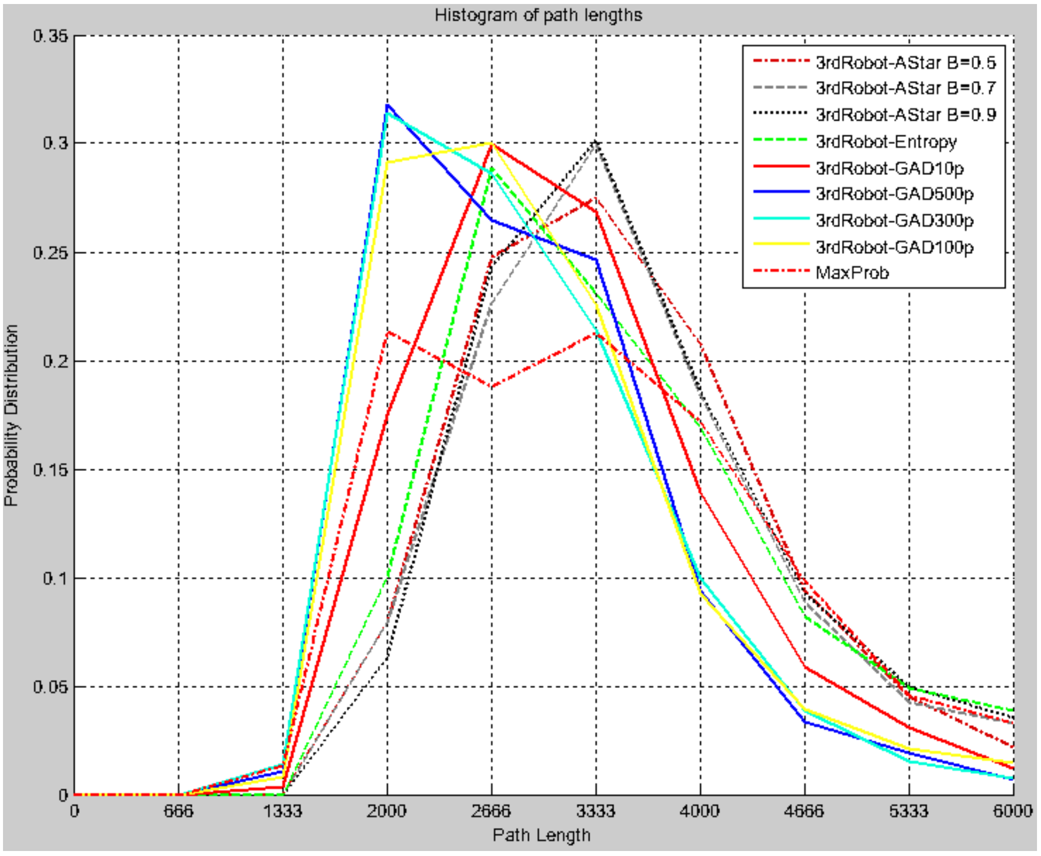


Fig. 52 The histogram of the longest travelled routes for the different algorithms

In the experiment, each algorithm has a probability to find the shortest travelled route length for the robot group, comparing with other algorithms. The probability is listed in Table 7.

The bigger re-plan number means the former planned allocations have higher probability to be blocked. Fig. 53 shows the average re-plan numbers, where the GAG Planner can reduce re-plan numbers significantly (only the MaxProb has less replan number than GAD Planner).

Table 7 The probability that each algorithm find the shortest travelled length for the robot group

Algorithms	Probability to find the shortest travel length
A* $\beta = 0.5$	0.0260
A* $\beta = 0.7$	0.0313
A* $\beta = 0.9$	0.0207
Entropy	0.0453
GAD 10p	0.0900
GAD 100p	0.1567
GAD 300p	0.2107
GAD 500p	0.3053
MaxProb	0.1140

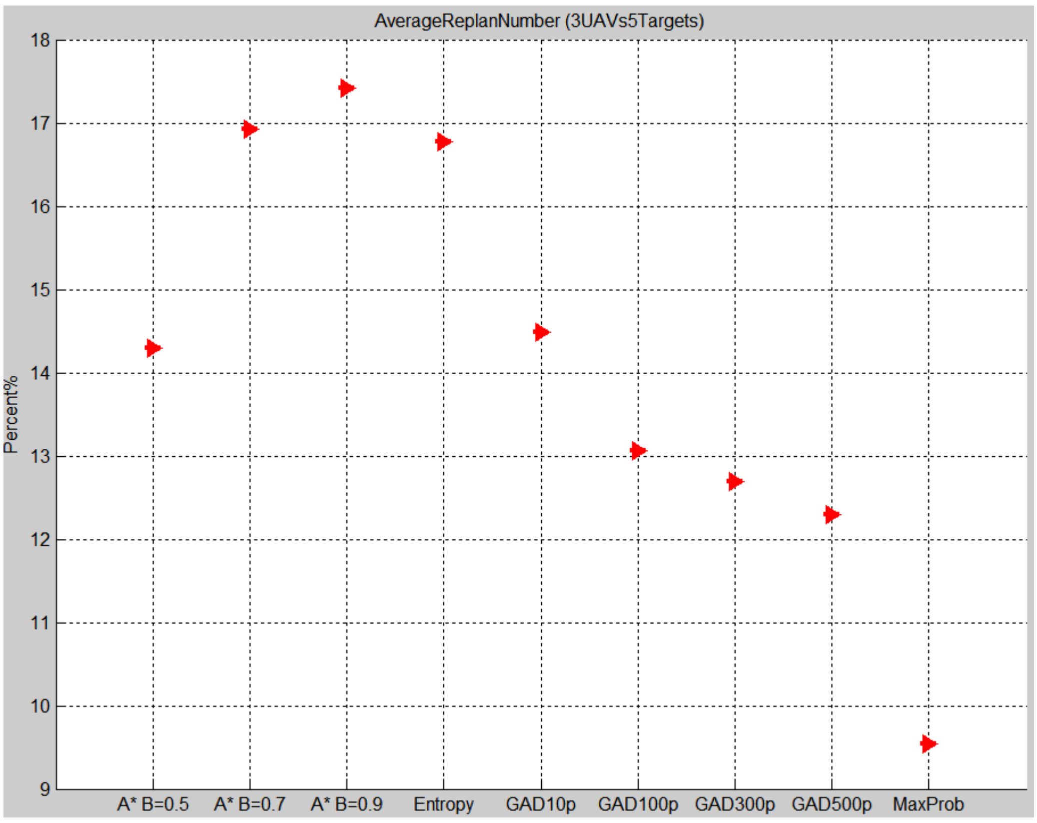


Fig. 53 The average number of replans for different algorithms

The average time t_{CMP} for the robot group to complete the mission can be calculated:

$$\boldsymbol{t_{CMP}} = \frac{\boldsymbol{l_{avg}}}{\boldsymbol{v}} + \boldsymbol{t_{replan}} \qquad (52)$$

where l_{avg} is the average length of the longest travelled route, v is the velocity for each robot to travel in an indoor uncertain environment, $\frac{\mathrm{l_{avg}}}{\mathrm{v}}$ is the robot travel time and t_{replan} is the initial plan and all re-plans' time used by the algorithm in CMP. In Fig. 54, we use 8-threads to sample map instances and estimate GAD Map for GAD500p and we can see that the GAD outperforms the other algorithms.

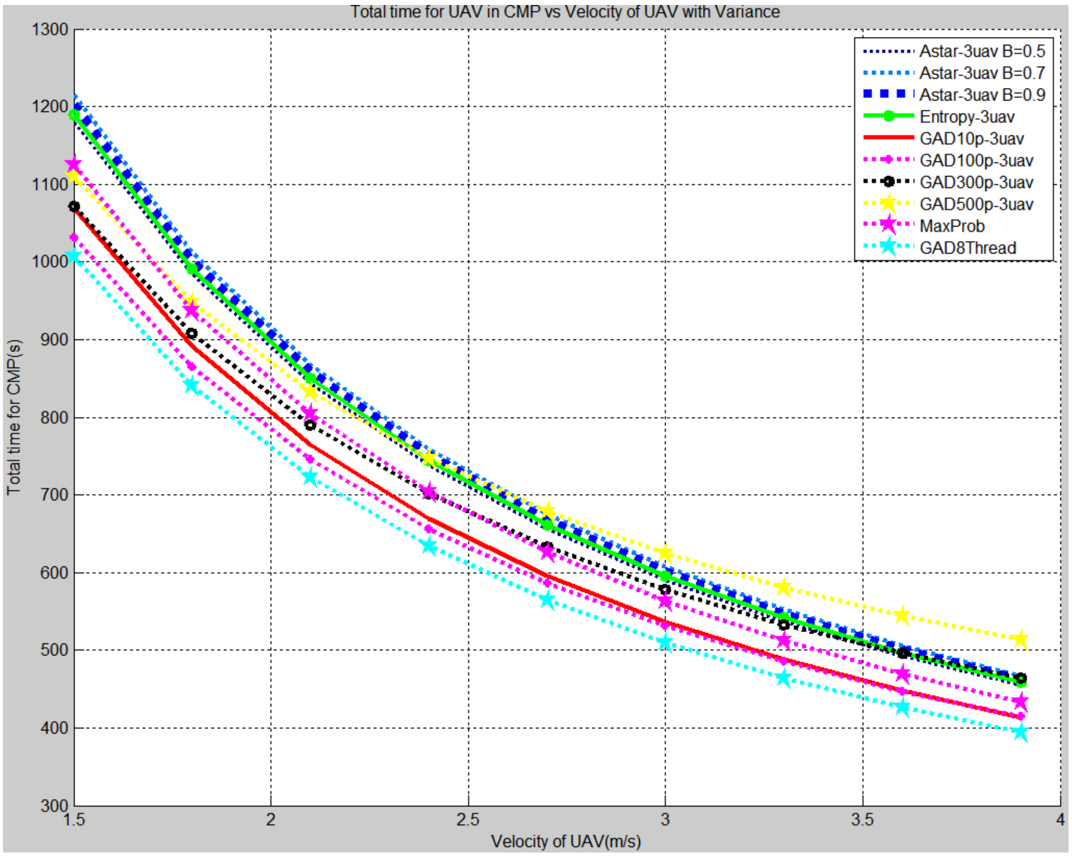


Fig. 54 Average CMT as a function of robot travel velocity.

We tasked one robot, two robots and three robots to visit 5 targets in the same uncertain environments. We compare the longest robot travelled distance in these three cases. The experiment result is shown in Fig. 55. If we increase the robot number, the longest route of the robot group will be decreased. For the single robot group, one robot has to visit all targets by itself. If we have two robots in the group, we can expect the group to visit all targets about twice faster than the single robot group. If we have three robots in the group, we can expect about three times faster than the single robot group. The average travel lengths with different number of robots are shown in Table 8.

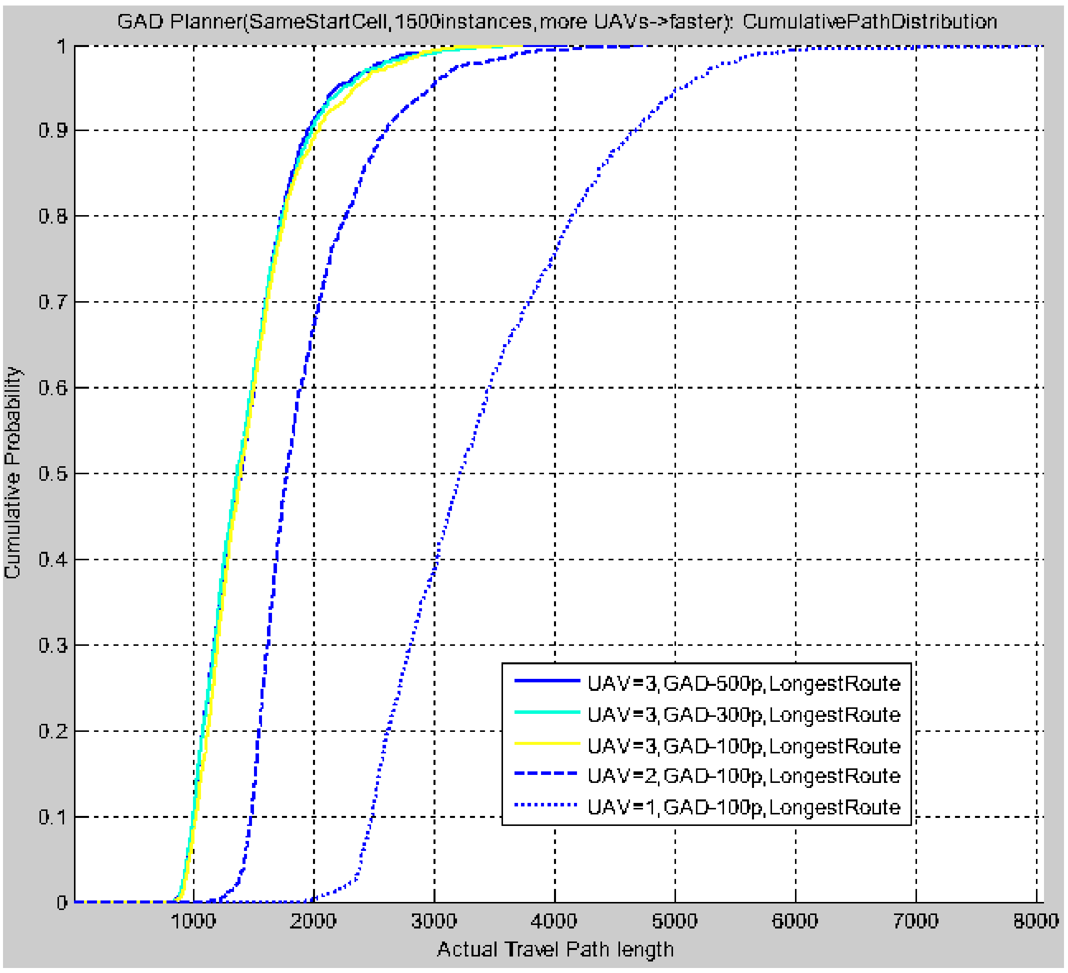


Fig. 55 The longest travelled distance for different number of robots.

In our implementation, we use the SiPP strategy to decrease the sample operations and two Group Allocation strategies (in section 5.5.3, Group Allocation base line and assignment extension) can remove some group allocations longer than the upper band in the searching process. Theoretically, we are not sure how much we can decrease the high computational cost for these operations.

Table 8 The average Length and the Standard Deviation of Fig. 55

UAV number and particle	Average Length	Standard Deviation
UAV=3,GAD500p	1448.1	419.9523
UAV=3,GAD300p	1450.6	432.3435
UAV=3,GAD100p	1476.6	435.1294
UAV=2,GAD100p	1932.6	482.0129
UAV=1,GAD100p	3415.2	883.8637

5.6 Summary

In this chapter, we consider the problem of planning an allocation for a robot group, with multi-robots and multi-targets in an uncertain environment. Our proposal is to maximize the probability that the planned allocation stays in the areas where most of the possible shortest allocations tend to cluster. The AD Planner is an extension of the PD Planner, which can make an assignment planning for one robot. The GAD Planner is an extension of AD Planner, which can make a group allocation planning for multiple robots.

The Assignment Distribution (AD) Planner algorithm plans an assignment based on the Assignment Distribution (AD) Map. The experiment shows that the autonomous robot with our assignment algorithm has a higher probability to visit all targets than other algorithms. The more accurately we increase the particle number to estimate the AD map, the better performance we can get.

The Group Allocation Distribution (GAD) Planner plans an allocation based on the Group Allocation Distribution (GAD) Map. The experiment shows that the autonomous robots group with GAD planner has a higher probability to visit all targets than other algorithms. We could also see that when we use more robots, the robot could travel shorter when all the targets are visited.

Our experiment shows that GAD Planner can reduce the travelled length and the time to complete the mission.

6 Summary and Future Research

6.1 Summary of Planning Based on Clustering

An autonomous robot is a set of advanced mechanical, electronic information, control tracing and problems solving planners. Our task is to make an allocation planning for the autonomous robot group to visit the target(s) in an uncertain environment. The targets are provided by the rescue team, because that the disastrous damage to the building is hard to be predicted. We use the Unmanned Aerial Vehicle (UAV) as the robot in our research.

Our research aims at dealing with the affection of uncertain positions of obstacles in uncertain environments of indoor scenarios. We design a novel Monte Carlo based planning approach, which is called the PD Planner, the AD Planner and the GAD Planner, based on POMDP model.

In the first chapter, we introduce problem of travelling in an uncertain environment inside a building to find survivor(s). The uncertain environment is caused by disasters, like an earthquake. In the second chapter, we review the different path planning algorithms in an uncertain environment based on POMDP model. We notice that the robot needs to make many re-plans in an uncertain environment before it reaches the target. Therefore, we are obliged to do the research based on the Complete Mission Process (CMP), where the robot starts from its initial point, and moves along the planned and re-planned paths until all targets are visited, or makes sure that the unvisited targets are not reachable, in an uncertain environment.

We design the planner in a 2-dimension uncertain environment, called Path Distribution (PD) Planner, for a single robot and single target scenario in Chapter 3. We simulate a UAV as the autonomous robot to search for survivors, which is a good assistant of the rescue team. We assume that the uncertain environment should be described correctly with an Occupancy Cell (OC) Map, where each grid cell records a probability that the grid cell is occupied by an obstacle. We implement a simulator to compare the performances of the PD Planner with other algorithms in CMP. When an autonomous robot is moving in the simulator, it can detect and update the OC Map within a short distance (can be set to different values). The experiment results show that the PD Planner has a higher cumulate probability to reach the target when robot travels the same distance. The PD Planner helps our robot to move efficiently in an uncertain environment.

In Chapter 4, we analysed the planning process of PD Planner and propose two strategies to reduce the computational cost of PD Planner: the Sampling in Planning Process (SiPP) and Hierarchal Path Distribution (HPD) Planner. The experiment shows that the computation time is decreased much and the performance of the planner is decreased a little.

In Chapter 5, we extend the PD Planner for a group robot and many targets in a 3-dimension uncertain environment. The Assignment Distribution (AD) Planner is the algorithm to plan an assignment for a single robot and many targets. The Group Allocation Distribution (GAD) Planner is the algorithm to plan paths for multiple robots and multiple targets. AD Planner compares different order of visiting many targets based on the evaluations of paths linking any two points, which is calculated by PD Planner. GAD compares different group allocations, where all targets are allocated to each robot and the path of each robot is evaluated using AD Planner. In the robot group, one robot has the longest travelled route. And we want this longest route to be the shortest among different group allocations. The experiment results show that the robot group using GAD Planner has a higher probability to visit all targets after travelled the fixed length. The time to complete the mission of GAD Planner are also smaller than other algorithms.

6.2 Potential Future Works

One limitation of our planners is the calculation of the Clustering Value (CV) map, which requires lots of calculations. There may be several researches for the future work to reduce the calculation.

One possible future research is to define some region templates for constructing regions faster in HPD Planner. The preconditioned data structure can be used to find the regions in a large area. We discuss a simple preconditioned data structure in Chapter 4, where the grid cells in a rectangle area have the same occupancy probability (similar as the percolation theory [101]). Other types region templates based on the preconditioned data structures in OC Map may be researched in the future.

Another possible research is to calculate the PD Map directly by analysing the structure of the general OC Map. In practice, we use the Monte Carlo Random Sampling strategy to estimate the PD Map. We could calculate/estimate PD map directly, by researching the skeleton analysis of OC Map combining with the sample rules.

Thirdly, we could use the non-deterministic algorithms, which can find the sub-shortest solution, to estimate CV Map. We mainly introduce PD Planner using the deterministic MDP algorithm to calculate/estimate the CV Map and find the path. For the non-deterministic algorithms, which can find a sub-shortest solution, the PD Map will be calculated/estimated as the sub-shortest paths Clustering Value. However, it is possible that each single sub-shortest solution is lying next to the shortest paths and offers some useful information about the location of the shortest path. If we can use the sub-shortest paths to get the PD Map accurately in the future research, we can speed up the PD/AD/GAD Planners. Because the sub-shortest paths can be found faster in most cases.

Another limitation in our approach experiment is different re-plan conditions. Currently, the re-plan occurs when an obstacle blocked the current planned path. We could change the re-planning condition: a new gap is found in the updating OC Map, like a new broken wall. For example, we define a width around the high clustering value cells to construct a “corridor” and re-plan when a whole broken boundary of the “corridor”. Due to the limited research time, we did not do this type of experiment in our research. But, it may be an interesting research in the future.

Lastly, we can ask some robots to explore and remove the uncertainty for the scenarios where the number of robots is bigger than the number of targets.

7 References

[1] K. Shiono, F. Krimgold and Y. Ohta, "Modeling of search-and-rescue activity in an earthquake", Earthquake Engineering, Tenth World Conference, Balkema, Rotterdam, 1992.

[2] B. Chazelle, "Approximation and decomposition of shapes". In J. T. Schwartz and C. –K. Yap, editors, Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics, pages 145-185. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[3] S. Bouabdallah, M. Becker, V. de Perrot and R. Siegwart, "Toward Obstacle Avoidance on Quadrotors", the XII International Symposium on Dynamic Problems of Mechanics(DINAME), P. S. Varoto and M. A. Rindade (Editors), ABCM, Ilhabela, SP, Brazil, February 26 –March 2, 2007.

[4] "The Bhuj, India Earthquake of 26th January 2001", A Field Report by EEFIT, 2001.

[5] "The Ji-Ji, Taiwan Earthquake of 21 September 1999", A Field Report by EEFIT, 1999.

[6] Prof. C. Pathak, "National Map Policy: Opportunity and Challenges", Indian Catographer, 2006.

[7] Geographical Survey Institute, Japan, "New NSDI and National Mapping Policy of Japan", ECONOMIC AND SOCIAL COUNCIL, Eighteenth United Nations Regional Cartographic Conference for Asia and the Pacific Bangkok, 26-29 October 2009 Item 7(b) of the provisional agenda Invited Papers, 2009.

[8] "The Erzincan, Turkey Earthquake of 13 March 1992", A Field Report by EEFIT, 1992.

[9] R. Bellman, "A Markovian Decision Process". Journal of Mathematics and Mechanics 6, 1957.

[10] R. E. Bellman. "Dynamic Programming". Princeton University Press, Princeton, NJ, 1957. Dover paperback edition (2003), ISBN 0486428095, 2003.

[11] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction". The MIT Press, Cambridge, MA, 1998.

[12] A. Brooks, A. Makarenko, S. Williams, and H. Durrant-Whyte, "Parametric POMDPs for planning in continuous state spaces", Robotics and Autonomous Systems, 54(11): 887-897, 2006.

[13] S. Thrun, W. Burgard and D. Fox, "Probabilistic Robotics", MIT Press, Cambridge, MA, 2005.

[14] J. Blythe. "An overview of planning under uncertainty", AI Magazine, 20(2): 37-54, 1999.

[15] C. Boutilier, T. Dean and S. Hanks, "Decision-theoretic planning: Structural assumptions and computational leverage", Journal of Artificial Intelligence Research, 11:1-94, 1999.

[16] R. Sutton and A. Barto, "Reinforcement Learning: An Introduction", MIT Press, Cambridge MA, 1998.

[17] N. Roy, G. Gordon and S. Thrun, “Finding approximate POMDP solutions through belief compression”, Journal of Artificial Intelligence Research, 23:1-40, 2005.

[18] B. Burns, O. Brock, “Single-Query Entropy-Guided Path Planning”, In IEEE Int. Conf. Rob. Aut., pages 3120-3125, Barcelona, Spain, 2005.

[19] Y. Tian, L. Yan, G. Park, S. Yang, Y. Kim, S. Lee, and C. Lee , “Application of RRT-based local Path Planning Algorithm in Unknown Environment”, Proceedings of the 2007 IEEE International Symposium on Computational Intelligence in Robotics and Automation Jacksonville, FL, USA, June 20-23, 2007.

[20] G. Kewlani, G. Ishigami, K. Iagnemma, “Stochastic Mobility-based Path Planning in Uncertain Environments”, The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems October 11-15, 2009 St. Louis, USA, 2009.

[21] P. H. Grossi and D. Windeler, “An introduction to catastrophe models and insurance”, Catastrophe modelling: a new approach to managing risk, Springer: New York, page245, 2005.

[22] National Institute of Building Safety, “HAZUS: Hazards U.S.: Earthquake loss estimation methodology, NIBS Document Number 5200”, National Institute of Building Sciences, Washington, DC, 1997.

[23] B. Goodno, A. Bostrom, J. Craig, and J. Park, “Probabilistic Decision Support for Regional Risk Assessment”, Final Report, Georgia Institute of Technology, Atlanta,GA, 2006, http://smartech.gatech.edu/dspace/handle/1853/23037, page 235.

[24] A. Elnashai, L. Cleveland, T. Jefferson, and J. Harrald, “Impact of Earthquakes on the Central USA, Mid-America Earthquake Center Report 08-02”, Mid-America Earthquake Center, Urbana, Illinois, 2008. https://www.ideals.uiuc.edu/handle/2142/8971, page 936.

[25] D. Boore, “Estimates of average spectral amplitudes at FOAKE sites”, Appendix C in An Evaluation of Methodology for Seismic Qualification of Equipment, Cable Trays, and Ducts in ALWR Plants by use of Experience Data, 1997.

[26] M. Wang and T. Takada, “Macrospatial correlation model of seismic ground motions”, Earthquake Spectra, 21: 1137-1156, 2005.

[27] K. Goda and H. Hong, “Spatial correlation of peak ground motions and response spectra, Bulletin of the Seismological Society of America, v. 98, p.354-365, 2008.

[28] H. Krawinkler and E. Miranda, “Performance-Based Earthquake Engineering”. Chapter 9 of Earthquake Engineering: From engineering seismology to performance based engineering, edited by Y. Bozorgnia, and V. Bertero, CRC press, 1996.

[29] H. Durrant-Whyte and T. Bailey. “Simultaneous localisation and mapping (SLAM): Part I - the essential algorithms”. Robotics and Automation Magazine, 13(2):99–110, 2006.

[30] S. Russell and P. Norvig, “Artificial Intelligence: A Modern Approach”, Prentice Hall, 1995.

[31] R. Fikes and N. Nilsson, “STRIPS: A new approach to the application of theorem proving to problem solving”, Artificial Intelligence, 2(3-4): 189-208, 1971.

[32] L. Kaelbling, M. Littman and A. Moore, “Reinforcement learning: A survey”, Journal of Artificial Intelligence research, 4:237-285, 1996.

[33] R. Fikes, P. Hart, and N. Nilsson, “Learning and executing generalized robot plans”, Artificial Intelligence, 3(4):251-288, 1972.

[34] A. L. Strehl, M. L. Littman, “An empirical evaluation of interval estimation for Markov decision process”, the 16th IEEE International on Tools with Artificial Intelligence Conference, Seattle: The MIT Press, 531-539, 2004.

[35] H.P. Moravec and A. Elfes, “high resolution maps from wide angle sonars”, IEEE International Conference on Robotics and Automation, pp. 116-121, St. Louis, Missouri, 1985.

[36] M. T. J. Spaan and F. A. Oliehoek. “The MultiAgent Decision Process toolbox: software for decision-theoretic planning in multiagent systems”. In Multi-agent Sequential Decision Making in Uncertain Domains. Workshop at AAMAS08, 2008.

[37] D. Szer, F. Charpillet, and S. Zilberstein. “MAA*: A heuristic search algorithm for solving decentralized POMDPs”. In UAI, 2005.

[38] J. Pearl. “Heuristics: Intelligent search strategies for computer problem solving”. Addison-Wesley, 1984.

[39] A. Stentz, “Optimal and Efficient Path Planning for Partially-Known Environments”, IEEE international Conference on robotics and Automation, the Robotics Institute, Pittsburgh, PA 15213, May 1994.

[40] R. Pepy and A. Lambert, “Safe Path Planning in an Uncertain-Configuration Space using RRT”, Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems October 9 - 15, Beijing, China, 2006.

[41] L. Jaillet, J. Cortes, and T. Simeon, “Sampling-Based Path Planning on Configuration-Space Costmaps”, IEEE Transactions On Robotics, VOL. 26, NO. 4, AUGUST 2010.

[42] I. Nourbakhsh, R. Powers, and S. Birchfield, “Dervish: An office-navigation robot”, AI Magazine, 16(2):53-60, 1995.

[43] M. Littman, A. Cassandra and L. Kaelbling, “Learning policies for partially observable environments: Scaling up”, In Proc. Intl. Conference on Machine Learning, pages 362-370, 1995.

[44] A. Cassandra, L. Kaelbling and J. Kurien, “Acting under uncertainty: Discrete Bayesian models for mobile robot navigation”, In Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, volume 2, pages 963-972, 1996.

[45] N. Roy and S. Thrun, “Coastal navigation – mobile robot navigation with uncertainty in dynamic environments”, In Proc. IEEE Intl. Conf. on Robotics and Automation, pages 35-40, 1999.

[46] C. Liu, J. Chang, G. Li, and C. Liu, “Mobile Robot Path Planning Based on an Improved Rapidly-exploring Random Tree in Unknown Environment”, Proceedings of the IEEE International Conference on Automation and Logistics Qingdao, China September 2008.

[47] G. J. Maeda, S. P. N. Singh, H. Durrant-Whyte, “A Tuned Approach to Feedback Motion Planning with RRTs under Model Uncertainty”, 2011 IEEE International Conference on Robotics and Automation Shanghai International Conference Center May 9-13, Shanghai, China, 2011.

[48] L. J. Guibas, D. Hsu, H. Kurniawati, and E. Rehman, “Bounded Uncertainty Roadmaps for Path Planning”, IN Proc. Int. Workshop on the Algorithmic Foundations of Robotics, 2008.

[49] M. Zucker, J. Kuffner, and M. Branicky, “Multipartite RRTs for Rapid Replanning in Dynamic Environments”, Proc. IEEE Int. Conf. Robotics and Automation, April, 2007.

[50] N. A. Melchior and R. Simmons, “Particle RRT for Path Planning with Uncertainty”, 2007 IEEE International Conference on Robotics and Automation, April, 2007, pp. 1617-1624.

[51] C. Fulgenzi, A. Spalanzani, and C. Laugier, “Probabilistic Rapidly-exploring Random Trees for autonomous navigation among moving obstacles.”, in Workshop on safe navigation, IEEE International Conference on Robotics and Automation, ICRA, 2009.

[52] R. Pepy, M. Kieffer, E. Walter, “Reliable Robust Path Planning With Application To Mobile Robots”, Int. J. Appl. Math. Comput. Sci., 2009, Vol. 19, No. 3, 413–424 DOI: 10.2478/v10006-009-0034-2.

[53] E. Hansen, “Solving POMDPs by searching in policy space”, In UAI 1998, pages 211-219, 1998.

[54] B. Bonet and H. Geffner, “Planning with incomplete information as heuristic search in belief space”, In Proc. Intl. Conf. on AI Planning and Scheduling, pages 52-61, 2000.

[55] H. Geffner and B. Bonet, “Solving large POMDPs by real time dynamic programming”, In Proc. Fall AAAI Symposium on POMDPs, pages 61-68, 1998.

[56] M. Kearns, Y. Mansour and A. Ng, “A sparse sampling algorithm for near optimal planning in large Markov decision processes”, In Proc. Intl. Joint Conf. on Artificial Intelligence, pages 1324-1331, 1999.

[57] S. Paquet, L. Tobin and B. Chaib-draa, “An online POMDP algorithm for complex multi-agent environments”, In Proc. Intl. Joint conf. on Autonomous Agents and Multi-Agent Systems (AAMAS), pages 970-977, 2005.

[58] N. Nilsson, “Principles of Artificial Intelligence”, Tioga, Palo Alto, 1980.

[59] J. Pearl. “Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference”, Morgan Kaufmann, San Mateo, California, 1988.

[60] H. Attias, “Planning by probabilistic inference”, In International Conference on Artificial Intelligence and Statistics, 2003.

[61] X. Li, W. Cheung, and J. Liu, “Towards solving large-scale POMDP problems via spatio-temporal belief state clustering”, In Proceedings of IJCAI-05 Workshop on Reasoning with Uncertainty in Robotics (RUR05), 2005.

[62] X. Li, W. K. Cheung, and J. Liu, “Improving POMDP Tractability via Belief Compression and Clustering”, IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, Vol, 40, No. 1, February 2010.

[63] G. Rens. “Towards real-time policy generation with belief state clustering for dynamic POMDP environments”. Technical Report KRR-11-02, Knowledge Representation and Reasoning, CSIR Meraka, Pretoria, South Africa, March 2011.

[64] F. A. Oliehoek, S. Whiteson, and M. T. J. Spaan, “Lossless clustering of histories in decentralized POMDPs”. In Proc. Of the Eighth Int. Joint Conf. on Autonomous Agents and Multiagent Systems, 2009.

[65] H. Gener and B. Bonet. “Solving large POMDPs using real time dynamic programming”. In Working Notes of AAAI Fall Symposium on POMDPs, 1998.

[66] T. Smith and R. Simmons. “Heuristic search value iteration for pomdps”. In Proceedings of the 20th conference on Uncertainty in artificial intelligence, UAI’04, pages 520~527, Arlington, Virginia, United States. AUAI Press, 2004.

[67] G. Shani, R. Brafman, and S. Shimony. “Forward search value iteration for POMDPs”. In IJCAI-07, pages 2619~2624, 2007.

[68] P. Poupart and C. Boutilier. “Value-directed compression of POMDPs”. In Advances in Neural Information Processing Systems (NIPS 2003), pages 1547~1554. MIT Press, Massachusetts/England, 2003.

[69] N. Roy and G. Gordon. “Finding approximate POMDP solutions through belief compressions”. Artificial Intelligence Research, 23:1~40, 2005.

[70] X. Li, W. K. W. Cheung, J. Liu, and Z. Wu. "A novel orthogonal NMF-based belief compression for POMDPs". In Proceedings of the 24th International Conference on Machine Learning, ICML'07, pages 537~544, New York, NY, USA, 2007. ACM.

[71] H. Wang and S. Julier, "Path Planning In Partially Known Environments", Proceedings of the IASTED International Conference November 7 - 9, 2011 Pittsburgh, USA.

[72] J. Fink, N. Michael, A. Kushleyev and V. Kumar, "Experimental Characterization of Radio Signal Propagation in Indoor Environments with Application to Estimation and Control", In Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems, St. Louis, MO, October 2009.

[73] Gong, D., Lu, L., Li, M., "Robot Path Planning in Uncertain Environments Based on Particle Swarm Optimization". Proc. IEEE Congress on Evolutionary Computation, p.2127-2134, 2009.

[74] P. Hart, N. Nilsson, and B. Raphael. "A formal basis for the heuristic determination of minimum cost paths". IEEE Transactions on Systems Science and Cybernetics, 2:100-107, 1968.

[75] V. Boor, N. H. Overmars, and A. F. van der Stappen. "The gaussian sampling strategy for probabilistic roadmap planners". In IEEE Int. Conf. Robot. & Autom., pages 1018-1023, 1999.

[76] L. Murphy and P. Newman, "Planning Most-Likely Paths From Overhead Imagery," in Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA), Anchorage, AK, USA, pp. 3059–3064, 3–8 May 2010.

[77] The map of Yale Law School 1st Lefel Plan Map, from webpage: http://www.law.yale.edu/images/YLSmap.jpg

[78] M. S. Nixon, A. S. Aguado, "Feature Extraction and Image Processing", An imprint of Butterworth-Heinemann Linacre House, Jordan Hill, Oxford OX2 8DP 225 Wildwood Avenue, Woburn, MA 01801-2041, A division of Reed Educational and Professional Publishing Ltd

[79] P. M. Mather, "Computer Processing of Remotely Sensed Images, An Introduction". West Sussex. John Wiley & Sons Ltd, 2004.

[80] S. K. Gan, K. Yang, and S. Sukkarieh, "3D Path Planning for a Rotary Wing UAV using a Gaussian Process Occupancy Map," Australasian Conference on Robotics and Automation (ACRA), Sydney, Australia, 2–4 December 2009.

[81] B. O. Koopman. "Search and Its Optimization". The American Mathematical Monthly, 86(7):527–540, 1979.

[82] L.D. Stone. "Theory of optimal search". MAS of INFORMS, 2nd edition, 2007.

[83] F. Bougault, T. Furukawa, and H. Durrant-Whyte. "Optimal search for a lost target in a Bayesian world". In S. Yuta et al., editor, Field and service robotics, volume 24 of STAR, pages 209–222. Springer, 2006.

[84] F. Bourgault, T. Furukawa, and H. Durrant-Whyte. “Coordinated decentralized search for a lost target in a Bayesian world”. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 48–53, 2003.

[85] B. Lavis, T. Furukawa, and H. Durrant-Whyte. “Dynamic space reconfiguration for Bayesian search and tracking with moving targets”. Autonomous Robots, 24:387–389, 2008.

[86] E. Wong, F. Bourgault, and T. Furukawa. “Multi-vehicle Bayesian search for multiple lost targets”. In Proceedings of the IEEE International Conference on Robotics and Automation, pages 3169–3174, 2005.

[87] C. Undeger, F. Polat, and Z. Ipekkan, “Real-Time Edge Follow: A New Paradigm to Real-Time Path Search”. In Proceedings of GAME-ON 2001, London, England, 2001.

[88] C. Undeger, and F. Polat, “Real-Time Edge Follow: A Real-Time Path Search Approach”. IEEE Transaction on Systems, Man and Cybernetics, Part C, 2007.

[89] R. He, A. Bachrach, andN. Roy,“Efficient planning under uncertainty for a target-tracking micro-air vehicle”. In Proc. IEEE international conference on robotics and automation, 2010.

[90] T. Maddula, A. A. Minai, M. M. Polycarpou, “Multi-target Assignment and Path Planning for Groups of UAVs”, Recent Developments in Cooperative Control and Optimization, Jan 2004.

[91] D. Jacques, “Search, classification and attack decisions for cooperative wide area search munitions”, Proc. Cooperative Optimization and Control Workshop, 1998.

[92] H. Lau, S. Huang, and G. Dissanayake, “Optimal Search for Multiple Targets in a Built Environment”, IEEE/RSJ International conference on Intelligent Robots and Systems, 2005.

[93] G. Zuo, P. Zhang, and J. Qiao, “Path Planning Algorithm Based on Sub-Region for Agricultural Robot”, 2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics, 2010.

[94] F. Lingelbach, “Path Planning using Probabilistic Cell Decomposition”, Proceedings d the 2004 IEEE International Conference on RobOtl~6. Automation New Orloans. LA -April 2004.

[95] H. Wang, S. Julier, “Reducing the Computational Cost of a Monte Carlo Based Planning Algorithm”, IEEE International Conference on Systems, Man, and Cybernetics (SMC) 2013, 13-16 October, 2013, Manchester, UK.

[96] S. Koenig, M. Likhachev, “Fast Replanning for Navigation in Unknown Terrain”, IEEE Transaction on Robotics, 21(3):354-363, 2005.

[97] L. Brillouin, “Science & Information Theory. Dover Publications”. p. 293. ISBN 978-0-486-43918-1, 2004.

[98] S. Candido and S. Hutchinson, “Minimum Uncertainty Robot Path Planning using a POMDP Approach”, The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems October 18-22, 2010, Taipei, Taiwan.

[99] E. Dijkstra. “A note on two problems in connexion with graphs”. Numerische Mathematik, 1:269-271, 1959.

[100] R. Dechter, N. Flerova, and R. Marinescu, “Search algorithms for m-best solutions for graphical models”. In Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012.

[101] B. Bollobás, O. Riordan, “Percolation”, Cambridge University Press, ISBN 0521872324, 2006.

[102] D. Dolgov, S. Thrun, M. Montemerlo and J. Diebel, “Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments”, The International Journal of Robotics Research 2010 29: 485 originally published online 25 January 2010, DOI: 10.1177/0278364909359210

[103] N. Buniyamin, N. Sariff, W. A. J. Wan Ngah, Z. Mohamad, “Robot global path planning overview and a variation of ant colony system algorithm”, International Journal of Mathematics and Computers in Simulation, Issue 1, Volume 5, 2011.

[104] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, “Practical search techniques in path planning for autonomous driving”. In Proceedings of the First International Symposium on Search Techniques in Articial Intelligence and Robotics (STAIR-08), Chicago, USA, June 2008. AAAI.

[105] M. V. Menshikov, "Coincidence of critical points in percolation problems", Soviet Mathematics Doklady 33: 856–859.

[106] M. Aizenman, D. Barsky, "Sharpness of the phase transition in percolation models", Communications in Mathematical Physics 108 (3): 489–526, Bibcode:1987CMaPh.108..489A, doi:10.1007/BF01212322, 1987.

[107] W. Mohibullah and Simon J. Julier. "Developing an Agent Model of Missing Person in Wilderness." In System Man and Cybernetics (SMC), 2013 Conference on, pp. 1-8. IEEE, 2013.

[108] W. Mohibullah and Simon J. Julier. "Stigmergic search for a lost target in wilderness." In Sensor Signal Processing for Defence (SSPD 2011), pp. 1-5. IET, 2011.

[109] W. Mohibullah and Simon J. Julier. "Bearings-only localisation of targets from low-speed UAVs." In Information Fusion (FUSION), 2010 13th Conference on, pp. 1-8. IEEE, 2010.

[110] W. S. Lovejoy, “A survey of algorithmic methods for partially observed Markov decision processes”. Annals of Operations Research, 28(1):47-65, 1991.

[111] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. “Acting optimally in partially observable stochastic domains”. In Proceed- zngs of the Twelfth National Conference on Artifictal Intellzgence, Seattle, WA, 1994.

[112] M. Littman, A. Cassandra, and L. Kaelbling. “Learning policies for partially observable environments: Scaling up”. In Machine Learning: Proceedings of the Twelfth International Conference, pages 362-370, San Francisco, CA, 1995. Morgan Kaufmann.

[113] S. White and P. Smyth, "A Spectral Clustering Approach To Finding Communities in Graphs", In SIAM International Conference on Data Mining, 2005.

[114] T. Bektas. “The multiple traveling salesman problem: an overview of formulations and solution procedures”. Omega, 34:209–219, 2006.

[115] B. Brumitt, and A. Stentz, “Dynamic Mission Planning for Multiple Mobile Robots,” Proceedings of the IEEE International Conference on Robotics and Automation, Minneapolis, MN, April 1996.

[116] A. H. Land and A. G. Doig (1960). "An automatic method of solving discrete programming problems". Econometrica 28 (3). pp. 497–520.M. Held; R. R. Karp, “A Dynamic Programming Approach to Sequencing Problems”, Journal of the Society for Industrial and Applied Mathematics, 1962, 10 (1): 196–210, doi:10.1137/0110015

[117] Thomas Whelan, Michael Kaess, Hordur Johannsson, Maurice Fallon, John J. Leonard, and John McDonald. Real-Time Large Scale Dense RGB-D SLAM with Volumetric Fusion. International Journal of Robotics Research, December 2014.