Robust Multi-UAV Planning in Dynamic and Uncertain Environments

by

Chung Tin

B.Eng. in Mechanical Engineering
The University of Hong Kong, 2002

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2004

Author ....
Department of Mechanical Engineering
August 10, 2004

Certified by ....
Jonathan P. How
Associate Professor of Aeronautics & Astronautics
Thesis Supervisor

Certified by ....
Daniela Pucci de Farias
Assistant [illegible] al Engineering
Thesis Reader

Accepted by ....
Ain A. Sonin
Chairman, Departm[illegible] uate Students

Robust Multi-UAV Planning in Dynamic and Uncertain Environments

by

Chung Tin

Submitted to the Department of Mechanical Engineering
on August 10, 2004, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

Future *unmanned aerial vehicles* (UAVs) are expected to operate with higher level of autonomy to execute very complex military and civilian applications. New methods in planning and execution are required to coordinate these vehicles in real-time to ensure maximal efficiency of the team activities. These algorithms must be fast to enable rapid replanning in a dynamic environment. The planner must also be robust to uncertainty in the situational awareness. This thesis investigates the impact of information uncertainty and environmental changes to the task assignment and path planning algorithms. Several new techniques are presented that both speed up and embed robustness into previously published algorithms. The first is an incremental algorithm that significantly reduces the time required to update the cost map used in the task assignment when small changes occur in a complex environment. The second introduces a new robust shortest path algorithm that accounts for uncertainty in the arc costs. The algorithm is computational tractable and is shown to yield performance and robustness that are comparable to more sophisticated algorithms that are not suitable for real-time implementation. Experimental results are presented using this technique on a rover testbed. This thesis also extends a UAV search algorithm to include moving targets in the environment. This new algorithm coordinates a team of UAVs to search an unknown environment while balancing the need to track moving targets. These three improvements have had a big impact because they modify the Receding Horizon Mixed-Integer Linear Programming (RH-MILP) control hierarchy to handle uncertainty and properly react to rapid changes in the environment. Hence, these improvements enable the RH-MILP controller to be implemented in more realistic scenarios.

Thesis Supervisor: Jonathan P. How
Title: Associate Professor of Aeronautics & Astronautics

Acknowledgments

I would like to thank my advisor, Professor Jonathan How, who guided me through this work with a lot of insight. Also, the support of the members in the research group is very much appreciated. In particular, Yoshiaki Kuwata and Luca Bertuccelli have provided many inputs to my work.

I would also like to thank Croucher Foundation for financially supporting my study in the past two years.

Last but not least, my deep thanks to my family and my friends for their support and care.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The use of *unmanned aerial vehicles* (UAVs) is evolving quickly [1, 2]. Recent advances in computing, wireless communications, and vehicle technologies have made it possible to deploy a multi-UAV system to operate autonomously to perform very complex tasks in a cooperative manner [2, 3, 4, 5, 6]. It is expected that the use of multi-UAV systems will further develop for both military and civilian applications. This interest in future UAV applications requires new methods in planning and execution in order to perform optimal coordination of the vehicles in real-time. Moreover, in reality, UAV missions usually involve a dynamic and uncertain environment, so the control architecture must be sufficiently robust and flexible to handle uncertainty and changes in the environment.

1.1 Receding Horizon Mixed-Integer Linear Programming (RH-MILP)

In a multi-vehicle system, the coupling between the vehicles, targets and obstacles creates a very complicated optimization problem [2, 9] in which the computation time increases dramatically with problem size. A *Receding Horizon Mixed-Integer Linear Programming* (RH-MILP) control hierarchy (Fig. 1-1) has been developed in

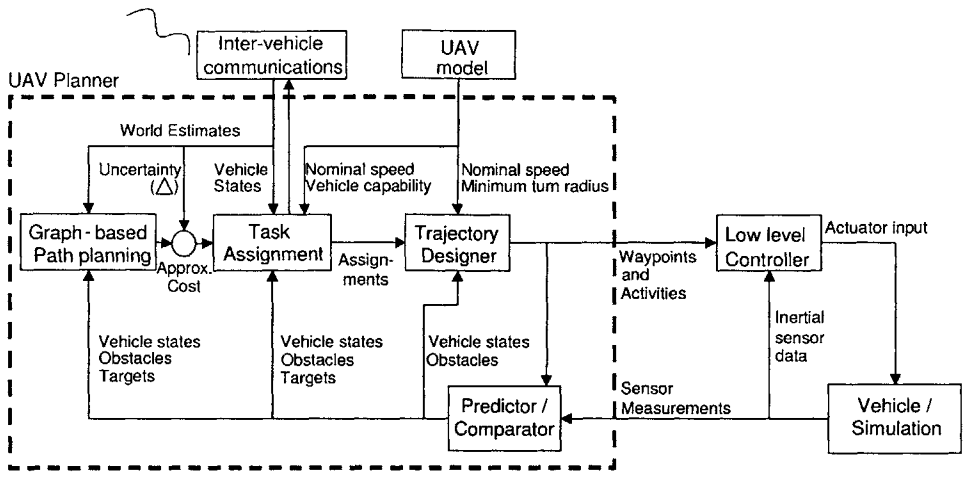


Figure 1-1: System Architecture [13].

the Aerospace Controls Lab. (ACL) at MIT to handle these UAV problems in real-time. MILP allows the inclusion of non-convex constraints and discrete decisions in the optimization problem. The *receding horizon control* (RHC) method is used to overcome the computational burden of MILP in the multi-vehicle optimization problem. Previous work demonstrated the use of RH-MILP in trajectory design and task allocation under various dynamic, kinematic, capability and timing constraints in real-time [7, 13, 14, 16, 17].

1.1.1 Receding Horizon Trajectory Design (RH-Traj)

The use of MILP to design an entire trajectory with a planning horizon fixed at the goal is very difficult to perform in real-time, since the complexity of the problem grows rapidly with the length of the route and the number of obstacles to avoid [13]. RHC overcomes this limitation by using a receding planning horizon in which MILP is used to form a shorter plan that extends towards the goal (Fig. 1-2). A heuristic is introduced to approximate the trajectory beyond the planning horizon using straight line paths. Ref. [13] presented a trajectory design algorithm using RH-MILP to guarantee a *kinodynamically feasible* and *stable* path to the goal in a highly constrained environment, and demonstrated that the trajectory design is applicable

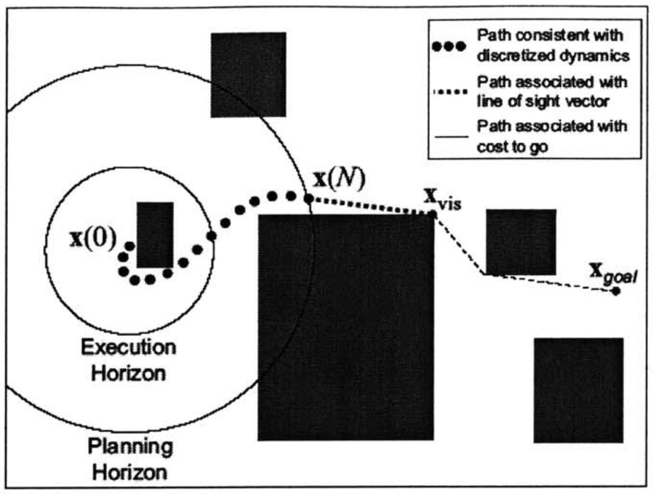

●●● Path consistent with discretized dynamics
..... Path associated with line of sight vector
___ Path associated with cost to go
x(N)
x_vis
x(0)
x_goal
Execution Horizon
Planning Horizon

Figure 1-2: Illustration of RH-Traj planning algorithm [13].

in real-time.

1.1.2 Receding Horizon Task Assignment (RHTA)

The UAV task assignment problem is essentially a *multiple-choice multi-dimension knapsack problem* [13]. However, adding vehicle capability and task timing constraints significantly complicates the optimization problem. This complete task assignment can be formulated in MILP [13, 15] by encoding the mission objectives and constraints as a combination of binary and continuous variables. To overcome the computation burden for the task assignment, the *Receding Horizon Task Assignment* (RHTA) algorithm attempts to solve a suboptimal problem with smaller size [15]. Instead of considering all of the possible permutations of the tasks, RHTA only looks at permutations which contain no more than m tasks, where $m \ll N$ and N is the total number of tasks in the problem. Optimization is performed for only this subset of the problem, and there is a *cost-to-go* added to the problem to ensure feasibility of the remaining tasks. The algorithm then iterates forward until all of the tasks have been assigned. The solution is now suboptimal, but reducing m significantly decreases the computational load. Ref. [15] shows that $m = 2$ appears to offer the best trade-off between performance and computation speed.

1.2 Planning in Uncertain and Dynamic Environments

In reality, UAVs operate in a dynamic and uncertain environment. As the mission proceeds, new information will be obtained through the onboard sensors. For example, there may be new targets discovered, or there is a new no-fly zone (or obstacle) in the area. Hence, the optimizations for task assignment and trajectory design will have to be repeated to incorporate the new information in order to maintain the optimality (or even feasibility) of the decision. Since the replan is performed online, the optimization process has to be fast such that the UAVs can properly react to the changes. When the environment is very complex, the computational load for the optimization is large. Hence, it is necessary to overcome this computation difficulty in the re-planning process.

Of course, the environment is not only dynamic but also uncertain. Uncertainty enters the problem as sensor errors and signal noise. The planner has to be robust to these uncertainties to maintain feasibility and optimality. Moreover, the robust planner has to be fast enough for real-time application. However, adding robustness typically increases the complexity of the optimization problem [12]. Hence, an approximation algorithm may be required to trade-off performance vs. computational load. This thesis extends the previous work to develop an approximate robust optimization tool for the RH-MILP problem.

1.3 Thesis Overview

This thesis presents several algorithms to handle uncertainty and changes in the UAV environment that can be implemented in the RH-MILP hierarchy. Chapter 2 presents an **incremental algorithm** to update the visibility graph in order to speed up the cost map calculation during replanning. The cost map and the visibility graph are key inputs to the RH-Traj and RHTA algorithms. Chapter 3 presents a **robust shortest**

path algorithm to account for uncertainty in the arcs of the cost map. The new algorithm yields robustness and performance that are comparable to more sophisticated algorithms that are not practical for real-time implementation. Chapter 4 discusses a multi-UAV search mission. This work extends [32, 33, 35] to include moving targets in the environment, which greatly complicates the problem. By adding tracking as a task, we demonstrate that the algorithm can coordinate the UAVs to search an environment and balance that against the need to track the known targets. The problem is solved as a single optimization using the environment uncertainty as the common cost. Finally, Chapter 5 discusses the rover testbed developed in the Aerospace Controls Lab., MIT and the experimental results of implementing the robust shortest path algorithm in Chapter 3 on the testbed.

Chapter 2

Incremental Update of Cost Map

In UAV mission, the preliminary planning will be based on some *a priori* information. As the mission proceeds, new knowledge about the terrain may be obtained, which can include finding new targets, finding new Surface-to-Air-Missile (SAM) sites and so on. As a result, the task allocation may need to reschedule such that the vehicles visit the targets with the maximum efficiency. Also, some vehicles may need to be assigned to clear the SAM site to reduce the risk of the mission. In summary, the previous plan may no longer be optimal or feasible given this new information. Hence, to maximize performance, the task assignment and path planning would require re-optimization.

In the RH-MILP framework, the vehicle and environment states are continuously tracked by the *BRAIN*. When new information is obtained by the vehicles, the BRAIN sends a re-plan request to the *Task Assignment* (TA) to re-optimize the plan, and sends the new assignments to the *Path Planner* to re-plan the trajectory, as illustrated in Fig. 2-1. The optimization is done based on the *cost map* which describes the cost to travel between the *nodes* of interest in the scenario. "Nodes" (or cost points) here refer to the locations of targets, vehicles, and obstacle corners [13, 15].

The *Receding Horizon Task Assignment* (RHTA) [15], which is formulated as a MILP problem, is a main element in the RH-MILP framework. It is a relative fast algorithm for solving task assignment problems with timing and capability constraints.

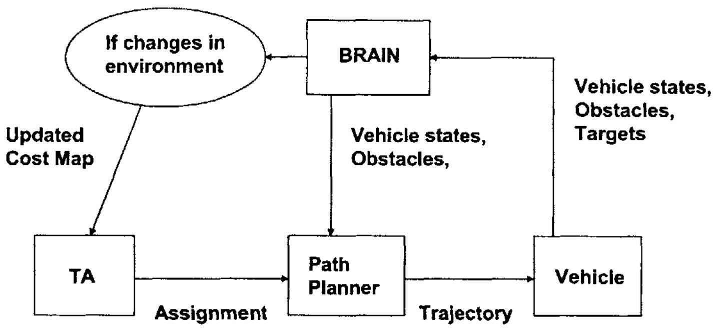

If changes in environment
BRAIN
Vehicle states, Obstacles, Targets
Updated Cost Map
Vehicle states, Obstacles,
TA
Path Planner
Vehicle
Assignment
Trajectory

Figure 2-1: The replan mechanism of RH-MILP.

This has been demonstrated in real-time experiment for dynamic environment [25, 13]. However, the computation time is still too long for more complex and more rapidly changing environment. The first task assignment is allowed to take a long time to compute since it is done off-line before the mission starts. However, replan is computed online. Hence, in order to implement the algorithm in real time, the replan has to be fast. It is observed that the re-computation of the cost map has taken a significant portion of the total computation time. Increasing the speed of this calculation would bring significant improvement in the overall computation speed.

This chapter presents a simple, yet effective way for re-computing this cost map with the new information. Section 2.2 addresses the essential observations for updating the cost map with various possible changes in the environment. These observations motivate to speed up the algorithm. Section 2.3 gives an example of a fairly complex multi-vehicle scenario to demonstrate the effectiveness of this approach.

2.1 Motivation

UAVs usually operate in dynamic environments in which information gets updated over time, and the previous plan is often no longer optimal as new information is obtained. The RH-MILP algorithm has demonstrated its capability to tackle these dynamics and re-optimize the plan according to these changes [25, 13].

As the environment becomes more complex, the computation load for planning grows. In addition, if the environment is highly dynamic, namely changes occur very rapidly, the planner may not be fast enough to properly react to these changes. So, we are concerned about speeding up the planning algorithm, especially during re-plan, which has to be done online.

The cost map serves as the main input to the RHTA. The cost map describes the connectivity and cost of travel among nodes. It is required that each time a re-plan request is called, this cost map be recomputed. It is noticed that the computation of this cost map has taken a large percentage of the total planning time. Fig. 2-2 compares the total planning time and the computation time for the cost map for the example shown in Section 2.3. It can be seen that the computation of the cost map takes up about 50% of the total planning time. This becomes a limiting factor for real-time implementation.

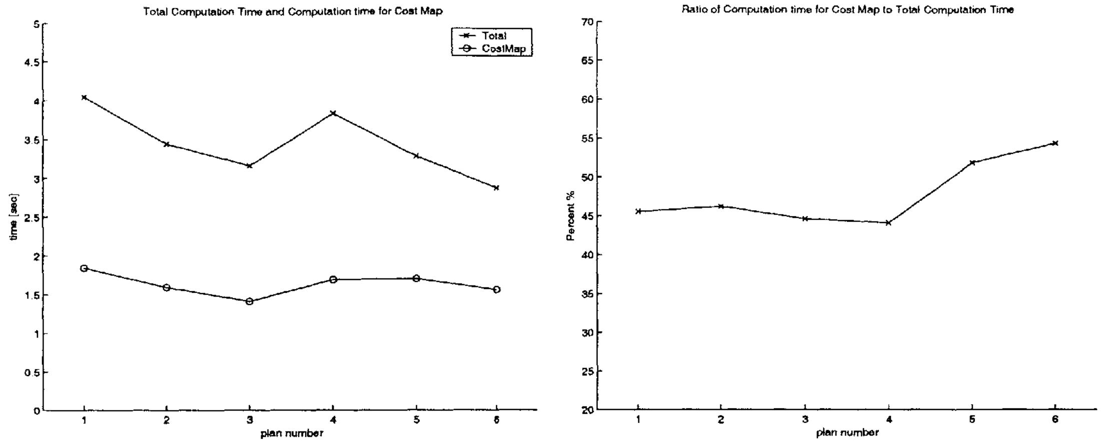
Moreover, it has been observed that on a short-time scale, the changes in the environment are often confined to a local region. Most of the information in the cost map is still valid. Hence, it is not reasonable to re-compute the entire cost map every time. This part can likely be spedup if we can identify these local regions under changes and only update the cost map for them.

Expected changes in environment for the RH-MILP problem can be summarized as follows,

1. Pop up new obstacles.
2. Removal of obstacles.
3. Pop up new targets.
4. Removal of targets.
5. Loss of vehicles.
6. Vehicle movement.

Each of these types of changes has a specific impact on the cost map. By identifying these specific impacts, a lot of work can be saved for updating the cost map, hence, speeding up the re-optimization of plan.

The following section discusses the impact of these dynamics on the cost map.



(a) Total computation time Vs. Computation time for cost map

(b) Ratio of computation time for cost map to total computation time.

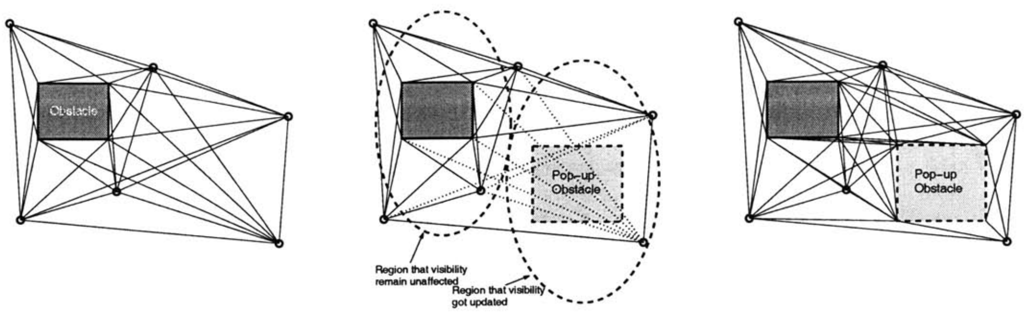
Figure 2-2: Computation load required for cost map.

An *incremental* update of the cost map is presented which effectively reduces the re-computation time of the cost map.

2.2 Incremental Update of Visibility Graph

The computation of the cost map in RHTA consists of two main parts, namely the generation of the *visibility graph* and running the *Dijkstra's* algorithm. The visibility graph describes how the nodes are connected to each other and the cost of each of these connected pairs. The Dijkstra's algorithm returns the path of minimum cost between each pair of nodes in the scenario. The Dijkstra's algorithm is generally fast for RHTA. However, there are still algorithms that can speed up the shortest path algorithm in a dynamic environment [26, 27, 28]. This section focuses on speeding up the generation the visibility graph in a dynamic environment as it is a much slower algorithm than the Dijkstra's algorithm. (Generally 4 - 5 times slower)

As mentioned in Section 2.1, we attempt to develop an incremental method of updating the visibility graph which would speed up the re-computation. To develop this incremental method, we need to understand the specific impact of each type of


Obstacle
Pop-up Obstacle
Region that visibility remain unaffected
Region that visibility got updated
Pop-up Obstacle

(a) Original visibility graph. (b) Localized update of visibility graph. (c) Final visibility graph.

Figure 2-3: Illustration of update of visibility graph with a pop-up obstacle.

the possible dynamics in Section 2.1 on the visibility graph.

2.2.1 Obstacles

Pop up new obstacles

Consider the situation when a new obstacle is found in the middle of the mission as illustrated in Fig. 2-3. The role of an obstacle is to constrain the area of movement of the vehicles. A new obstacle reduces the visibility in the environment. Hence, in the sense of visibility graph, the changes occur only on the pairs of nodes that are connected before the pop-up of this new obstacle. Furthermore, the new obstacle only affects the arcs which are "close" to it. This is illustrated in Fig. 2-3(b). To be more precise, "close" refers to arcs that are cross the new obstacle. So, we have to consider arcs (or nodes) that satisfy these two criteria only. Hence, the only work is to check the visibility of each pair of these connected nodes. The problem size has been reduced from $O(N^2)$ (considering all pairs of nodes) to $O(n^2)$ (considering only specific pairs of nodes), where N is the total number of nodes and n is the number of nodes that may require update, and in a complex scenario, it is typically that $n \ll N$.

Removal of obstacles

Removing obstacles would happen, for example, when vehicle finds that the obstacle is actually not present where it is expected, or when one of the vehicles has removed a SAM site which is essentially a no-fly zone, hence an obstacle. The process of updating the visibility graph for removal of obstacles is similar to that for pop-up of new obstacle. Removing an obstacle enhances the visibility in the environment. Hence, the impact is only placed on the nodes that are previously not connected. Furthermore, the changes occur only around the removed obstacle, as illustrated in handling pop-up obstacles. So, the work is to check if these nodes are now connected with the obstacle removed. Similarly, we have reduced the problem size from $O(N^2) \rightarrow O(n^2)$.

2.2.2 Targets

The way to handle targets is much simpler than obstacles, since targets do not interfere with the connectivity of other nodes. This also emphasizes the usefulness of incremental update of visibility graph, as much less work is actually required than computing it from scratch.

Pop up new targets

When new targets are found, the only work to update the visibility graph would be to compute the visibility of the new targets to all other nodes. The computation is only $O(mN)$ rather than $O(N^2)$, where m is the number of new targets and N is the total number of nodes, and in general, $m \ll N$.

Removal of targets

When targets are visited by the vehicles, or when targets are not present where they are supposed to be, the targets are removed from the list of nodes in the visibility graph. No computation for the visibility graph is actually required. It is only necessary to erase the data associated with the removed targets.

2.2.3 Vehicles

Vehicle states are effectively the same as targets in computing the visibility graph. Hence, the way to update the visibility graph associated with vehicle states is just the same as for targets.

Loss of vehicles

When vehicles are destroyed or lose communication, the planner considers it as *dead* and subsequent planning would not consider them any more. The update of the visibility graph for this is the same as for removal of targets.

Vehicle movement

As the mission proceeds, the vehicles move around in the environment. Hence, when a re-assignment is called, the positions of the vehicles have been updated. However, instead of dealing with the entire visibility graph, we have to consider only the visibility graph between the vehicles and all other nodes. The visibility of the vehicles at the new position with all other nodes is checked, and the arc lengths are re-computed as well. The problem size is reduced from $O(N^2)$ to $O(mN)$, where m is the number of vehicles and N is the total number of nodes, and in general, $m \ll N$.

2.3 Effect on Computation Time

This section presents a simulation example to demonstrate the impact on computation time by using the incremental update of visibility graph. The scenario of the example is shown in Fig. 2-4(a). This is a fairly complex and dynamic scenario with multiple re-assignments. The example consists of 4 vehicles ($\square$), 3 obstacles, and 17 targets ($\circ$). The vehicles start from the bottom of the figure and travel upward at a nominal speed of 0.25m/s. Targets 1-8, 10, 12, 13 are known at the beginning. The dynamics of the scenario, indexed by the sequence of re-assignment:

1. Vehicle 2 died soon after the start of the mission.
2. Positions of Targets 8 and 10 were updated by Targets 9 and 11 respectively.

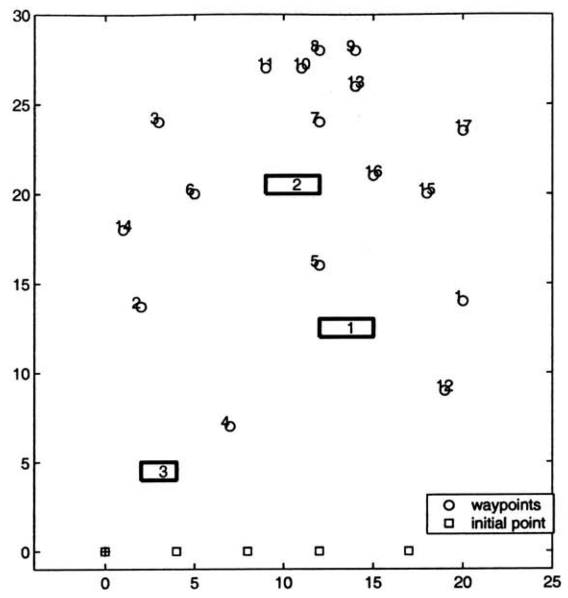
3. Targets 14 and 15 were found.
4. Target 16 was found.
5. Target 17 was found.

Fig. 2-4(b)–Fig. 2-4(m) show the trajectories of the vehicles when the request of new task assignment is sent (figures on the left) and the results of each task assignment (figures on the right). The entire trajectories are shown in Fig. 2-4(n).

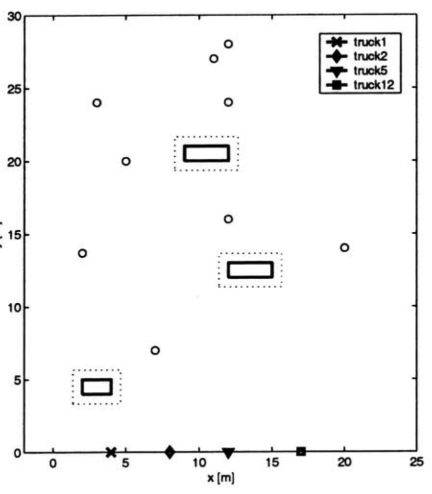
The computation time for the scenario is shown in Fig. 2-5. The computation time using the incremental update and update from scratch are both shown in the figures. Note that the first plan is generated before the start of the mission, the incremental update of visibility is not applied. So, only the second plan and onward are compared. Fig. 2-5(b) shows the computation time to generate the cost map, which is the sum for generating the visibility graph and running the Dijkstra's algorithm. It shows that it took only about 0.15 seconds to compute the cost map using the incremental update method for the visibility graph, while it took about 1.6 seconds to re-compute the cost map from scratch. Hence, we have reduced the computation time for the cost map by about 90%. Fig. 2-5(a) shows the total planning for each task assignment request. It is shown in the figure that the incremental method has reduced the total planning time from 3.3 seconds to 1.6 seconds approximately. Hence, we have obtained a reduction of total planning time of about 50%.

2.4 Conclusions

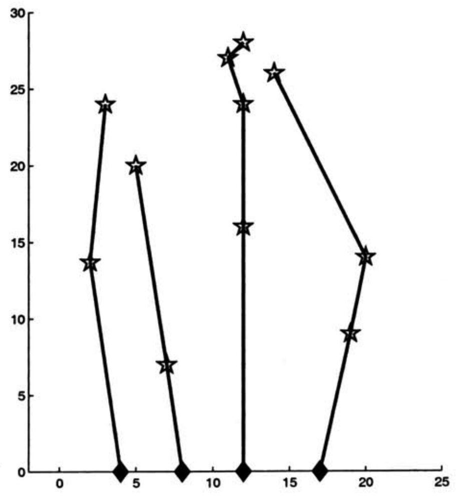
An incremental method of updating the visibility graph is presented in this chapter. Despite of its simplicity, it is efficient in reducing the computation time of re-assignment using RHTA. A simulation example is shown to demonstrate the improvement on computation time using the incremental method. With the reduced computation time, the RHTA will be able to handle more complex and dynamic environments. Hence, the overall RH-MILP algorithm will be able to react to a rapidly changing, complex environment with an optimal planning.



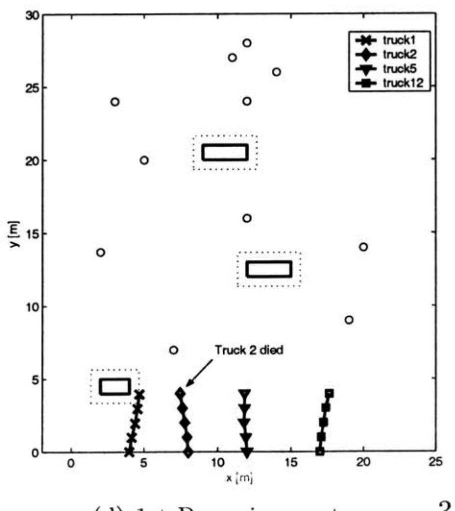
(a) Scenario.



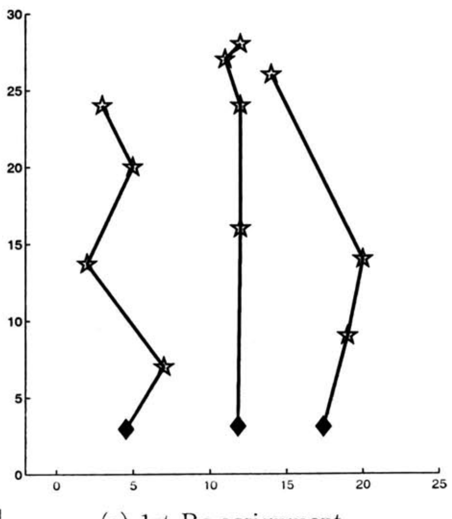
(b) Initial assignment.



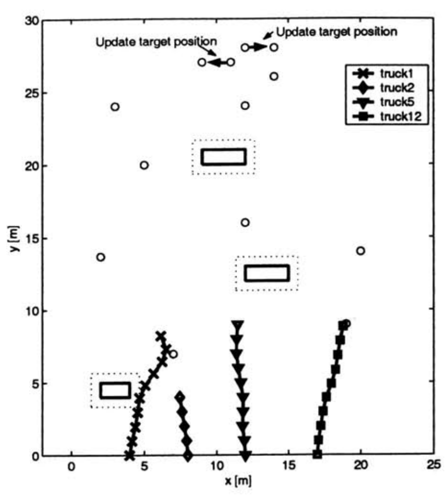
(c) Initial assignment.



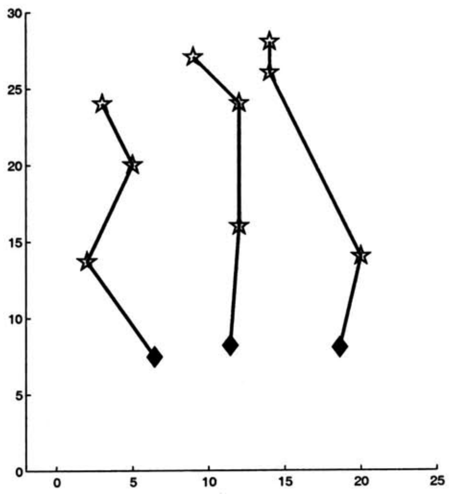
(d) 1st Re-assignment.

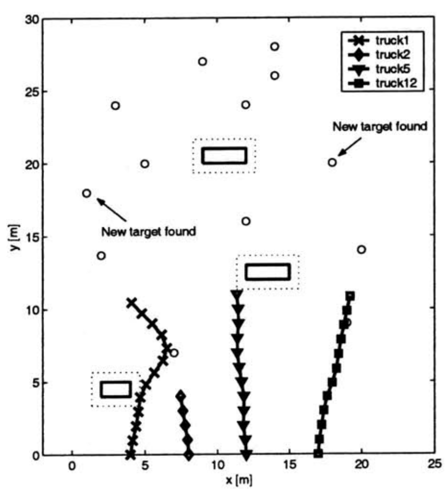


(e) 1st Re-assignment.

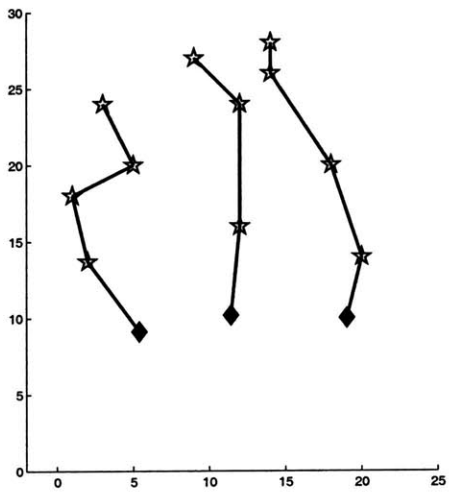


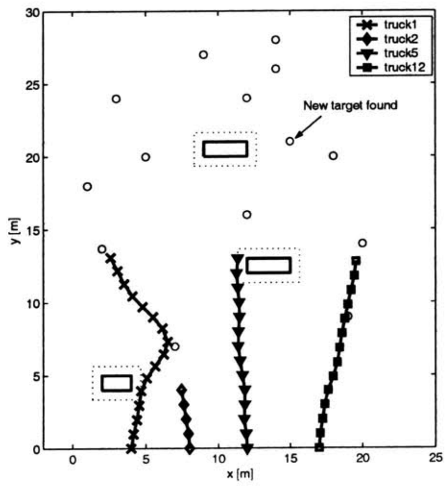
(f) 2nd Re-assignment.


(g) 2nd Re-assignment.

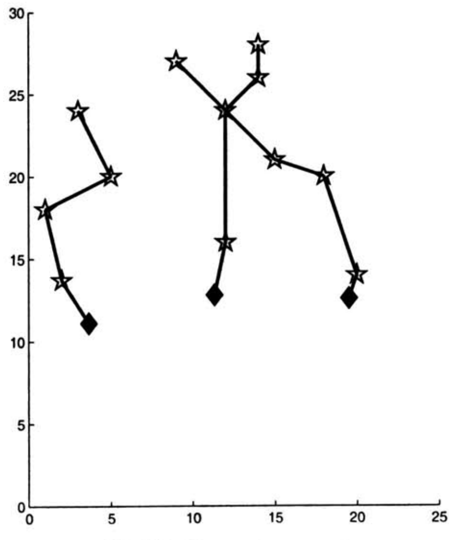


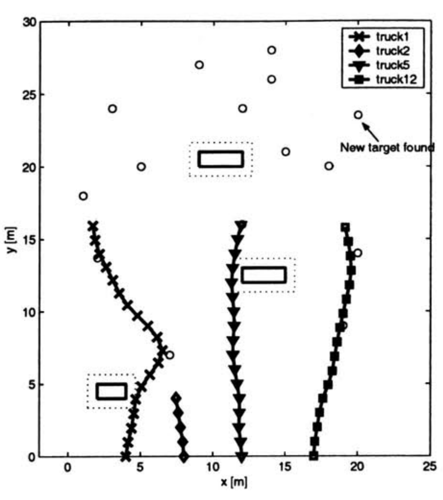
(h) 3rd Re-assignment.


(i) 3rd Re-assignment.

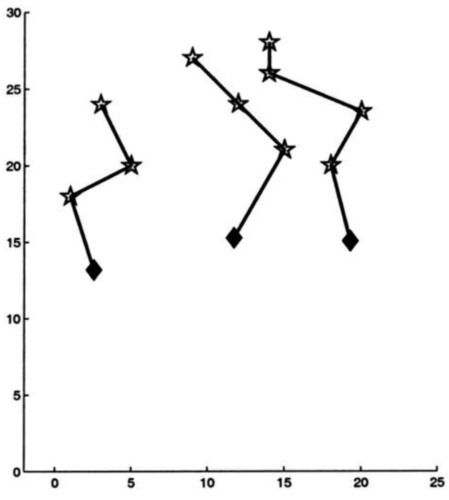


(j) 4th Re-assignment.

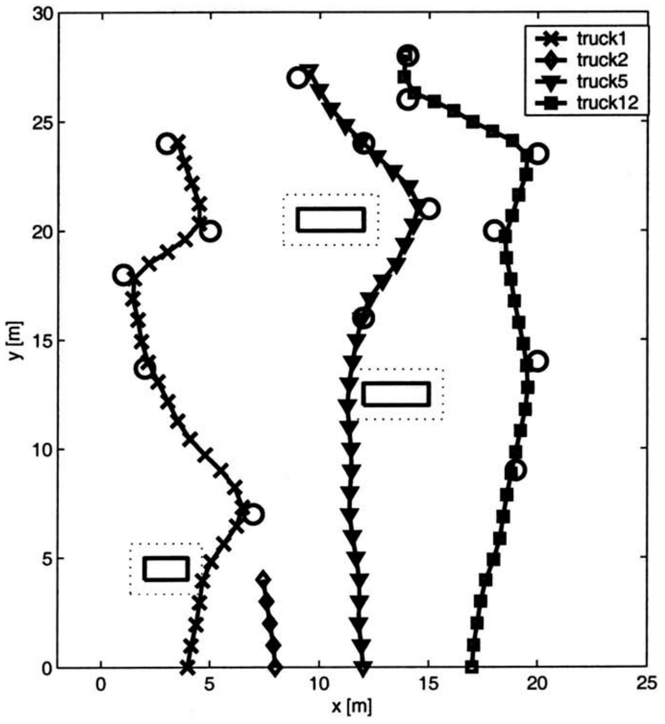

(k) 4th Re-assignment.



(l) 5th Re-assignment.

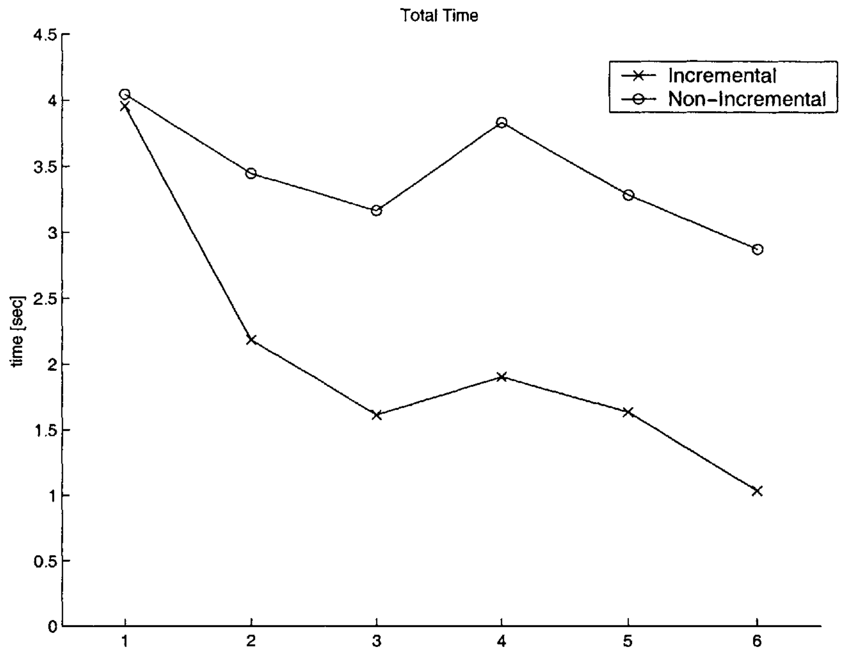


(m) 5th Re-assignment.

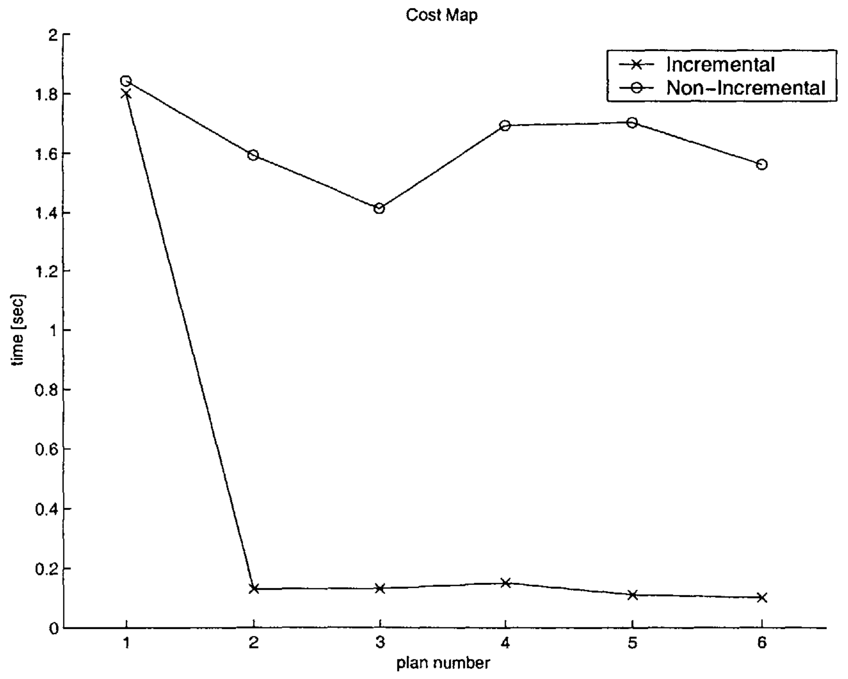


(n) Final trajectory.

Figure 2-4: Example of dynamic scenario.



(a) Total computation time.



(b) Computation time for cost map.

Figure 2-5: Comparison of computation time for update of visibility graph.

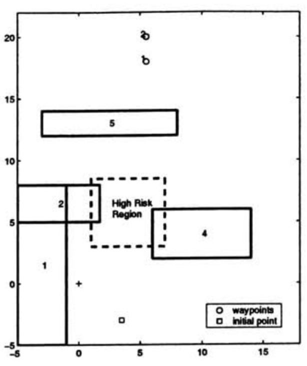
Chapter 3

Approximate Robust Shortest Path Algorithm

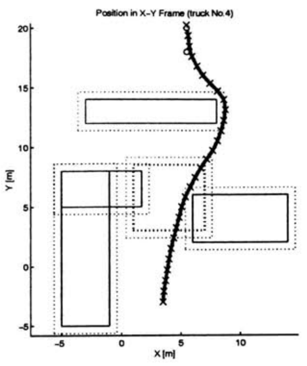
In UAV trajectory planning problem, the objective is often to find a dynamically feasible path which guides the UAV to the goal in the shortest time. Various algorithms can be used to solve this shortest path problem. However, they usually ignore any uncertainty in the problem.

Fig. 3-1 shows an example when uncertainty comes into the UAV path planning problem. The boxes with solid lines are obstacles, and the box with a dashed line is a region of high risk. If the uncertainty assoicated with this high risk region is ignored, the vehicle will go through this high risk region as the distance of travel is minimum. However, when the uncertainty is taken into account, the path planner may tell the vehicle to take a detour instead. The choice clearly depends on the trade-off between traveling time and risk, but this decision must be included in the algorithm.

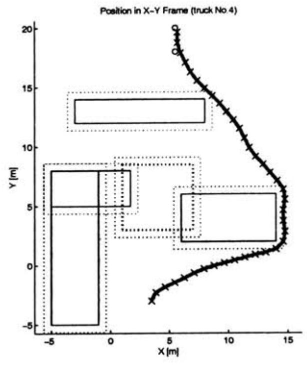
This chapter presents a robust shortest path algorithm based on Dijkstra's algorithm. Section 3.1 gives an introduction to robust optimization, and the notion of *robustness* is discussed. Section 3.4 discusses a systematic approach to choose the robust parameter α. Section 3.5 discusses the formulation of the *Approximate Robust Shortest Path algorithm* (ARSP). ARSP is compared with two other robust



(a) Environment Setup.



(b) Nominal decision: Vehicle goes through the uncertain region.



(c) Robust decision: Vehicle avoids the uncertain region.

Figure 3-1: Robust vs Nominal path planning.

algorithms, namely *Bertsimas-Sim Algorithm* [10] and the *Robust Deviation Shortest Path* Algorithm [11]. Simulations with different problem size are used to verify the performance of the ARSP.

3.1 Overview of Robust Optimization

Decision making in reality always consists of uncertainty. Uncertainty is not an occasional, temporary occurrence in decision planning. A realistic and robust decision cannot be made without understanding and bringing uncertainty into the planning process [12].

One way to solve optimization problem with uncertainty is to use *stochastic optimization*. Stochastic optimization attempts to generate a solution that maximizes (or minimizes) the expected value of the objective function. However, the stochastic formulation recognizes only the central tendency (i.e. the first moment) of the data. This could lead to a problem if two sets of data having the same expectations but different spread (i.e. the second moment). The stochastic formulation sees the two sets of data as being equivalent, but decision maker would probably prefer the one with smaller variance.

The expected values forms only one of the potential realizable scenarios. Under uncertainty, what is required is a decision that performs well over all the possible scenarios. This is known as the *Robust Decision* making. Two formal definitions of robustness approach are discussed in the following.

3.1.1 Absolute Robust Criterion

Absolute robust criterion is also known as the *minimax* criterion. The robust decision is made to minimize (or maximize) the performance of the worst case scenario. Use of this criterion results in a conservative solution which assumes the worst case likely to happen. In mathematics, it is formulated as,

$$J^* = \min_{Y \in Y^F} \max_{s \in S} f(Y, D^s) \tag{3.1}$$

where Y is the set of decision variables and D is the set of input data. D^s denotes the realization of D in scenario s. Y^F denotes the set of all feasible solutions. The absolute robust criterion is applied when the primary concern is about not exceeding a certain target level of performance

3.1.2 Robust Deviation Criterion

Robust deviation criterion is also known as the *minimax regret* criterion. "Regret" can be defined as the difference between the resulting cost (or benefit) from the decision made and the cost (or benefit) from the decision that would have been made if the particular input data scenario had been known a prior. The minimax criterion is then applied to minimize the maximum regret, which is formulated as

$$J^* = \min_{Y \in Y^F} \max_{s \in S} (f(Y, D^s) - f(Y_s^*, D^s)) \tag{3.2}$$

where Y_s^* is the optimal solution under scenario s.

The robust deviation criterion results in a less conservative decision, since it allows benchmarking of the performance of the solution against the best possible outcome under any realization of the data. It can serve as an indicator of how much can be improved if the uncertainty can be removed.

The decision maker should choose between the two robust criteria according to the problem statement. Each of the choice may robustisfy the solution to different extent. The following section discusses the *robust shortest path* problem for RH-MILP and the appropriate robust criterion is chosen for the UAV planning problem.

3.2 Robust Shortest Path Problem for RH-MILP

In the UAV path planning problem, we search for the shortest path connecting the UAV current position to the goal. The *Dijkstra's algorithm* is used to solve the shortest path problem in the RH-MILP formulation [13]. The Dijkstra's algorithm finds the shortest path to all nodes from an origin node on a graph $G = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is a finite set of nodes and $\mathcal{A}$ is a collection of arcs joining a pair of nodes which are members of $\mathcal{N}$. Each arc (i, j) in the set of arcs $\mathcal{A}$ is associated with a cost $c_{ij}(\geq 0)$. This algorithm involves the labeling of a set of fixed nodes $\mathcal{P}$, of which the shortest node distances D_j from the origin node to each node $j \in \mathcal{N}$ has been found. Dijkstra's algorithm is efficient and has a complexity of $O(|\mathcal{N}|^2)$ [23]. To take the advantage of this simplicity, a robust shortest path algorithm, called *Approximate Robust Shortest Path (ARSP) Algorithm*, is developed based on the Dijkstra's algorithm. This choice simplifies the implementation of the robust algorithm in the RH-MILP framework of the UAV planning algorithm.

When uncertainty is incorporated in the cost of the arcs, c_{ij} is no longer a specific number, but $c_{ij} \in C_{ij}$ where C_{ij} is some uncertainty set. The shortest path problem is to obtain the "best" solution across all possible realizations of data. The uncertainty set can be modeled in various ways. One way is to model C_{ij} as the lower bound value, $c_{l,ij}$, and the upper bound value, $c_{u,ij}$.

Given a window of arc cost $c_{ij} = [c_{l,ij}, c_{u,ij}]$, either the absolute robust criterion or the robust deviation criterion can be used to solved the shortest path problem. In the UAV path planning problem, we are concerned about having the vehicle reach the target within a certain time interval. It is often essential that the mission time does not exceed a certain limit for the vehicles to coordinate their maneuvers. This would favour the choice of the absolute robust criterion. In which case, the problem can be formulated as,

$$\begin{aligned} \min_x J &= \sum_{i,j} c_{u,ij} x_{ij} \\ \text{subject to :} \quad & x \in X \end{aligned} \tag{3.3}$$

where X is the constraint set. This formulation generates an extremely conservative solution, since it is unlikely that each arc will indeed achieve its worst case cost. Furthermore, it is unlikely that all the arcs will attain the worst case cost *simultaneously*. So, to be more optimistic, the cost of the arcs are allowed to attain some values somewhat less than $c_{u,ij}$. The formulation then becomes,

$$\begin{aligned} \min_x J &= \sum_{i,j} \beta c_{u,ij} x_{ij} \\ \text{subject to :} \quad & x \in X \end{aligned} \tag{3.4}$$

where $0 < \beta \leq 1$. However, $\beta c_{u,ij}$ captures only a single "side" of the uncertainty set, namely the proximity of the upper bound. It ignores the "shape" of the uncertainty set. For example, consider two arcs with the same $c_{u,ij}$ but different $c_{l,ij}$. Eq. 3.4 takes the two arcs equivalently. However, in reality, the decision maker would probably prefer the one with higher $c_{l,ij}$ since it has a smaller uncertainty set.

Instead of considering only the upper bound value, the uncertainty is *approximated* using the first and the second moments, i.e. mean and variance, of the data set in the ARSP. The mean captures the central tendency of the uncertainty while variance captures the spread of the uncertainty. The data set is hence transformed from $[c_{l,ij}, c_{u,ij}]$ into $[\bar{c}_{ij}, \sigma_{ij}^2]$, where $\bar{c}_{ij}$ and σ_{ij} are the mean and standard deviation of the cost of arc (i, j) respectively. The new representation of the data set implies some distribution of the data. If we have some *a priori* knowledge about the uncertainty, for example, from collected sample data, some distribution functions can be fit into this data set. If we simply have the upper and lower bound values, it would be natural to use some standard distribution functions to model the data set, for example, the *uniform distribution*, or the *normal distribution*. Hence, the robust shortest path problem becomes to minimize the cost function, $f(\bar{c}, \sigma^2)$.

The following section discusses the formulation of the cost function. Since statistics is introduced into the robust shortest path problem, the cost function takes up the well-defined properties of the standard normal distribution. However, the preliminary cost function is nonlinear and non-convex. An approximation method is used to overcome the difficulty arising from the nonlinearity and non-convexity.

3.3 The Performance Index, $\bar{c} + \beta\sigma$

When statistics comes into the problem, we attempt to quantify the quality of the robust solution by the quantity $\bar{c} + \beta\sigma$, where β is some constant to be picked. This performance index serves as an estimate to the cost of the arc, such that it guarantees a certain probability ($P(c < \bar{c} + \beta\sigma)$) that the actual cost falls below this value. The probability depends on the type of distribution. For example, with $\beta = 1$, the corresponding probability for uniform distribution is about 78% and that for normal distribution is about 84% [19]. However, as we are adding up a number of random variables in the solution path, we can get around this variation in probability by applying the *Central Limit Theorem*. Hence, this performance index is independent of the type of distribution.

The Central Limit Theorem [21]

Let $X_1, X_2, \ldots$ be a sequence of independent distributed random variables with mean

$\mu_1, \mu_2, \ldots$ and variance $\sigma_1^2, \sigma_2^2, \ldots$. Define,

$$Z_n = \frac{\sum_i^n (X_i - \mu_i)}{\sqrt{(\sum_i^n \sigma_i^2)}}$$

Then, the CDF of Z_n converges to the standard normal CDF when $n \to \infty$,

$$\lim_{n\to\infty} P(Z_n \le z) = \Phi(z) \quad \forall z$$

where $\Phi(z)$ is the *standard normal cumulative distribution function* (CDF).

The Central Limit Theorem requires only (i) independence and (ii) finite mean and variance for the distribution of X_i.

For a large n, it is fair to assume normal distribution. In general, when $n \ge 30$, the normal approximation is satisfactory regardless of the shape of the population [20]. For the case of *uniform* distribution, $n = 12$ is a reasonable point to assume normality [19].

However, problems arise when n is small. The approximation of normality may give a significant error depends on the type of distribution. Fortunately, this error can be estimated using the *Berry-Esseen Theorem* [18].

The Berry-Esseen Theorem [18]

Assume $E(X_i) = 0$ and $E(|X_i|^3) < \infty$. Then,

$$\begin{aligned} |P_n(z) - \Phi(z)| &\le \frac{cL_n}{(1+|z|)^3} \qquad (3.5) \\ \text{where} \qquad P_n(z) &= P(Z_n \le z) \\ \text{and} \quad L_n &= \frac{\sum_{i=1}^{n} E(|X_i|^3)}{\left(\sum_{i=1}^{n} \sigma_i^2\right)^{3/2}} \end{aligned}$$

where c is some positive constant not less than $(2\pi)^{-0.5}(\approx 0.4)$ [18], and $\Phi(z)$ is the standard normal CDF.

The Berry-Esséen Theorem provides an upper bound for the error between the cumulative distribution of the sum of random variables and the standard normal CDF. Fig. 3-2 shows the predicted upper bound errors with the Berry-Esséen Theorem and the errors from simulation using a uniform distribution with *zero* mean and finite variance. The value of c is taken as 0.4 and the value of z is taken to be 1. Fig. 3-2(a) shows the results when all X_i have identical distribution, and Fig. 3-2(b) shows results with randomly generated variance over the range $[0, \frac{20^2}{3}]$ for each X_i.

From the figures, it can be seen that the Berry-Esséen Theorem provides a reasonably tight bound on the error for the Central Limit Theorem. The predicted error deviates from the simulated error by less than 3%, which is sufficiently small to predict the errors in the simulations. The bound is particularly useful for cases when n is small such that the assumption of normality is not appropriate. Since $\Phi(z)$ can easily be found from the look-up table. The Berry-Esséen Theorem provides a simple way to estimate $P_n(z)$.

Note that the result is not restricted to distributions with zero mean nor to uniform distribution. For distribution with finite mean, the Berry-Esséen Theorem can be applied through the transformation,

$$Y_i = X_i - \mu_i$$

such that Y_i is a random variable with zero mean and variance σ_i^2. Moreover, by choosing different values for the constant c, the theorem can be applied for different distribution models of the uncertainty.

To apply the Berry-Esséen Theorem into the robust shortest path problem, consider C_{ij}, the arc cost, as the random variable. The cumulative distribution of the sum of C_{ij}, $P_n(\beta)$, can be estimated as

$$P_n(\beta) = P\left(\frac{C - \bar{c}}{\sigma} \leq \beta\right) = Q(C \leq \bar{c} + \beta\sigma) \approx \Phi(\beta)$$

where C is the sum of the random variables C_{ij} over i, j, and $\bar{c}$ and σ are the mean and standard deviation of C respectively. P is the cumulative distribution function for the "standardized" random variable $\frac{C-\bar{c}}{\sigma}$ and Q is the cumulative distribution function for C.

$Q(C \leq \bar{c} + \beta\sigma)$ represents the confidence level for the cost to fall below the value $(\bar{c} + \beta\sigma)$. Hence, the value of β can be adjusted to satisfy the confidence level required by the mission, even at low value of n. For a given value of β, decision maker is guaranteed with the same probability that the actual path cost will fall below the value $\bar{c} + \beta\sigma$. Hence, to find the robust shortest path, the objective becomes to minimize this performance index.

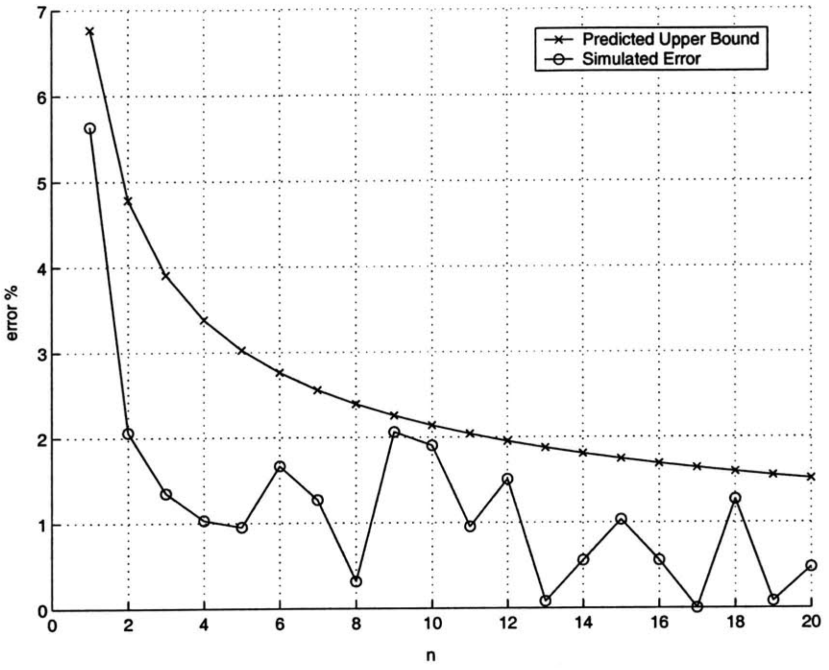
3.3.1 Nonlinearity

Using the performance index in the previous section, the robust shortest path problem is to find the path that minimizes the quantity $\bar{c} + \beta\sigma$, which can be written as

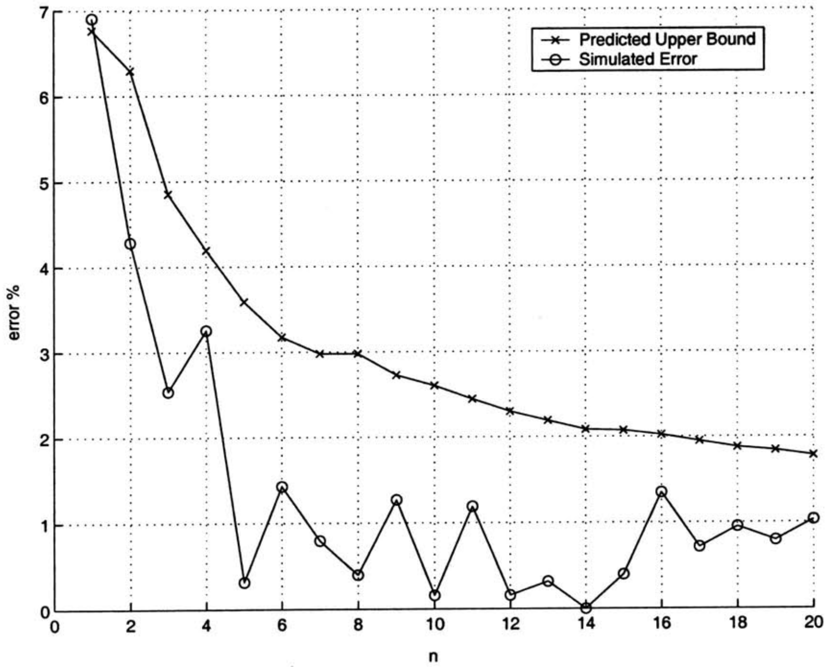
$$\begin{aligned} \min_x J &= \sum_{i,j} \bar{c}_{ij} x_{ij} + \beta \sqrt{\sum_{i,j} \sigma_{ij}^2 x_{ij}} \\ \text{subject to :} \quad & x \in X \end{aligned} \tag{3.6}$$

However, this cost function is nonlinear and non-convex, which makes the problem very difficult to solve. Moreover, the nonlinearity of the square root creates a problem for the ARSP algorithm which is built on the Dijkstra's algorithm.

To illustrate the difficulty with the nonlinearity in the cost, consider a simple scenario shown in Fig. 3-3. There are 3 nodes in this simple scenario. There are two paths connecting nodes 1 and 2, but only a single path between nodes 2 and 3. The cost of the paths are presented as $[\mu_i, \sigma_i]$, which are the mean and standard deviation



(a) Identical Uniform distribution.



(b) Uniform distribution with different variance.

Figure 3-2: Berry-Esséen Inequality.

of the cost. The problem is to find the robust shortest path between nodes 1 and 3. The problem is solved with $\beta = 1$, and it is assumed that $\mu_2 + \sigma_2 < \mu_1 + \sigma_1$.

ARSP algorithm is started from node 1 and the cost function $\sum_i \bar{c}_i + \beta\sqrt{\sum_i \sigma_i^2}$ is propagated towards node 3. At node 2 the algorithm picks path 2 as the robust shortest path between nodes 1 and 2, based on the assumption above. Now consider the values of $\mu_i + \mu_3 + \sqrt{\sigma_i^2 + \sigma_3^2}$, $i = 1, 2$ for a fixed μ_3 but variable σ_3, which are plotted in Fig. 3-3. The plot shows that for $\sigma_3 \geq 0.025$,

$$\mu_1 + \mu_3 + \sqrt{\sigma_1^2 + \sigma_3^2} < \mu_2 + \mu_3 + \sqrt{\sigma_2^2 + \sigma_3^2}$$

which means that the best robust shortest path between nodes 1 and 3 is to take paths 1 and 3. However, given $\mu_2 + \sigma_2 < \mu_1 + \sigma_1$, ARSP will choose paths 2 and 3, which is incorrect. This simple example illustrates a problem that can become quite severe as more paths are added to the scenario.

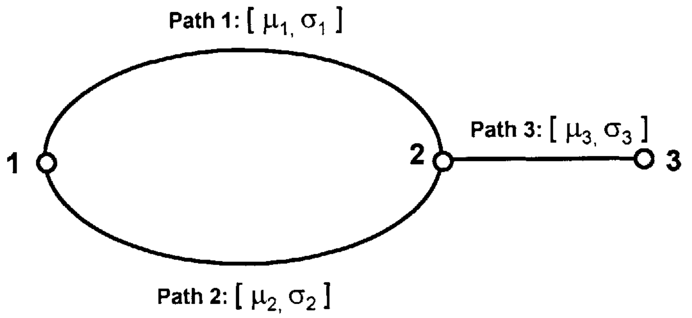

Path 1: [μ_1, σ_1]
Path 3: [μ_3, σ_3]
1
2
3
Path 2: [μ_2, σ_2]

Figure 3-3: Simple shortest path problem.

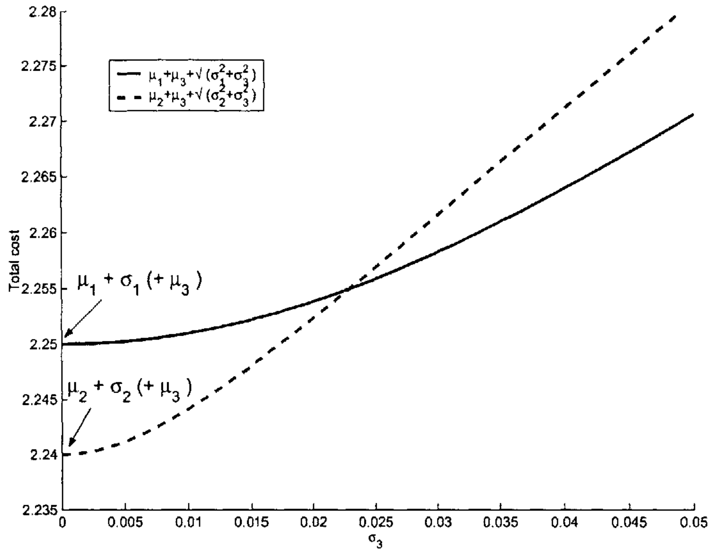


Figure 3-4: Nonlinearity of cost functions.

3.4 Approximation with $\bar{c} + \alpha\sigma^2$

In order to avoid the problem with the nonlinearity in the cost discussed in the previous section, we approximate the cost function in Eq. 3.6 with the *mean-variance* formulation [24]

$$\begin{aligned} \min_x J &= \sum_{i,j} \bar{c}_{ij} x_{ij} + \alpha \sum_{i,j} \sigma_{ij}^2 x_{ij} \\ \text{subject to :} & \quad x \in X \end{aligned} \tag{3.7}$$

The square-root in the original cost can be approximated using a two-point linearization, as shown in Fig. 3-5. This linearization is done for predicted values of σ_L and σ_H, which are the expected lower and upper bound for σ of the solution path respectively. One way to estimate these values is to solve the nominal problem and the following problem

$$\begin{aligned} \min_x J &= \sum_{i,j} \sigma_{ij}^2 x_{ij} \\ \text{subject to :} & \quad x \in X \end{aligned} \tag{3.8}$$

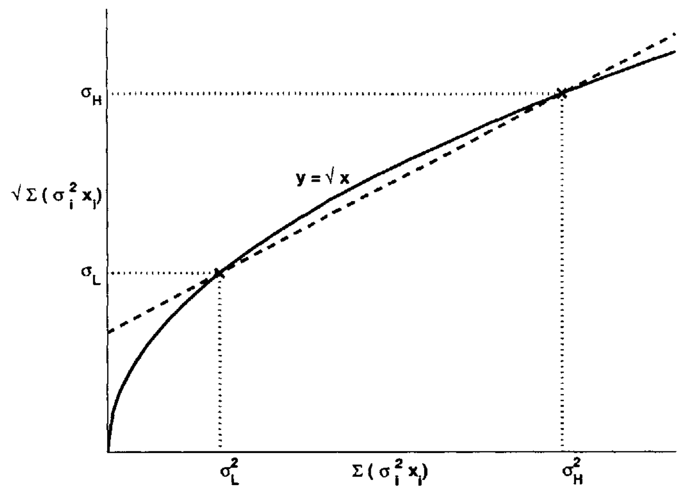


Figure 3-5: Plot of $y = \sqrt{x}$ and linear approximation.

which essentially is to solve the robust problem with a very large value for α. The nominal solution will give the value σ_H and Eq. 3.8 will give σ_L.

Hence, by using the equation of the straight line in Fig. 3-5, the square root term can be approximated as

$$\sqrt{\sum_{i,j} \sigma_{ij}^2 x_{ij}} \approx \left(\frac{\sigma_H - \sigma_L}{\sigma_H^2 - \sigma_L^2}\right)\left(\sum_{i,j} \sigma_{ij}^2 x_{ij}\right) + K \tag{3.9}$$

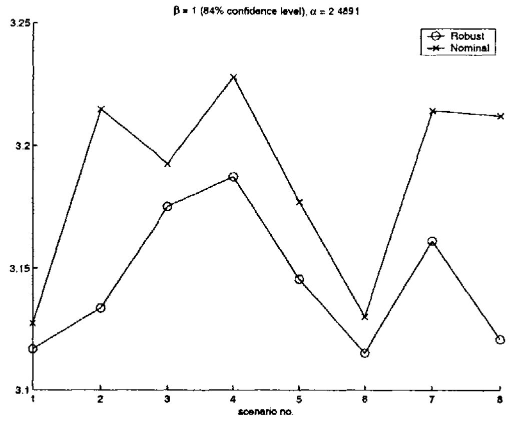
where K is the constant of the linearization. Substituting Eq. 3.9 into Eq. 3.6 gives,

$$\begin{aligned} &\min_x \sum_{i,j} \bar{c}_{ij} x_{ij} + \beta\sqrt{\sum_{i,j} \sigma_{ij}^2 x_{ij}} \\ \approx &\min_x \sum_{i,j} \bar{c}_{ij} x_{ij} + \beta\left(\frac{\sigma_H - \sigma_L}{\sigma_H^2 - \sigma_L^2}\right)\left(\sum_{i,j} \sigma_{ij}^2 x_{ij}\right) + \beta K \end{aligned} \tag{3.10}$$

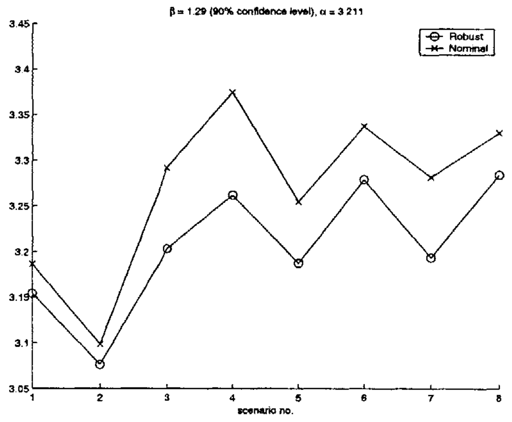
Since βK is a constant, it can be ignored in the minimization. Comparing the coefficient for $\sum_{i,j} \sigma_{ij}^2 x_{ij}$ in Eq. 3.7 and Eq. 3.10, suggests that an appropriate choice of α is

$$\alpha = \beta\left(\frac{\sigma_H - \sigma_L}{\sigma_H^2 - \sigma_L^2}\right) \tag{3.11}$$

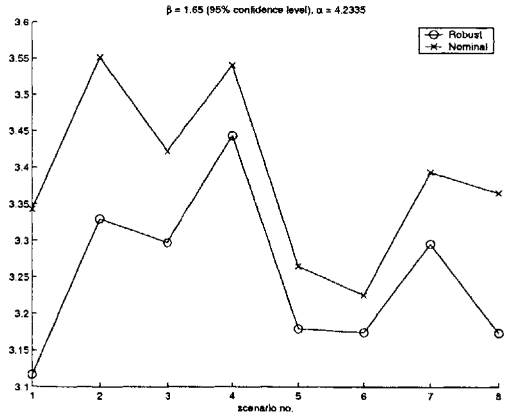
Fig. 3-6 shows the results of several simulations done to verify this approximation.



(a) $\beta = 1$, 84% confidence level.



(b) $\beta = 1.29$, 90% confidence level.



(c) $\beta = 1.65$, 95% confidence level.

Figure 3-6: Plot of $\bar{c} + \beta\sigma$.

The robust shortest path problem was solved for 8 random scenarios using three values of β in the approximation discussed above. Then, the values of the performance index $(\bar{c} + \beta\sigma)$ are compared for the robust and nominal solutions. Fig. 3-6 shows that the robust solutions always have a smaller performance index than the nominal solutions, indicating that the approximation method is a valid method of robustifying the path choice to the uncertainty in the data.

3.5 ARSP as a Modification to Dijkstra's Algorithm

The previous section introduced a cost function that avoided the nonlinearity associated with the performance index $\bar{c} + \beta\sigma$. The new version of the robust shortest path problem is

$$\begin{aligned} \min_x J &= \sum_{i,j} (\bar{c}_{ij} + \alpha\sigma_{ij}^2)x_{ij} \\ \text{subject to :} \quad & x \in X \end{aligned}$$

where α is a positive constant. Note that the cost function is now linear in the decision variables x_{ij}. The parameter α represents the relative weighting between the mean cost $\bar{c}_{ij}$ and the variance σ_{ij}^2, so it can be used as a tuning knob to adjust the robustness of the solution. For example, a decision maker who is completely risk-averse might choose a large value of α, while one who that is not concerned with risk might choose a lower value of α. Choosing $\alpha = 0$ would be equivalent to solve the nominal problem. Since this cost function is linear in the decision variable, x_{ij}, it can also be used in the framework of the Dijkstra's algorithm.

Algorithm Overview

The *Approximate Robust Shortest Path (ARSP) Algorithm* is presented as follow as a modification to the Dijkstra's algorithm. The index of the goal point is set to be 1.

1. Set current node $i = 1$, and initialize:

$$\begin{aligned} \mathcal{P} &= \{1\} \\ D_j &= \begin{cases} 0 & j = 1 \\ \bar{c}_{1j} + \alpha\sigma_{1j}^2 & j \neq 1 \end{cases} \end{aligned}$$

$$M_j = \begin{cases} 0 & j = 1 \\ \bar{c}_{1j} & j \neq 1 \end{cases}$$

$$V_j = \begin{cases} 0 & j = 1 \\ \sigma_{1j}^2 & j \neq 1 \end{cases}$$

   D_j is the "cost" of the shortest path from the goal to node j. M_j and V_j are the *mean* and *variance* of cost of shortest path from the goal to node j, assuming independent arcs.

2. Set node i as the successor of each node j. A successor node is the next node on the path towards the goal.

3. Find the next closest node from the set of unfixed nodes[1], and set it as a new current node i,

$$i := \arg\min_{j \notin \mathcal{P}} D_j$$

4. Fix node i, and update the set $\mathcal{P}$ of fixed nodes,

$$\mathcal{P} \to \mathcal{P} \cup \{i\}$$

5. If all nodes in $\mathcal{N}$ are also in the set $\mathcal{P}$, i.e. all nodes are fixed, terminate.

6. For all the nodes that are unfixed and visible from the current node i, i.e., for each $\{j | j \notin \mathcal{P}, \bar{c}_{ij} + \alpha\sigma_{ij}^2 \neq \infty\}$

[1]Unfixed nodes are those whose shortest path to the goal has not been found.

(a) Update the temporary labels,

$$D_j = \min(D_j, D_j + \bar{c}_{ij} + \alpha\sigma_{ij}^2)$$

(b) If D_j is updated with $D_j + \bar{c}_{ij} + \alpha\sigma_{ij}^2$, set i as the successor of node j. If D_j is updated, update M_j and V_j accordingly,

$$\begin{aligned} M_j &\rightarrow M_j + \bar{c}_{ij} \\ V_j &\rightarrow V_j + \sigma_{ij}^2 \end{aligned}$$

(c) Pick the next j and go to Step 6a.

7. Go to Step 3.

The algorithm produces a tree of nodes with the goal node as the root. D_j gives the minimum cost from j to the goal. Note that by assuming independency of the arcs, the calculation of M_j and V_j is exact, so they give accurate estimates of the mean cost and variance of the shortest path from the goal to node j.

3.6 Complexity of Robust Dijkstra's Algorithm

The advantage of ARSP is its simplicity. The Dijkstra's algorithm itself is known to be an efficient algorithm for solving one-to-all-nodes shortest path problem, with a complexity of $O(|\mathcal{N}|^2)$ [23]. The ARSP essentially imposes *no* additional complexity to Dijkstra's algorithm, with robustness taken into account. The only difference is that ARSP propagates a different cost function. To demonstrate this strength of ARSP, its performance is compared with two other robust optimization formulations, the Robust Deviation Shortest Path (RDSP) formulation [11] and the robust integer programming algorithm introduced by *Bertsimas et al.* [10].

3.6.1 Robust Deviation Shortest Path (RDSP)

The RDSP in Ref. [11] implements the robust deviation criterion. The robust deviation of a path p is defined as the difference between the cost of the path p and the cost of the shortest path for a specific realization of arc cost. Hence, a path p is defined as the robust deviation shortest path if it has the least maximum robust deviation among all paths. It is in general an NP-hard problem [12]. However, given the uncertainty set of the cost being interval data, the solution set can be confined to a finite number of realizations because the robust deviation for a path p is only maximized for a realization in which the cost of all arcs on p is set at the upper bounds and the cost of all others arcs are set at the lower bounds [11]. The problem is formulated as a mixed integer program as follows [11],

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in\mathcal{A}} c_{u,ij} y_{ij} - x_n \\
\text{subject to :} \quad & \sum_{\{j:(i,j)\in\mathcal{A}\}} y_{ij} - \sum_{\{j:(j,i)\in\mathcal{A}\}} y_{ji} = \begin{cases} 1 & i = 1 \\ -1 & i = n \\ 0 & \text{otherwise} \end{cases} \\
& x_j \le x_i + c_{l,ij} + (c_{u,ij} - c_{l,ij}) y_{ij} \quad \forall (i,j) \in \mathcal{A} \\
& y_{ij} \in \{0,1\} \quad \forall (i,j) \in \mathcal{A} \\
& x_1 = 0 \\
& x_i \ge 0 \qquad i = 1, 2, \cdots, n
\end{aligned} \tag{3.12}$$

where x_i is the minimum cost to go from the starting node (node 1) to node i in scenario defined by y_{ij}. Hence, x_n is the path with minimum cost to go from the starting point to the goal (node n) in scenario defined by y_{ij}. arc(i,j) indicates an arc going from node i to node j.

A preprocessing algorithm has been developed in reference [11] to reduce the solution set in order to speed up the optimization process. The algorithm is generally applicable to acyclic directed graphs. However, for the path planning problem we are dealing with, the visibility graph is undirected and highly connected, the preprocessing algorithm will be of little use, so it is not used here.

3.6.2 Bertsimas-Sim Algorithm

In [10], Bertsimas *et al.* proposed an approach to address data uncertainty for discrete optimization and network flow problems which is computationally tractable and it allows the decision maker to control the degree of conservatism of the solution. It is similar to ARSP in the way that it applies the absolute robust criterion and has a tuning knob to adjust the level of the robustness of the solutions. When only cost coefficients are subject to uncertainty and the problem is a $0-1$ discrete optimization problem on n variables, they show that the robust problem can be solved by solving $n+1$ nominal problems.

In the formulation, they argue that nature will be restricted in its behavior in such a way that only a subset of the costs will change in order to adversely affect the solution. If the problem behaves in this way, it can be guaranteed that the robust solution will be feasible deterministically. Moreover, even if more changes occur than expected, the robust solution will be still feasible with very high probability. For the shortest path problem, the deterministic problem is posed as

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} \\
\text{subject to :} \quad & \sum_{\{j:(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j:(j,i)\in\mathcal{A}\}} x_{ji} = \begin{cases} 1 & i=1 \\ -1 & i=n \\ 0 & \text{otherwise} \end{cases} \\
& x_{ij}\in\{0,1\} \quad \forall (i,j)\in\mathcal{A}
\end{aligned} \tag{3.13}$$

The robust counterpart is

$$\begin{aligned} Z^* = \quad & \min \sum_{(i,j)\in\mathcal{A}} c_{l,ij}x_{ij} + \max_{\{S|S\subseteq\mathcal{A},|S|=\Gamma\}} \sum_{(i,j)\in S} d_{ij}x_{ij} \\ \text{subject to :} \quad & \sum_{\{j:(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j:(j,i)\in\mathcal{A}\}} x_{ji} = \begin{cases} 1 & i=1 \\ -1 & i=n \\ 0 & \text{otherwise} \end{cases} \\ & x_{ij} \in \{0,1\} \quad \forall (i,j)\in\mathcal{A} \end{aligned} \tag{3.14}$$

where $d_{ij} = c_{u,ij} - c_{l,ij}$ and Γ is the number of c_{ij} that are allowed to change. Γ serves as a tuning knob to adjust the level of robustness of the solution. The robust counterpart is NP-hard. However, by using the algorithm in [10], the robust problem can be solved in polynomial time.

Without loss of generality, d_{ij} can be arranged in a descending order. They are re-indexed as $d_1 \geq d_2 \geq \cdots \geq d_n$ and we define $d_{n+1} = 0$. The algorithm is posed as

Bertsima-Sim Algorithm [10]

1. For $m = 1, \ldots, n+1$,

$$G^l = \Gamma d_m + \min_{x\in X}\left(\sum_{(i,j)\in\mathcal{A}} c_{l,ij}x_{ij} + \sum_{\{(i,j)|d_{ij}\geq d_m\}} (d_{ij}-d_m)x_{ij}\right)$$

   and let x^m be the optimal solution to the above optimization problem.

2. Let $m^* = \arg\min_{m=1,\ldots,n+1} G^m$.

3. $Z^* = G^{m^*}; x^* = x^{m^*}$.

Using Dijkstra's algorithm, the shortest path problem can be solved in $O(|\mathcal{N}|^2)$, while Bertsimas-Sim Algorithm will solve it in $O(|\mathcal{A}||\mathcal{N}|^2)^2$ [10].

$^2|\mathcal{A}|$ actually refers to number of distinct d_i

Table 3.1: Summary of problem sizes. $|\mathcal{N}|$ is the number of nodes and $E(|\mathcal{A}|)$ is the average number of arcs.

$\lvert\mathcal{N}\rvert$	50	80	110	140	170	200
$\mathbf{E}(\lvert\mathcal{A}\rvert)$	262	466	625	746	1162	1609

3.6.3 Comparison with RDSP and Bertsimas-Sim Algorithm

ARSP is compared with the RDSP and Bertsimas-Sim algorithms to test the performance. The two algorithms are written in ILOG CPLEX using Concert Technology [22]. Scenarios of different size are randomly generated within a square from $(0,0)$ to $(1,1)$. The "size" refers to the number of nodes in the scenario, and a "scenario" refers to a particular randomly generated graph. For each problem size, 8 different scenarios are created and 2,000 random realizations of arc cost are generated for each scenario to compute the cost of the optimal path solved. The costs of the arcs are uniformly distributed within the interval $[c_{l,ij}, \gamma_{ij}c_{l,ij}]$. A square region of high uncertainty is located from $(0.4, 0.5)$ to $(0.8, 0.9)$. For any arcs crossing this region, the values for γ_{ij} are uniformly distributed in [0, 10], and for the other arcs, in [0, 3]. Hence, some of the arcs have much larger cost deviation than others. The problems are solved on a Dell Pentium4 (2.40GHz and 1GB RAM). Comparisons are made among ARSP (for $\alpha = 3$), RDSP, Bertsimas-Sim Algorithm (for $\Gamma = 8$), as well as Dijkstra's with nominal cost (a stochastic problem). The simulation results are summarized in Table 3.1. (Note: the values for α and Γ are chosen such that both algorithms indicate a relatively high level of robustness for the solutions [10].)

The results of the simulations for 110 nodes and 200 nodes are shown in Fig. 3-7 and Fig. 3-8. (Results for other problem sizes are shown at the end of the chapter.) The mean and the standard deviation of path cost are plotted for the 4 algorithms. The percentage difference from the mean and standard deviation of path cost of the nominal solution are plotted as well. In particular, the result for the case of 200 nodes is tabulated in Table 3.2 to show some numerical values. The percentage is calculated as $\frac{Robust-Nominal}{Nominal} \times 100\%$.

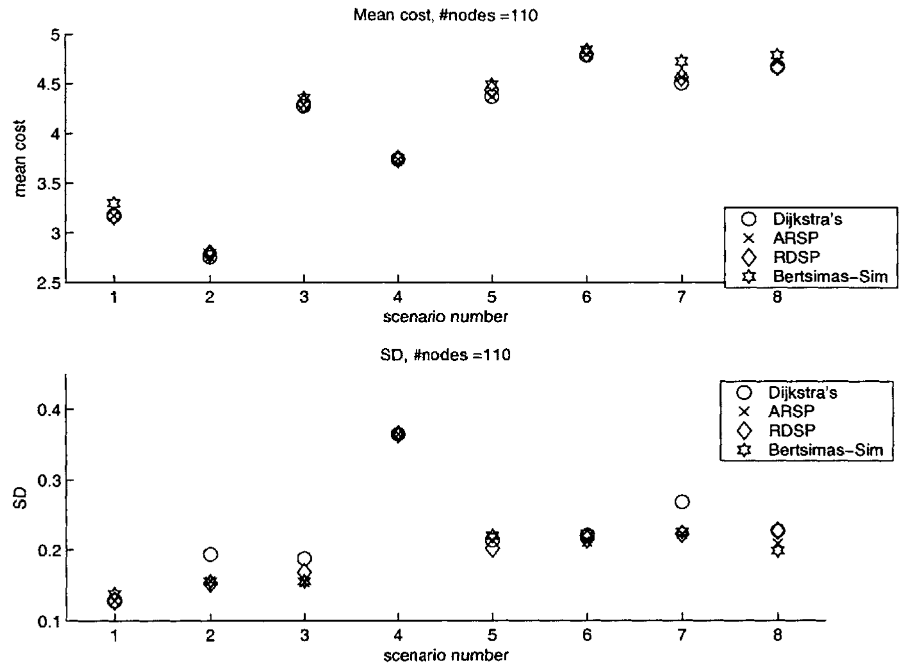
Table 3.2: Mean cost and SD for #nodes = 200.

Nominal		**ARSP**				**RDSP**				**Bertsimas-Sim**			
$\bar{c}$	σ	$\bar{c}$	$\%_{\bar{c}}$	σ	$\%_{\sigma}$	$\bar{c}$	$\%_{\bar{c}}$	σ	$\%_{\sigma}$	$\bar{c}$	$\%_{\bar{c}}$	σ	$\%_{\sigma}$
3.53	0.12	3.55	0.41	0.086	-27.44	3.55	0.41	0.086	-27.44	3.67	3.84	0.097	-18.26
3.93	0.14	3.95	0.65	0.13	-8.47	3.96	0.74	0.14	-1.20	4.01	2.10	0.13	-5.34
4.10	0.14	4.12	0.50	0.12	-13.35	4.14	0.78	0.12	-14.11	4.17	1.52	0.14	0.37
3.80	0.10	3.80	0.10	0.10	-1.96	3.81	0.34	0.10	-2.14	3.88	2.35	0.10	-0.57
4.01	0.24	4.01	0	0.24	0	4.08	1.84	0.24	-2.45	4.22	5.27	0.23	-4.91
4.10	0.11	4.12	0.47	0.10	-11.41	4.14	0.92	0.093	-17.09	4.25	3.62	0.096	-13.80
4.39	0.17	4.40	0.18	0.16	-8.72	4.42	0.53	0.15	-9.72	4.66	6.09	0.17	-0.42
3.55	0.08	3.55	0	0.08	0	3.56	0.31	0.08	-0.55	3.65	2.68	0.079	-1.33

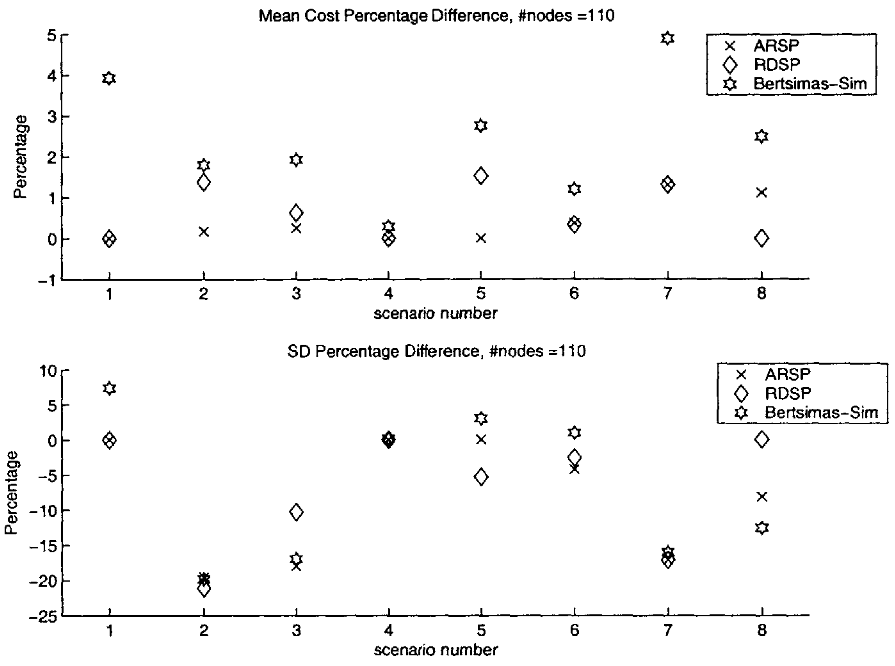
The results show that the robustness of the solution (reflected by a reduction in the standard deviation) is typically associated with an increase in the mean. From the figures and table, it can be seen that ARSP (average 8.92%)[3] has reductions in the standard deviation that are comparable to RDSP (average 9.34%) and the Bertsimas-Sim Algorithm (average 5.5%). Moreover, ARSP (average 0.29%) has a smaller increase in the mean value than RDSP (average 0.73%) and the Bertsimas-Sim Algorithm (average 3.43%). The result clearly illustrate that the ARSP performance is at least **comparable** to the RDSP and Bertsimas-Sim Algorithms.

So while the performance is comparable to the other algorithms, the primary advantage of ARSP is its simplicity. Fig. 3-9 shows the average planning time for the 4 algorithms as a function of problem size. The figure indicates that while incorporating robustness criterion into the formulation, the ARSP does not require any significant increase in computation load over Dijkstra's algorithm. On the other hand, RDSP and the Bertsimas-Sim Algorithm are about 10 times and 1000 times, respectively, slower than ARSP for problem size of 50 nodes. Furthermore, their computation time increases very rapidly with problem size, so that for a problem with 200 nodes, they take about 100 times and 10,000 times, respectively, more time to solve than ARSP. By curve fitting to the simulation results, the complexity of the Dijkstra's, ARSP, RDPS and Bertsimas-Sim algorithms are estimated to be $O(|\mathcal{N}|^2)$, $O(|\mathcal{N}|^2)$, $O(|\mathcal{N}|^5)$ and $O(|\mathcal{N}|^6)$, respectively.

[3](Note: the numbers in bracket refers to the numbers in Table 3.2).

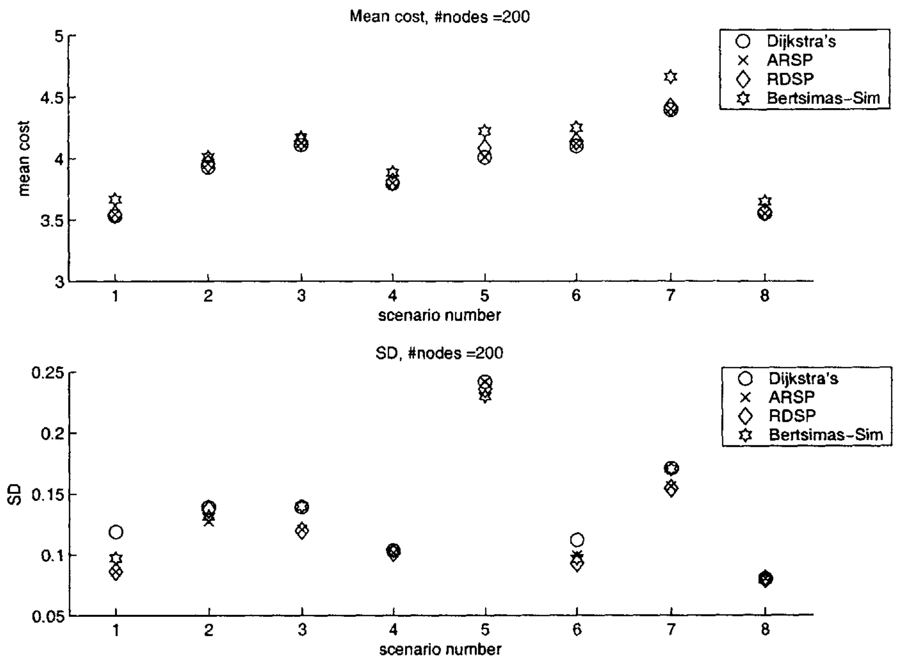


(a) Mean and standard deviation of cost.

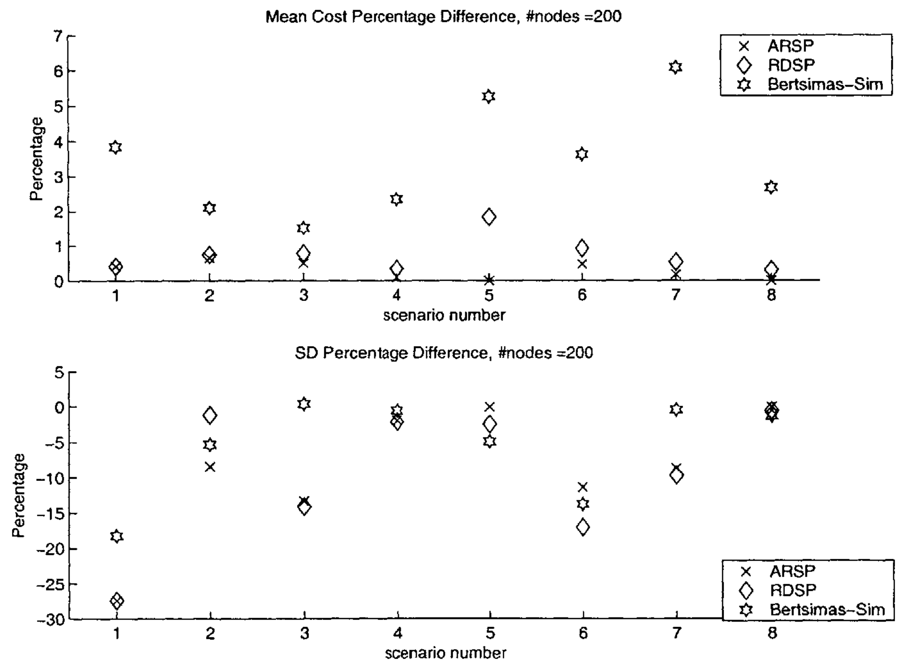


(b) Mean and standard deviation of cost difference from nominal Dijkstra's.

Figure 3-7: Cost distribution for scenario with 110 nodes (4 Algorithms).



(a) Mean and standard deviation of cost.



(b) Mean and standard deviation of cost difference from nominal Dijkstra's.

Figure 3-8: Cost distribution for scenario with 200 nodes (4 Algorithms).

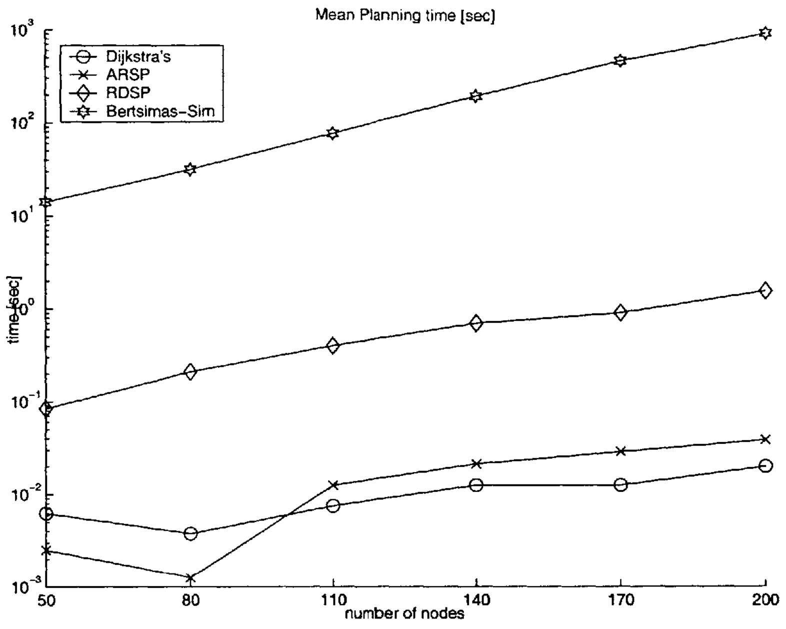
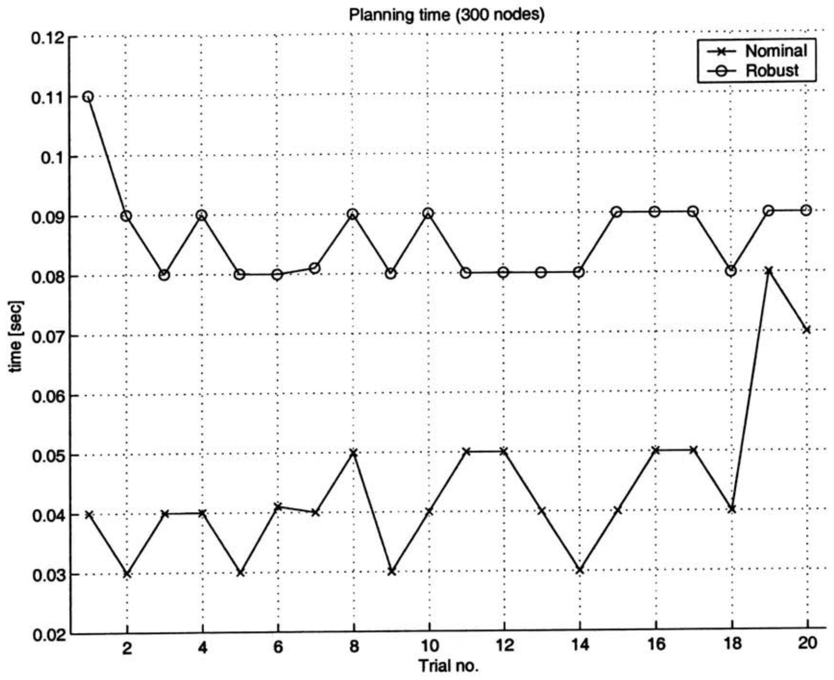


Figure 3-9: Mean planning time of 4 algorithms

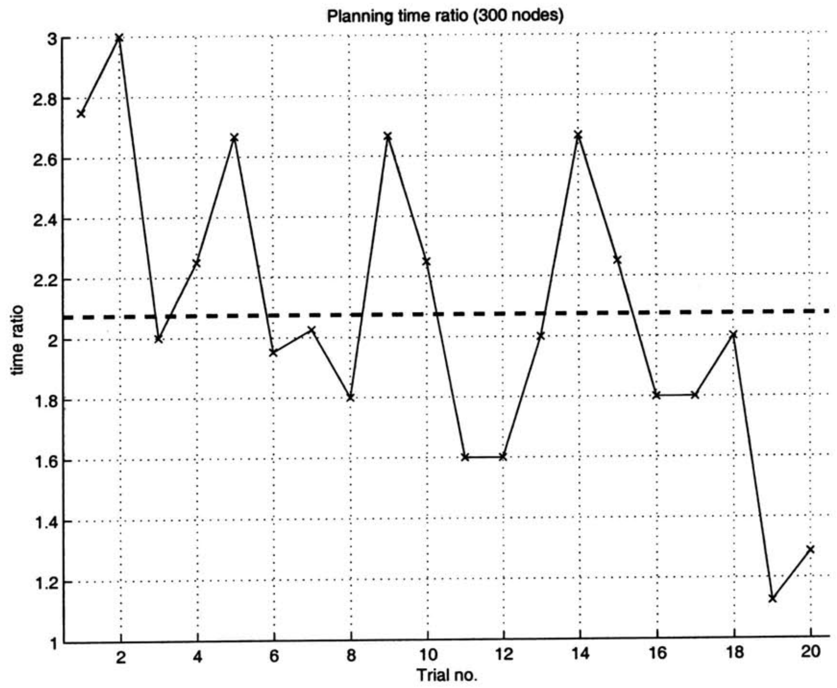
Fig. 3-10(a) shows the planning time for the ARSP and Dijkstra's (on the nominal problem) algorithms for a problem size of 300 nodes. And Fig. 3-10(b) shows the ratio of planning time of ARSP to nominal Dijkstra's. The dotted line indicates the average ratio. By adding the robustness criterion, ARSP is only about 2.1 times slower than the nominal Dijkstra's at a problem size as large as 300 nodes. For, the UAV problem we are dealing with, the problem size seldom reach this level. Hence, we can apply the robust algorithm with a trivial increase in the computation time.

3.7 Examples

An example adopted from the Ref. [10] is presented here to show the validity of the ARSP algorithm. A randomly generated undirected graph with 250 nodes is constructed as shown in Fig. 3-11(a). The starting node is at the origin $(0, 0)$ and the destination node is at $(1, 1)$. The mean cost, $\bar{c}_{ij}$ indicates the average time of travel on arc(i, j). All $\bar{c}_{ij}$ are kept within the range 0.19 ± 0.0285. Uncertainty can be introduced into the problem as variations in speed of travel along the arcs as a result

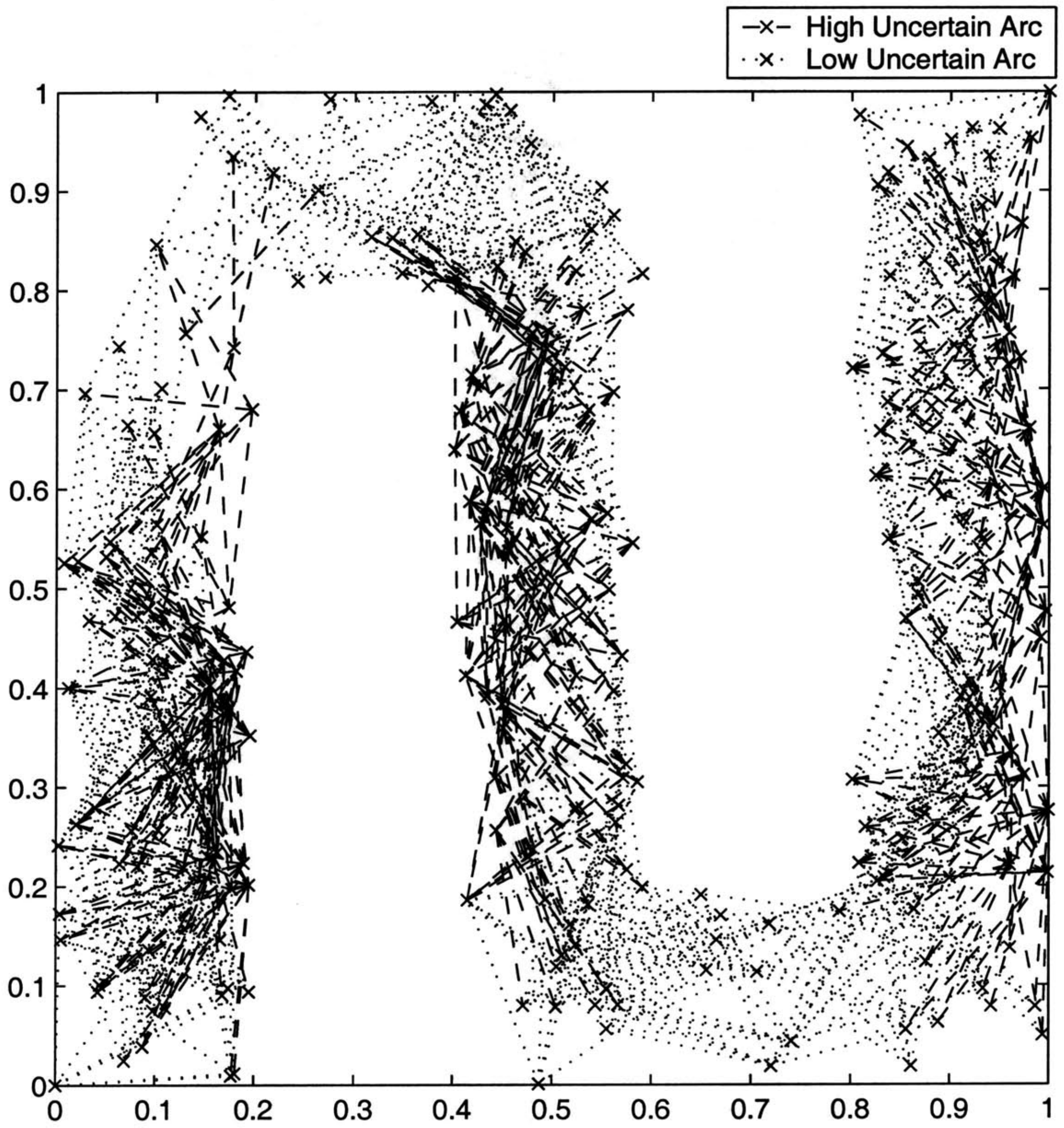


(a) Planning Time.

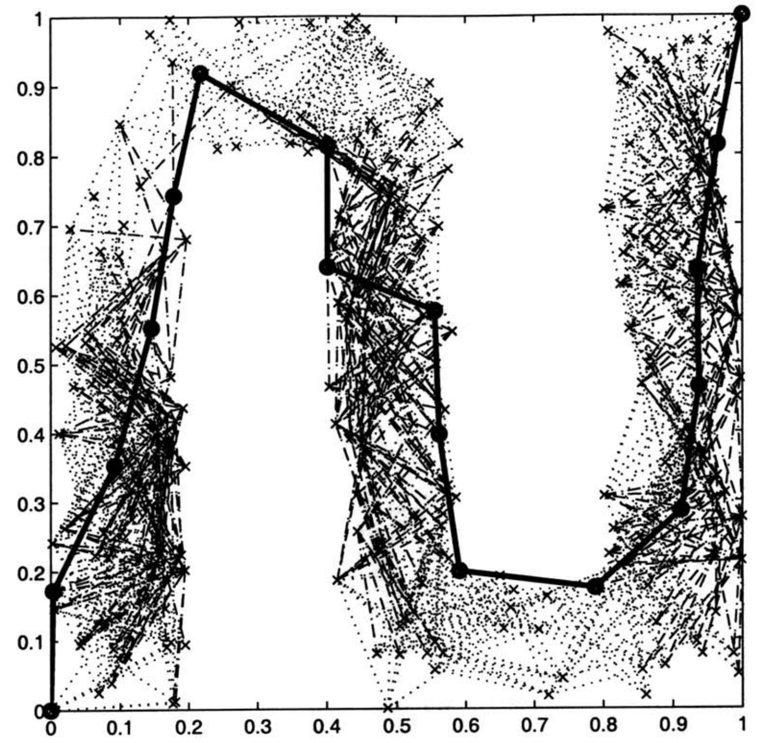


(b) Planning Time Ratio.

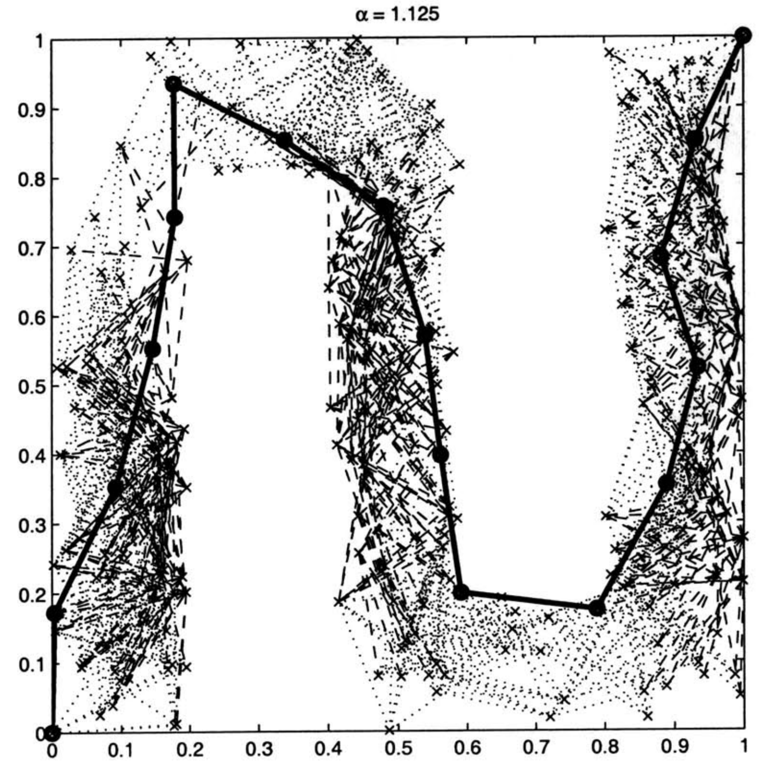
Figure 3-10: Planning Time Comparison for ARSP and Dijkstra's Algorithms.


High Uncertain Arc
Low Uncertain Arc
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

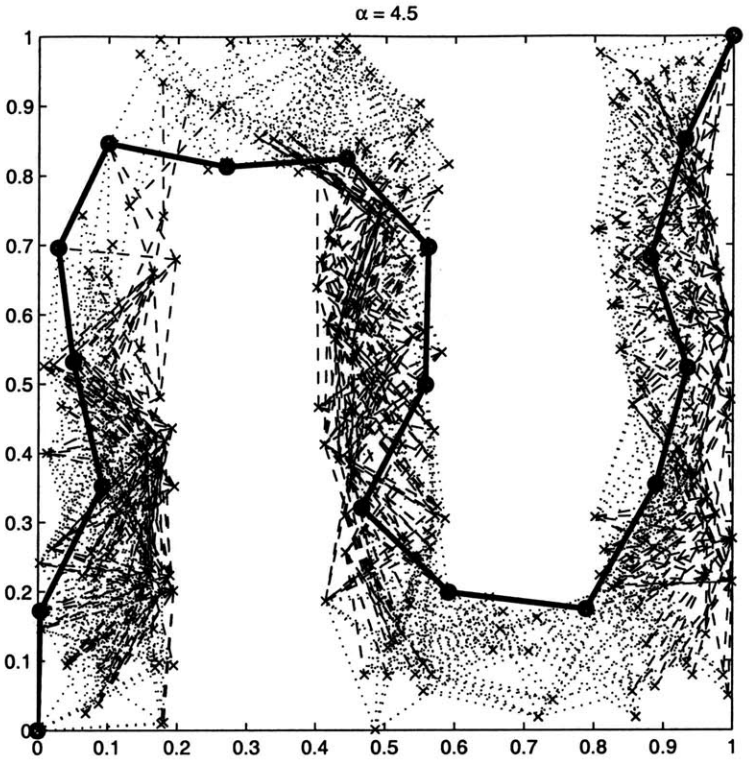
(a) Scenario.



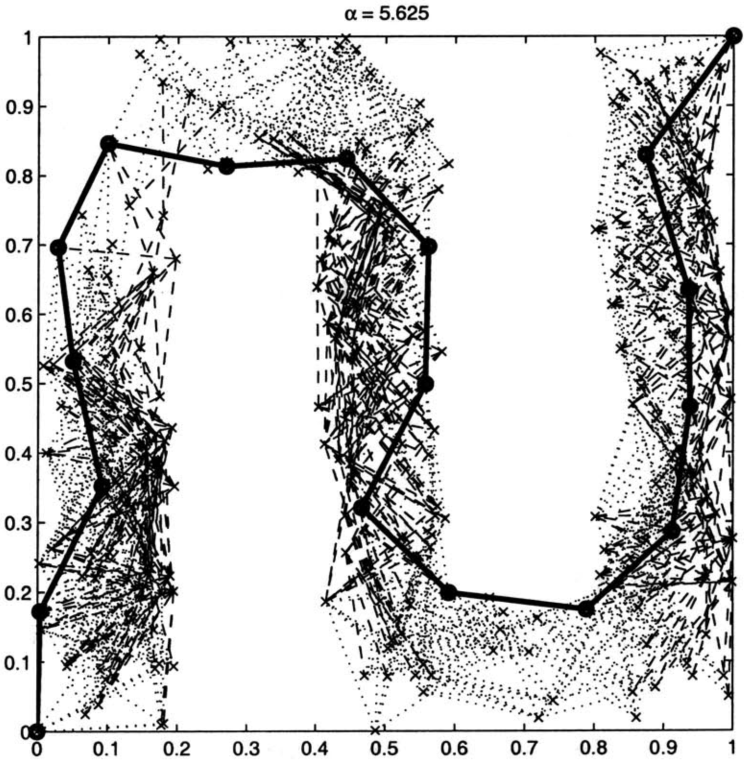
(b) Nominal Solution.



(c) Robust Solution, $\alpha = 1.125$.



(d) Robust Solution, $\alpha = 4.5$.



(e) Robust Solution, $\alpha = 5.625$.

Figure 3-11: Random Graph Example.

of different terrain, hence, the time of travel is uncertain. The standard deviations σ_{ij} are divided into two categories. The arcs in dotted lines (light) are assigned with the values $\sigma + \triangle\sigma$ while the arcs in dashed lines (light) are assigned with the values $10(\sigma + \triangle\sigma)$, where $\sigma = 0.0089$, and $\triangle\sigma$ is a positive random number smaller than 0.05σ. Hence, the arcs in dashed line (light) are as much as 10 times more "uncertain" than the dotted line (light). The problem is solved using different values of α in the range of $[0, 50]$. The solutions are shown in Fig. 3-11(c)–Fig. 3-11(e). The nominal solution is shown in Fig. 3-11(b). The nominal solution takes the path with minimum average time of travel, without considering the embedded uncertainty along the path. However, it can be seen that the robust solutions try to avoid some of the regions of high uncertainty. When α is small, the robust path is similar to the nominal path, except that the robust path takes a few detour to avoid the region of high uncertainty. As α becomes larger, the robust path deviates more from the nominal path and tries to avoid more high uncertainty regions, hence the nominal cost is increased as a result of taking detour, but the variance in cost is reduced by avoiding more high uncertainty regions.

We assume that the arc cost is subject to uniform distribution with mean $\bar{c}_{ij}$ and variance σ_{ij}^2. 2000 realizations are generated. The mean and variance (or standard deviation) in cost of the solutions are calculated. The results are shown in Fig. 3-12 and the cost distribution is shown in Fig. 3-13. The mean and standard deviation in cost and the corresponding percentage difference from the nominal solution are summarized in Table 3.3.

Fig. 3-12 shows the mean and variance of all the solutions over the range of α considered. The figure shows that there are 4 distinct solutions (including the nominal case) over this range of α, and it is clear that the variance is negatively correlated to the mean cost. Fig. 3-13 shows the distribution of cost from the realizations. It is shown that for $\alpha = 5.625$, the standard deviation is reduced by 83.31% with only 4.74% increase in mean. Although the robust solution has a higher expected cost, there is more guarantee for this value with the lower variance. Moreover, it is shown in the histogram that the robust formulation has protected the solution against the "worst-case" situation, i.e. the upper bound of cost.

Table 3.3: Summary of results of robust shortest path problem.

α	$\bar{c}$	$\%_{\bar{c}}$	σ	$\%_{\sigma}$
0	2.9735	0	0.2297	0
1.125	2.9756	0.07	0.1905	17.06
4.5	3.0680	3.18	0.1009	56.06
5.625	3.1144	4.74	0.0383	83.31

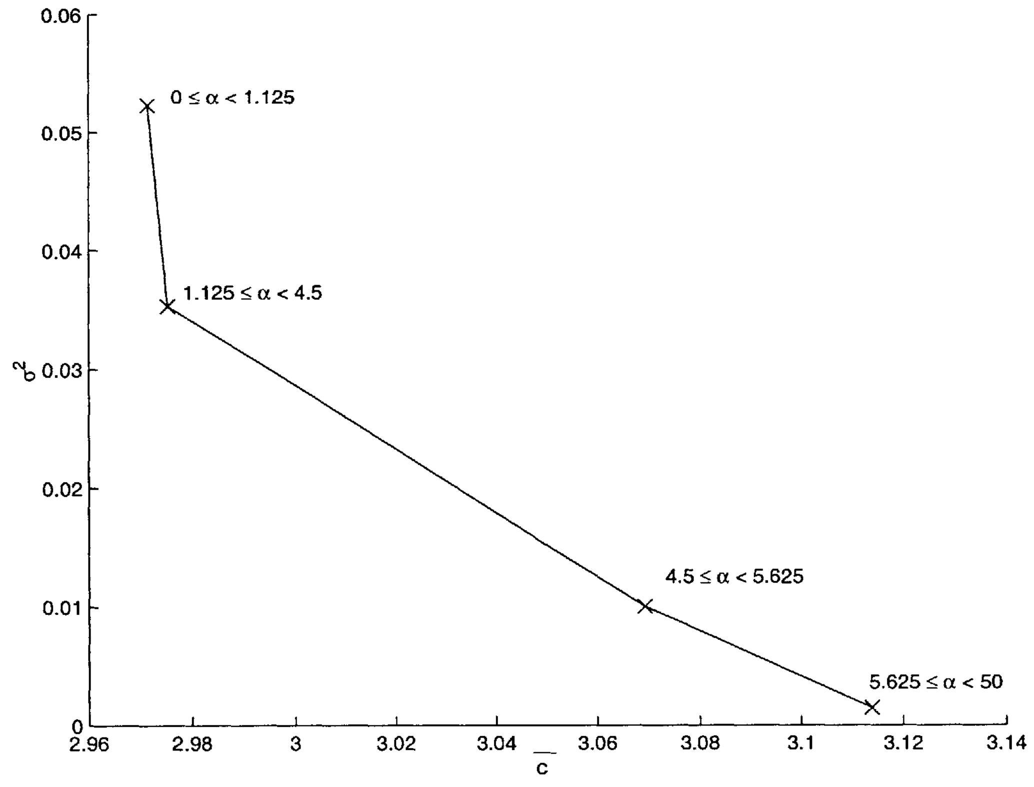


Figure 3-12: Plot of $\bar{c}$ vs. σ^2.

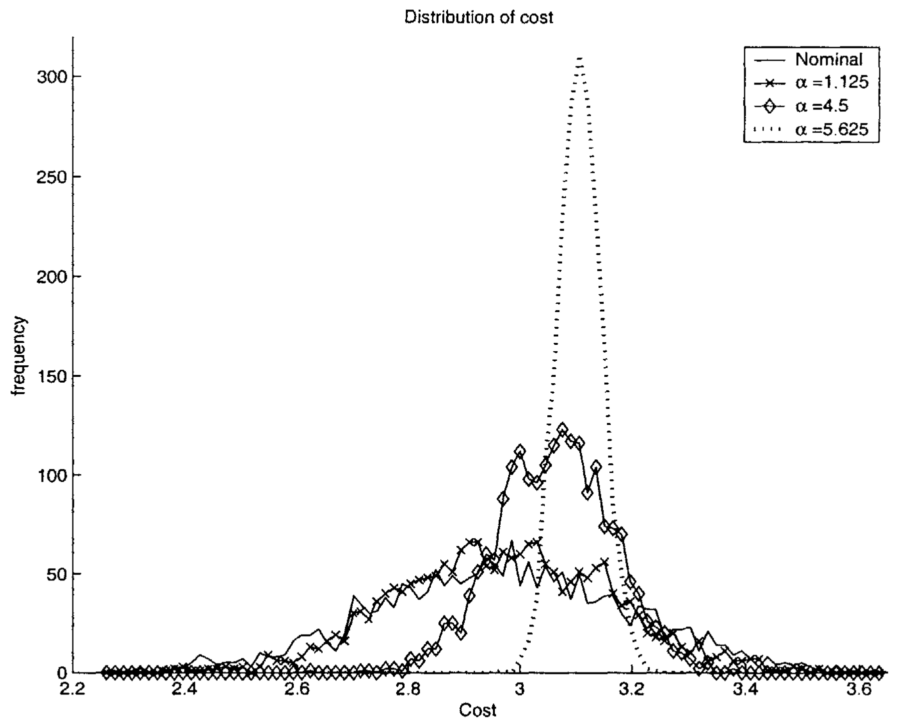
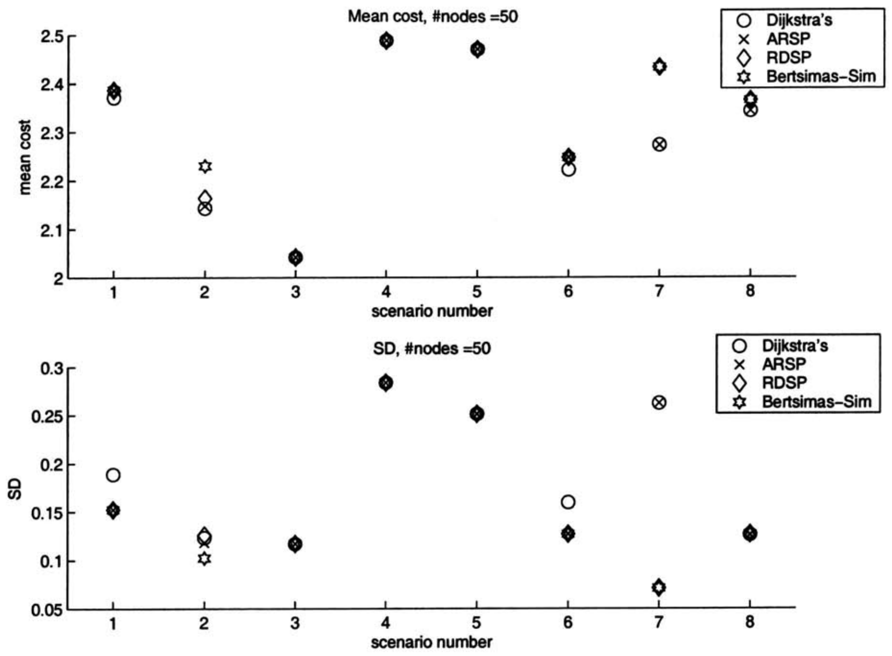


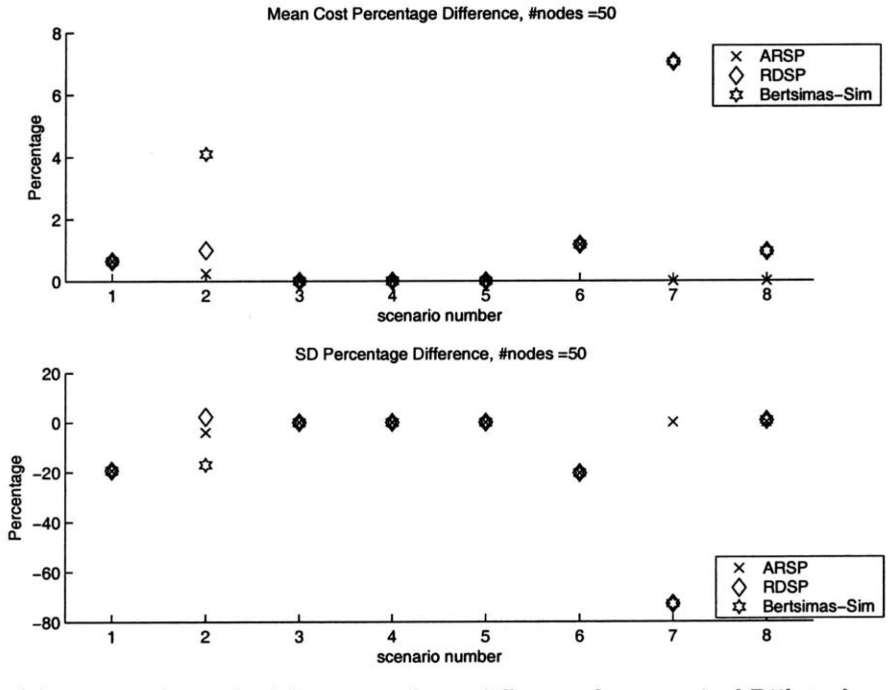
Figure 3-13: Cost Distribution of Random Graph example.

3.8 Conclusions

This chapter has discussed the formulation of the ARSP algorithm to solve the shortest path problem with uncertainty in arc cost. ARSP has been shown to be an efficient and flexible tool to solve the problem. It is compared with the RDSP and Bertsimas-Sim algorithm. Simulation results have shown that ARSP algorithm has comparable performance with the other two robust algorithms , but with trivial computation cost over the nominal problem. A systematic way to choose the robustness parameter α is also discussed.

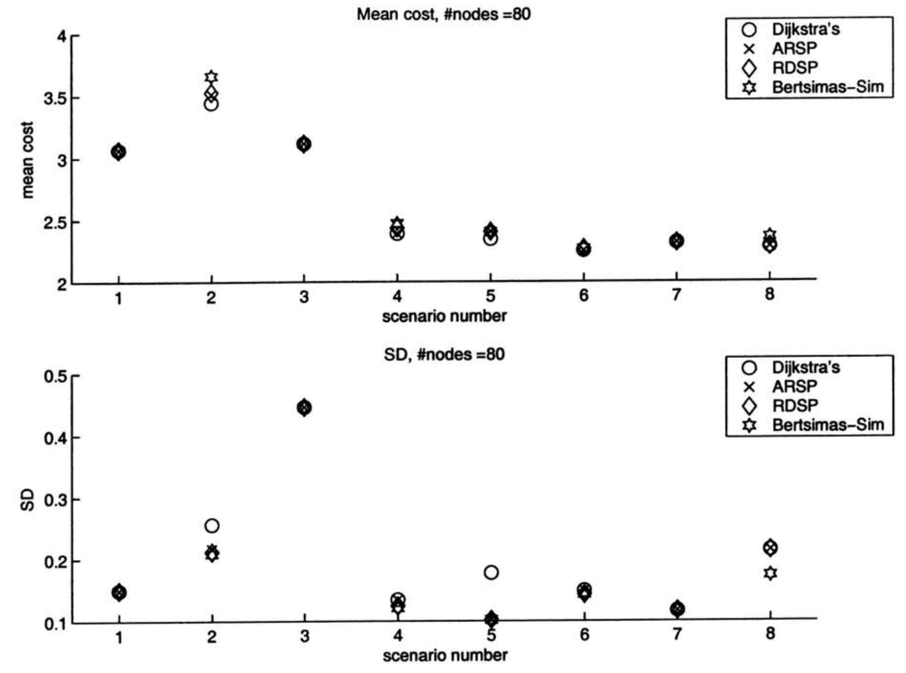


(a) Mean and standard deviation of cost.

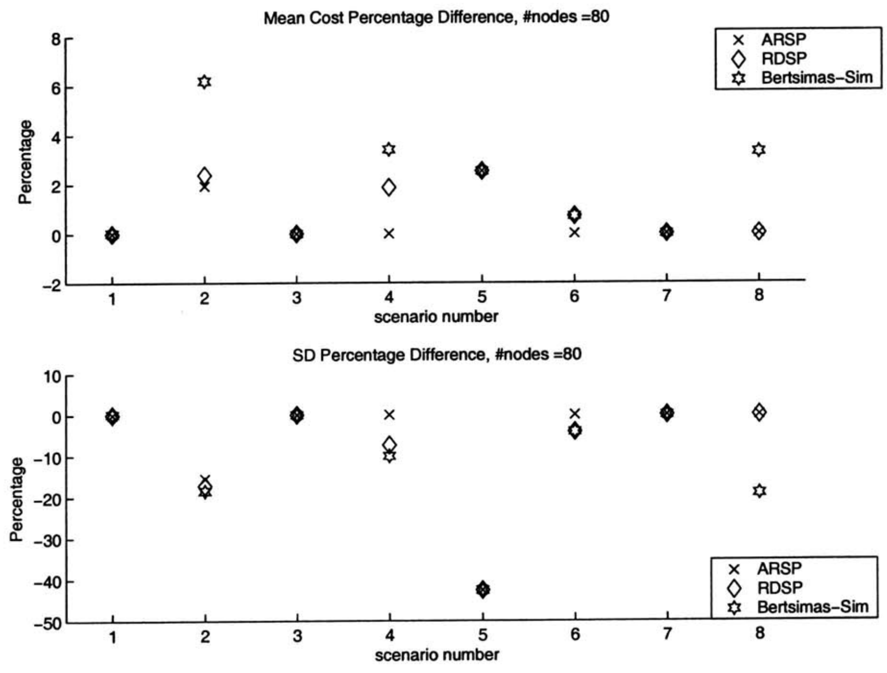


(b) Mean and standard deviation of cost difference from nominal Dijkstra's.

Figure 3-14: Cost distribution for scenario with 50 nodes (4 Algorithms).

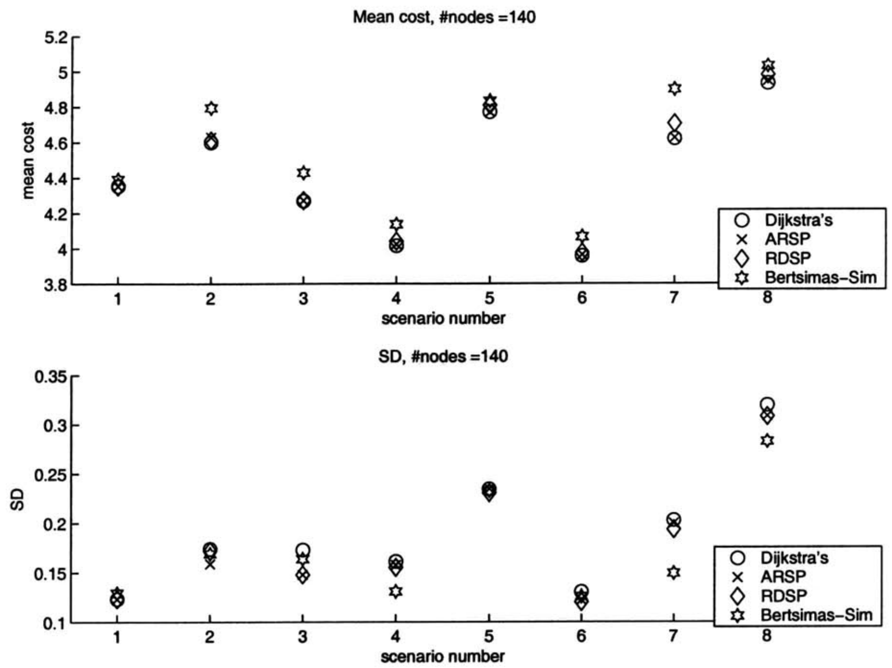


(a) Mean and standard deviation of cost.

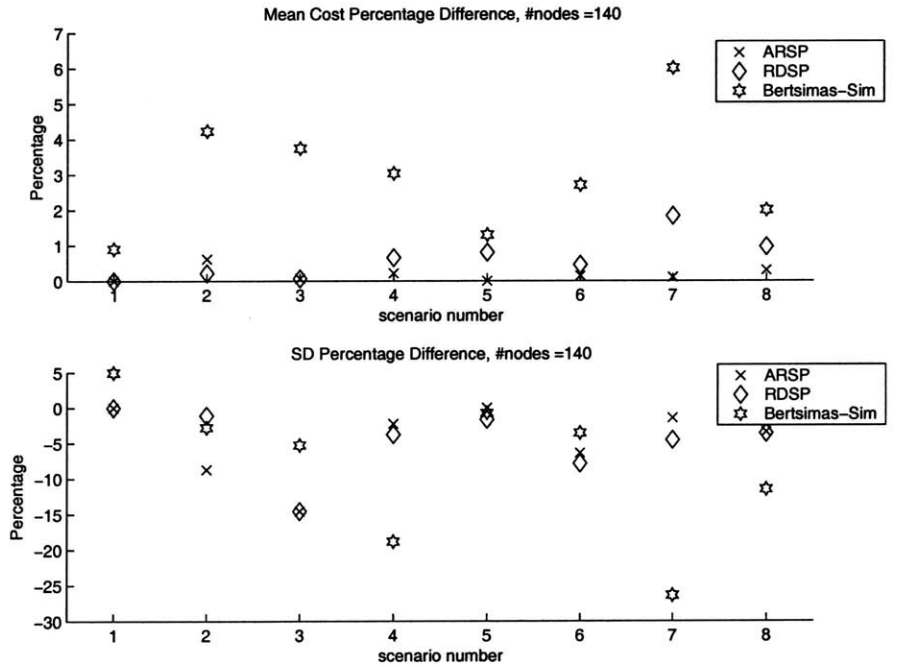


(b) Mean and standard deviation of cost difference from nominal Dijkstra's.

Figure 3-15: Cost distribution for scenario with 80 nodes (4 Algorithms).

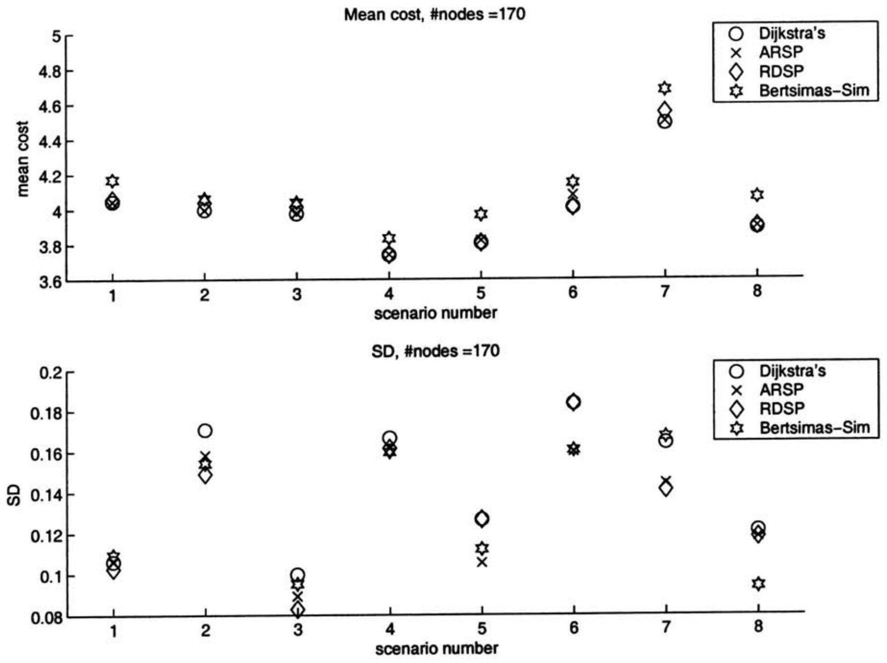


(a) Mean and standard deviation of cost.

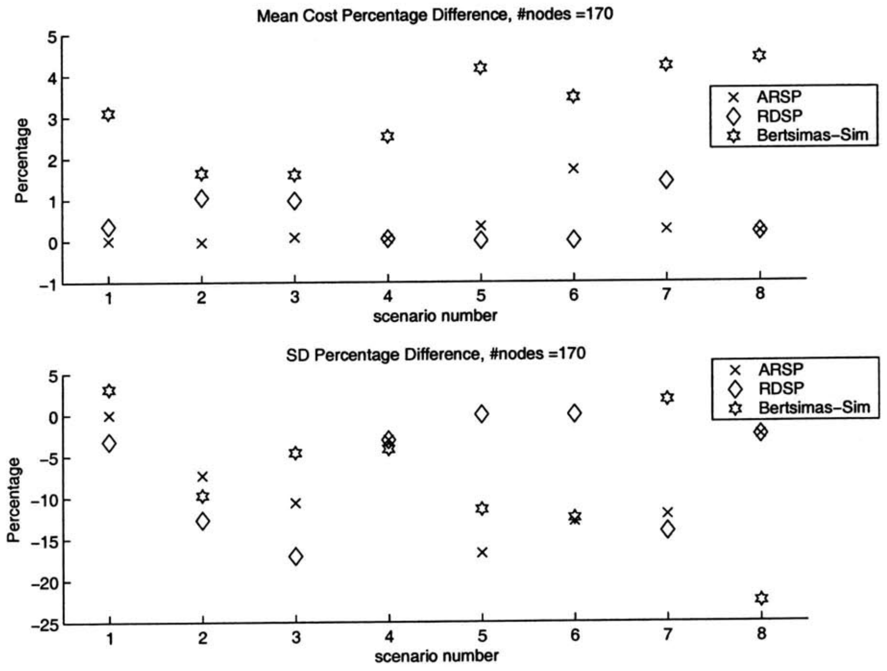


(b) Mean and standard deviation of cost difference from nominal Dijkstra's.

Figure 3-16: Cost distribution for scenario with 140 nodes (4 Algorithms).



(a) Mean and standard deviation of cost.



(b) Mean and standard deviation of cost difference from nominal Dijkstra's.

Figure 3-17: Cost distribution for scenario with 170 nodes (4 Algorithms).

Chapter 4

Search and Track Mission

This chapter describes a decision algorithm for cooperative search among a team of UAVs in the presence of unknown static and moving targets. The search mission occurs in a number of military and civilian applications, for instance, search-and-rescue in open sea, and search for previously spotted enemy targets. We consider a team of UAVs, each of which is equipped with a sensor that provides information about the environment that is used to reduce the uncertainty by confirming the presence (and determining the location) of the targets. The UAVs must cooperate to obtain an efficient search strategy. Furthermore, since many of the targets move, the UAVs must to decide whether to keep tracking the target or to search new areas.

There has been extensive work developing algorithms for cooperative search mission [32, 33, 34, 35]. This chapter extends that work to handle missions involving multiple moving targets. Section 4.1 describes the simplified path planning algorithm used for this search-and-track algorithm. Section 4.3 presents a searching algorithm in the presence of only static targets. The two sections adopted the work presented in Refs. [32, 33]. Section 4.4 discusses the extensions of that algorithm to handle moving targets.

4.1 Path Planning Algorithm

The search and track mission for the UAVs is essentially collapses to a path planning problem. The UAV chooses the path to travel in order to optimize the *gain of information* from the search or track. Similar to the RH-MILP problem [13], the UAVs are assumed to move with constant speed and altitude. Moreover, the UAVs are constrained with a certain maneuverability limit, which is captured by the minimum turning radius (or maximum turning angle, $\theta_{\max}$). The path planning is done in discrete time in a receding horizon manner. At time k, the path planner designs the path for steps $k+1, k+2, \ldots k+N_p$, where N_p is the *planning horizon*. By considering the speed and the maximum turning angle of the UAV and assuming that the UAV follows a straight line path in moving from time step $k+q$ to $k+q+1$, the new position of the UAV at each time step within the planning horizon can be easily identified. At each time step, the UAV has m choices to move within the maximum turning angle limit. Fig. 4-1 shows an example of the tree of waypoints for $m = 3$ and $N_p = 3$. The UAV executes N_e steps each time, which is called the *execution horizon* ($N_e < N_p$), and the planning algorithm repeats.

4.2 The Environment

The search area is discretized in space, as well as in time. At each planning step k, the UAVs carry a map of the environment in memory, and it is assumed that each UAV has the same map. This map describes the state of each cell, (x, y) at time k. This work uses a *probability map*, $P(x, y, k)$ which indicates the probability of targets being present at the location (x, y) at time k, and $P(x, y.k) \in [0, 1]$. As the UAV moves across the cell, it obtains sensor readings and updates this probability map.

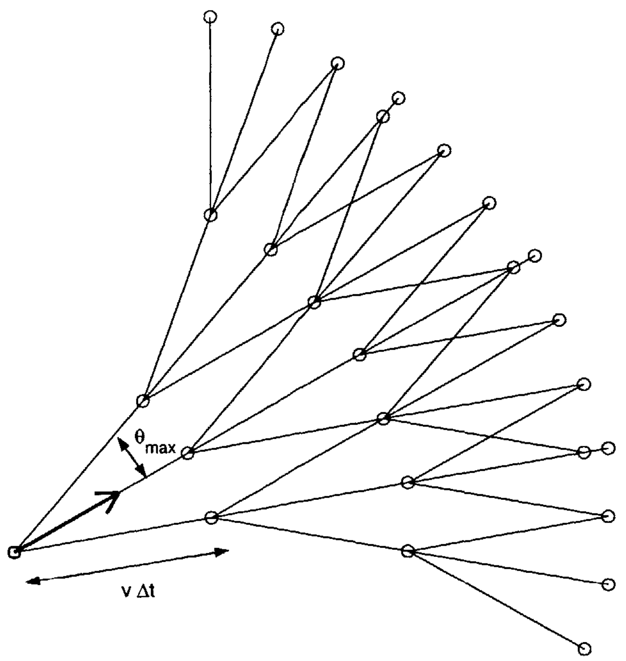


Figure 4-1: Tree of nodes for the path planning

4.2.1 Bayesian Update Rule

The probability map is updated using a Bayesian update rule [35]

$$\begin{aligned} P(x,y,k+1) &= b(x,y,k)\frac{\zeta P(x,y,k)}{\zeta P(x,y,k)+(1-\zeta)(1-P(x,y,k))} \\ &\quad + (1-b(x,y,k))\frac{(1-\zeta)P(x,y,k)}{\zeta(1-P(x,y,k))+(1-\zeta)P(x,y,k)} \end{aligned} \tag{4.1}$$

where $b(x,y,k)$ is a binary variable, with $b(x,y,k) = 1$ if the sensor reports a target detection in the cell (x,y) at time k, and $b(x,y,k) = 0$ otherwise. ζ is the sensor accuracy, which is defined as the probability of a correct sensor reading [35]

$$\zeta = P(A|b(x,y,k) = 1) \tag{4.2}$$

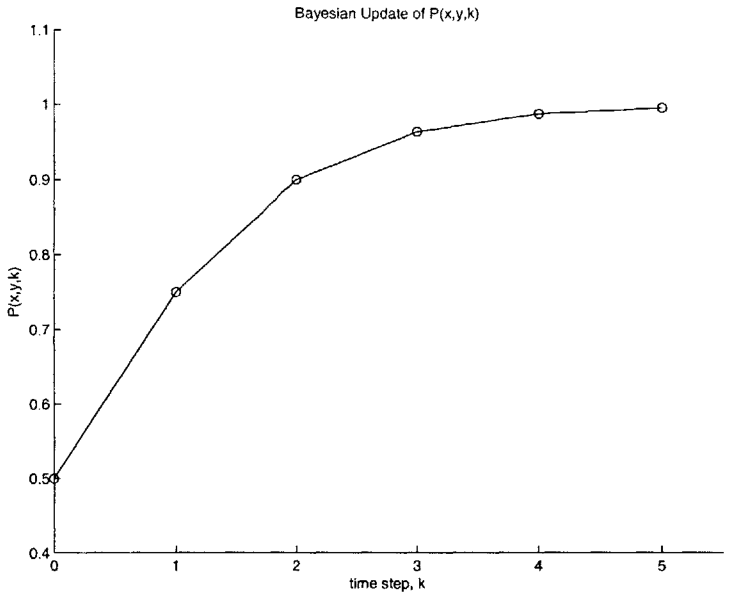


Figure 4-2: Illustration of Bayesian update rule for $b(x, y, k) = 1$.

where A is the event that the target is actually present at (x, y, k). Note that if $\zeta > 0.5$ (and hence the sensor gives useful information),

$$\begin{aligned} P(x,y,k+1) &> P(x,y,k) \quad \text{if} \quad b(x,y,k) = 1 \\ \text{and } P(x,y,k+1) &< P(x,y,k) \quad \text{if} \quad b(x,y,k) = 0 \end{aligned}$$

Fig. 4-2 illustrates the Bayesian update rule with $P(x, y, 0) = 0.5$ and $b(x, y, k) = 1$ for all k. It is shown that the rate of increase in $P(x, y, k)$ reduces, which means that searching an area that has already been searched does not produce as much increase in information as searching an area that has never been searched before. This emphasizes the importance of cooperation in the search since multiple UAVs searching the same area is likely to produce less gain than having them to search different areas.

4.2.2 Uncertainty Map

The location and number of targets are not known initially. The mission goal is to collect information in the searching area and find as many targets as possible. The amount of information obtained is quantified by measuring the *uncertainty* associated with the environment. The uncertainty, $U(x, y, k) \in [0, 1]$ is calculated as,

$$U(x, y, k) = -4\left(P^2(x, y, k) - P(x, y, k)\right) \tag{4.3}$$

Fig. 4-3 shows the plot for $U(x, y, k)$. For $U(x, y, k) = 1$, we are completely uncertain about whether a target is present at (x, y) at time k, which corresponds to $P(x, y, k) = 0.5$. For $U(x, y, k) = 0$, we are sufficiently confident to conclude whether a target is present at (x, y) at time k (or not), which corresponds to $P(x, y, k) = 1$ (or 0). Ref. [35] formulated the uncertainty as the Shannon entropy,

$$U(x, y, k) = -P(x, y, k)\log_2(P(x, y, k)) - (1 - P(x, y, k))\log_2(1 - P(x, y, k)) \tag{4.4}$$

Our formulation is inspired by Eq. 4.4. However, we modify the curve such that the slope is less steep at either end. Hence, the same change in probability results in a smaller change in uncertainty (Fig. 4-3). This is helpful particularly in problems involving moving targets. The uncertainty does not grow too rapidly as a result of untracked moving targets. The information remains useful for a longer time before the probability map becomes flat (or the uncertainty becomes high everywhere). Hence, the UAVs have more capability to re-acquire the moving targets after searching.

4.3 Algorithm For Static Targets

This section discusses the search problem with static targets only [34, 35]. The goal of the mission is to reduce the uncertainty in the area and find all of the targets. Given *a priori* information of the environment, the UAVs head to the region of interest (probably with high likelihood of finding targets there) and reduce the uncertainty

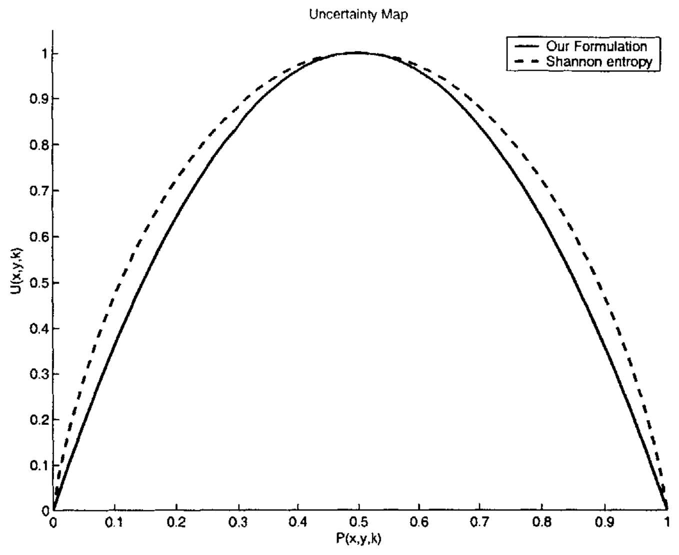


Figure 4-3: The uncertainty map.

on the way. This leads to a multi-objective optimization problem.

4.3.1 Mission Objectives

The UAV search mission consists a number of subgoals:

1. *Follow the path of maximum reduction in uncertainty.* – This subgoal considers the uncertainty reduction associated with the region swept by the sensor between time step k and $k+1$. (See the rectangular regions (immediate gain) in Fig. 4-4).
2. *Follow the path that will guide the UAV to regions of maximum uncertainty in the future.* – This subgoal causes the UAV to choose a path that has potentially more reduction in uncertainty in the next planning step, i.e. beyond the planning horizon. (See the sector region (look ahead gain) in Fig. 4-4).
3. *Follow the path of maximum probability of finding targets.* – This subgoal considers the sum of $P(x, y, k)$ of the cells associated with the region swept by the sensor between time step k and $k+1$. It guides the UAVs along a path of high probability of target presence. (See the rectangular regions (immediate gain) in Fig. 4-4).

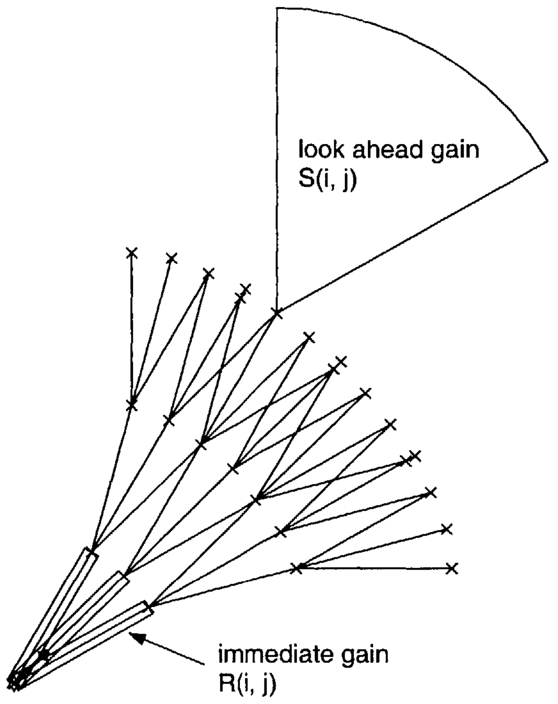


Figure 4-4: Illustration of the regions for computing the cost functions.

4. *Follow the path that will guide the UAV to regions of highest probability of finding targets in the future.* – This subgoal causes the UAV to choose a path that has higher potential of finding targets in the next planning step, i.e. beyond the planning horizon. (See the sector region (look ahead gain) in Fig. 4-4).

5. *Cooperation within the team.* – This subgoal minimizes the overlap of the UAV trajectories. As discussed in Section 4.2.1, the searching is more efficient if the UAVs search different regions. Hence, by minimizing the overlap of the trajectories, it maximizes the utilization of all the UAVs in the team.

Subgoals 2 and 4 serve as a heuristic to approximate the environment beyond the planning horizon [32]. Hence, the planning is not just confined within the planning horizon, but the UAV is still aware of the rest of the environment.

4.3.2 Objective Function

Each of the subgoals is associated with an individual objective function, J_i. The overall cost function can be written as,

$$J = w_1 J_1 + w_2 J_2 + \ldots + w_n J_n$$

where J_i represents the cost function for the $i-th$ subgoal, and w_i is the corresponding weight. The weights are normalized, i.e. $0 \leq w_i \leq 1$, and $\sum_i w_i = 1$. These w_i's represent the trade-offs between the subgoals, since they are sometimes competing. The objective functions for the subgoals are formulated as follows:

For subgoal 1

$$J_1(i,j,k) = \sum_{(x,y)\in R_{i,j}} (U(x,y,k) - U(x,y,k+1)) \tag{4.5}$$

$J_1(i,j)$ is the reduction of uncertainty for UAV i to select node j at time k, and $R_{i,j}$ is the corresponding rectangular region associated with immediate gain as shown in Fig. 4-4.

For subgoal 2

$$J_2(i,j,k) = \sum_{(x,y)\in S_{i,j}} U(x,y,k) \tag{4.6}$$

$J_2(i,j)$ is the potential reduction on the uncertainty within the sector region $S_{i,j}$ in Fig. 4-4.

For subgoal 3

$$J_3(i,j,k) = \sum_{(x,y)\in R_{i,j}} (1 - \chi(x,y))P(x,y,k) \tag{4.7}$$

$J_3(i,j)$ is the award for finding a target at (x,y,k). $\chi(x,y)$ is a binary variable that indicates whether a target is confirmed to be present at (x,y).

$$\chi(x,y) = \begin{cases} 1 & \text{if} \quad P(x,y,k) \geq \Gamma \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

where Γ is a pre-defined threshold value close to 1.

For subgoal 4

$$J_4(i,j,k) = \sum_{(x,y)\in S_{i,j}} (1-\chi(x,y))P(x,y,k) \tag{4.9}$$

$J_4(i,j)$ is the potential award of finding targets within the sector region $S_{i,j}$ in Fig. 4-4. This serves as a driving term to "pull" the UAVs to a region of high likelihood of finding new targets.

For subgoal 5

$$J_5(i,j,k) = \sum_{j}\sum_{l\neq i} f(d_{lj},\theta_{lj}) \tag{4.10}$$

$f(d_{lj},\theta_{lj})$ is a non-decreasing function in d_{lj} and θ_{lj}, where d_{lj} and θ_{lj} are the distance and angle between node j and UAV l respectively. Hence, by cooperation, we attempt to increase the difference in position and heading angle of the UAVs such that they would search different areas. As shown in Fig. 4-2, it is more beneficial to have the UAVs search different area and to search areas that have never been searched.

After normalization, the overall objective function is given by,

$$\bar{J}(i,j,k) = w_1\bar{J}_1 + w_2\bar{J}_2 + w_3\bar{J}_3 + w_4\bar{J}_4 + w_5\bar{J}_5 \tag{4.11}$$

where $\bar{J}_i$ is the normalized objective function. Hence, the UAV selects a path that maximizes $\bar{J}(i,j,k)$ as computed in Eq. 4.11 with given w_i.

4.3.3 Simulation Results

This section presents a simulation showing the results of the above searching algorithm. The scenario used for the simulation is shown in Fig. 4-5. The search region is a 100×100 area, which is discretized into 1×1 cells. The search region is highly uncertain initially, but there is some à priori information about the search area, as shown by the probability map in Fig. 4-6. In particular, it is initially believed that there is high probability of finding targets in some regions (the circular regions in Fig. 4-5). There are 8 static targets placed in the search region and we consider a search mission using 3 UAVs, which start at the bottom left corner in Fig. 4-5. The results of the search algorithm are compared with a random search. In a random search, the UAV randomly selects one of the m waypoints within the heading constraints at each time step with equal probability. The parameters for the simulations are:

1. $v\Delta t = 5$
2. $\theta_{\max} = \frac{\pi}{12}$
3. $m = 3$
4. $N_p = 3$
5. $N_e = 1$
6. $\zeta = 0.75$
7. $\Gamma = 0.85$
8. $w = [0.1 \quad 0.13 \quad 0.17 \quad 0.18 \quad 0.18]$

The values of w_3 and w_4 are chosen to be relative high compared to w_1 and w_2, which emphasizes finding all of the targets within the simulation time. The simulations are run for 500 time steps. The outputs for one of the simulations are shown. Fig. 4-7 shows the average uncertainty of the search region at each simulation step. Fig. 4-8 shows the number of targets found over the entire simulation. Fig. 4-7 shows that the uncertainty level in the search region decreases as the searching mission proceeds. Hence, we are obtaining more and more information about the region. Furthermore, the search algorithm results in a higher rate of decrease in uncertainty than the random search. More specifically, the algorithm search has reduced the uncertainty from 99.5% to only 7.5%, while the random search has reduced the uncertainty to only 23%. The search algorithm was able to find all 8 targets in the region, but the random search managed to find only 6 of them, as shown in Fig. 4-8. Moreover, the search algorithm is able to find the targets at an earlier time, which is beneficial because the value of finding the targets would probably decrease with time.

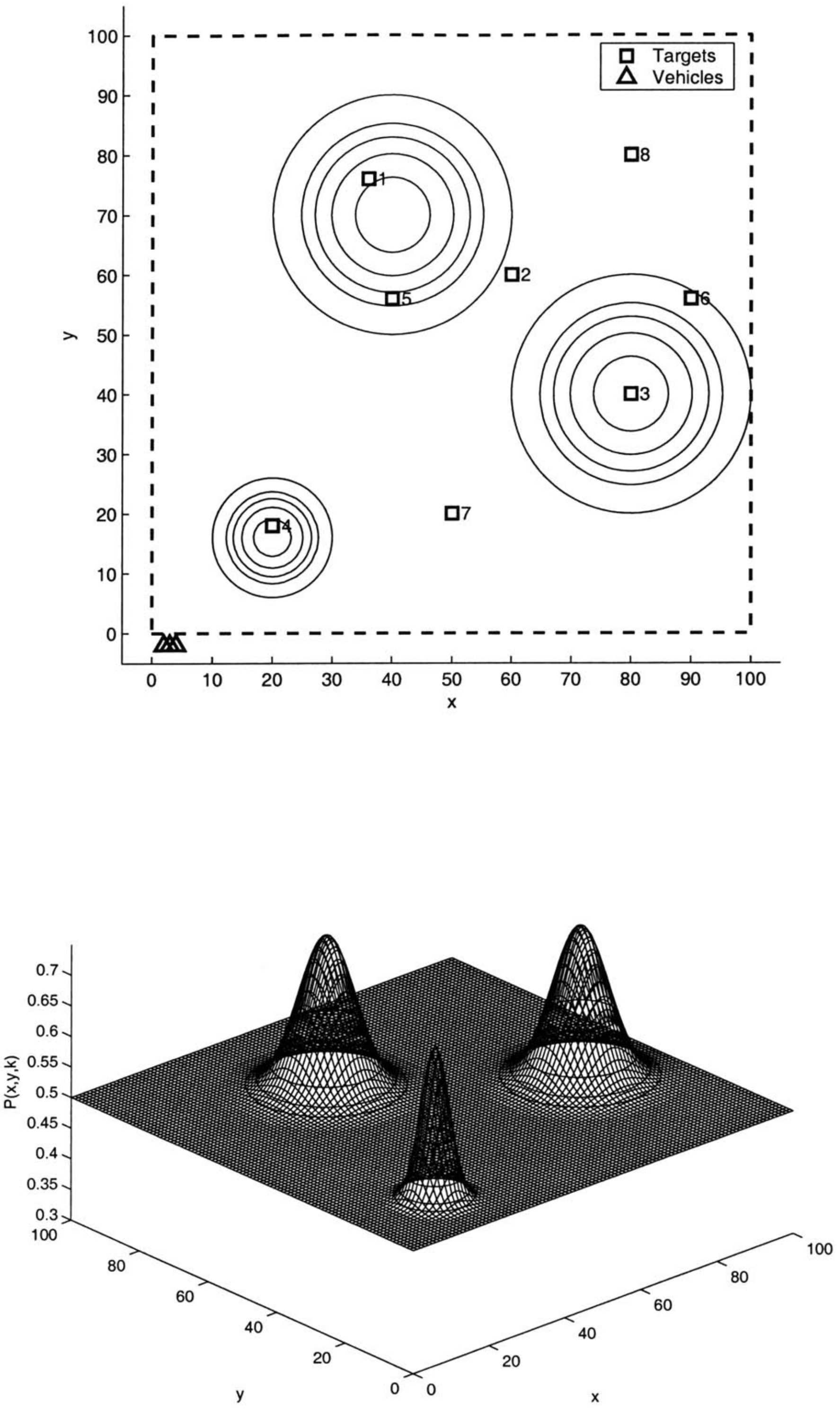


Figure 4-6: Initial probability map.

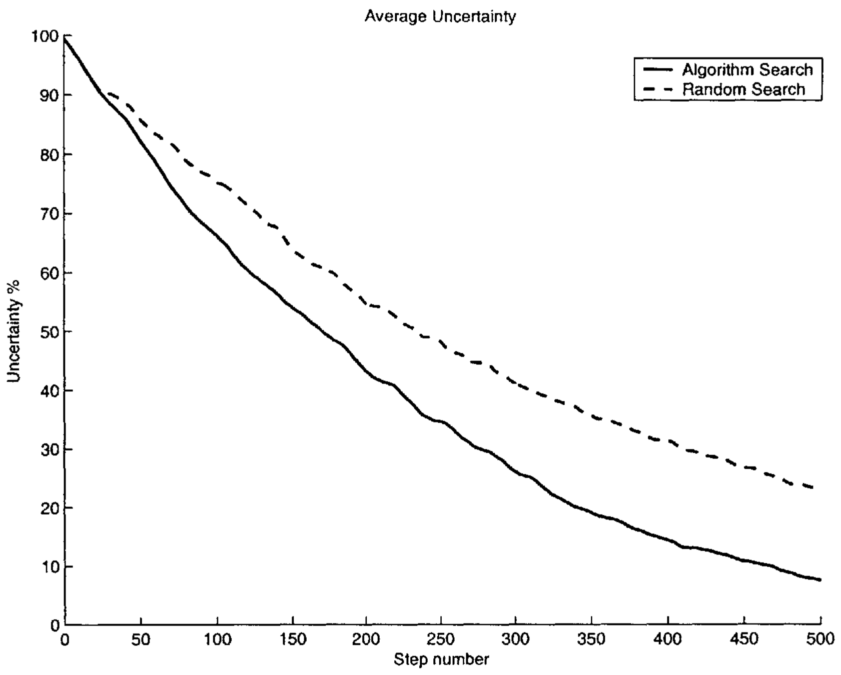


Figure 4-7: Average uncertainty for algorithm search and random search.

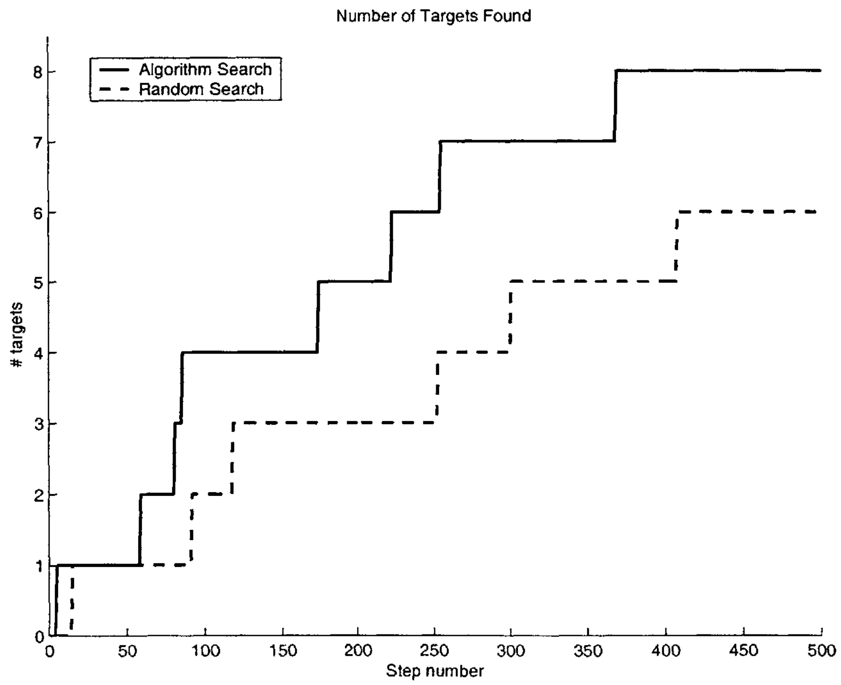


Figure 4-8: Number of targets found for algorithm search and random search.

This section has discussed the algorithm for cooperative search mission. Similar results can be also found in Refs. [32, 33]. The following section extends the work to handle moving targets. As the targets are not static, if they are not under track, we lose information about them and the uncertainty will increase. On the other hand, there may be other high value targets in the region. So, the planner has to compromise between the *search* and *track* action of the UAVs.

4.4 Moving Targets

In the UAV mission, targets, such as trucks and tanks can be moving around. When the target leaves the field of view, the information about the target is lost. The only way to maintain accurate information of the target is to keep tracking it as it moves around, but this loses that resource for searching other regions to discover new targets. This is a challenging problem in general, so we make a fundamental assumption for the following discussion. In particular, we assume that the targets will be moving in a reasonably predictable manner, for instance, moving in a straight line with a bounded velocity range. This assumption is realistic for vehicles traveling on a road, but less so for vehicles driving over difficult terrain. With this assumption, this section describes the modeling of the uncertainty growth associated with the expected movement of the target. Then, a new objective function associated with the track action is appended to the objective function in Section 4.3.2.

4.4.1 Uncertainty Growth Associated with Target Movement

This section presents a model of the uncertainty growth as an untracked target moves around. The growth of uncertainty happens around the expected position of the target at each time step. As the uncertainty increases, it will eventually attract attention from the UAV team. If the expected position of the target is consistent with the actual position, then the UAV can re-acquire the target easily. However, this requires that the target moves in a way that can be predicted. For example, if the UAV finds a car on a road, we might expect the car to keep moving along the road for a certain time span. Hence, given the velocity of the car, the position of the car can be projected in time, and the UAV can come back later to track the car again.

By tracking the target at time k, we obtain a measure of the targets position (x, y), velocity v, and heading θ. We assume that the sensor error is such that there is a bounded variation in both velocity and heading measurements. Hence the knowledge about the target during track is given by (x, y), $[v_l, v_u]$ and $[\theta_l, \theta_u]$. It is assumed that the movement of the target is predictable, such that the motion at time $k+T$ $(T > 0)$ is correlated with the motions for all time $t < k+T$ while tracking. Hence, by using a simple polynomial fitting technique, we can obtain a prediction in the velocity and heading as a function of time. Moreover, as the target is being tracked for several consecutive time steps, the expected variations in velocity and heading shrink. So, the confidence in the prediction increases. As a summary, by tracking, we have,

$$\text{TRACK} \quad \begin{cases} v(k) = \bar{v}(k) \pm \epsilon_v(k) \\ \theta(k) = \bar{\theta}(k) \pm \epsilon_\theta(k) \end{cases} \tag{4.12}$$

where $\bar{v}(k)$ and $\bar{\theta}(k)$ are the nominal predictions in velocity and heading respectively, and $\epsilon_v(k)$ and $\epsilon_\theta(k)$ are the corresponding errors. Furthermore, the $\epsilon_v(k)$ and $\epsilon_\theta(k)$ are expected to decrease with time as the target is tracked.

Hence, given a measured position (x_0, y_0) of the target at time k, we would expect the target to be inside the shaded area in Fig. 4-9 at time $k+1$ if we leave the target untracked after time k. If ϵ_v and ϵ_θ are small, the shaded area is sufficiently small such that we would have a high probability of finding the target again in a short time. We assign a 2-D probability distribution for this shaded area as shown in Fig. 4-10, which transforms the velocity and heading estimations into the information state.

When ϵ_v and ϵ_θ are small, the loss in information (or increase in uncertainty) associated with the shaded area in Fig. 4-9 is small. Hence, it will be more beneficial to have the UAV move to another region to search, instead of keeping track on the target. This switches the *track* action into the *search* action. On the other hand,

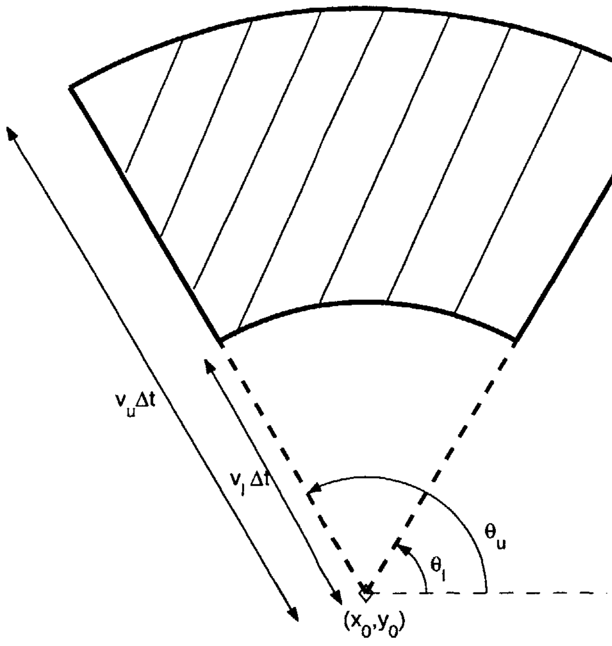
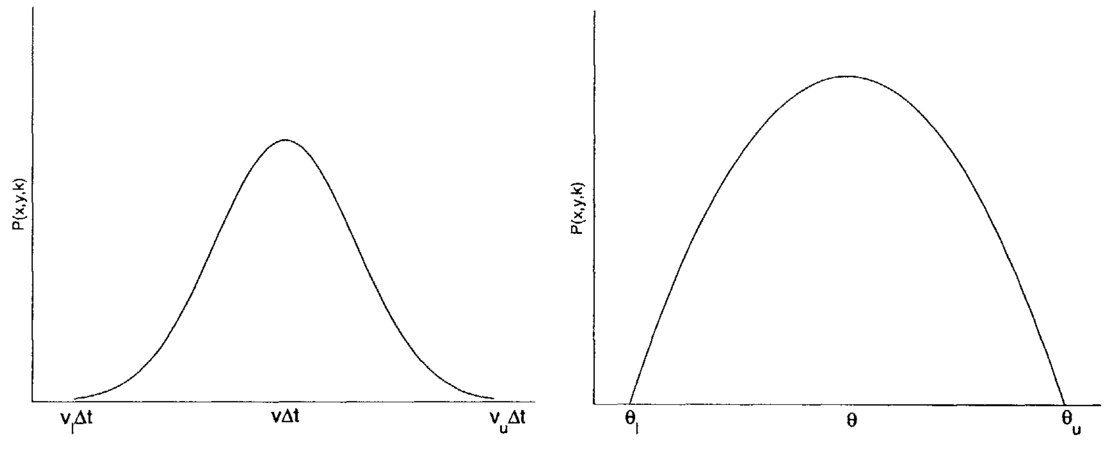


Figure 4-9: Predicted position of moving target.

while the target remains untracked, its position over time is estimated using (x_0, y_0), $\bar{v}(k)$ and $\bar{\theta}(k)$. However, ϵ_v and ϵ_θ will grow with time, so the shaded area increases as well as the uncertainty associated with it. Hence, during *not tracked* (or *search*), we have,

$$\text{SEARCH} \begin{cases} (\bar{x}(k), \bar{y}(k)) = f(x_0, y_0, \bar{v}(k), \bar{\theta}(k)) \\ \dot{\epsilon_v}(k) = g(\epsilon_v(k)) \\ \dot{\epsilon_\theta}(k) = w(\epsilon_\theta(k)) \end{cases} \tag{4.13}$$

$g(\epsilon_v(k))$ and $w(\epsilon_\theta(k))$ are linear functions in ϵ_v and ϵ_θ respectively. Hence ϵ_v and ϵ_θ grow exponentially with k. As ϵ_v and ϵ_θ grow with time, the uncertainty around the target will eventually become so high that it attracts the attention of the UAVs, according to the objective function given in Section 4.3.2. However, if the prediction is bad, or the target remains not tracked for a long time, then the shaded area becomes very large and the associated probability map becomes flat. Hence, the UAVs essentially have no useful information about the target and have to start searching for the target from scratch. Hence, the compromise between *search* and *track* is important.



(a) $v\Delta t$. (b) θ.

Figure 4-10: Probability distribution for target movement.

4.4.2 Objective Functions

The objective function in Section 4.3.2 is formulated as a multi-objective optimization problem. Hence, we can easily add on a new objective function associated with tracking to Eq. 4.11. For tracking, the objective function is formulated as,

$$J_6(i,j,k) = \sum_{p\in S_T}\left(\sum_j h(d_{jp}) + \sum_{(x,y)\in T_{i,j,p}} (1-b_T(j,p))\left(U(x,y,k)-U(x,y,k+1)\right)\right) \quad (4.14)$$

where S_T is the set of targets within the tracking range. $h(d_{jp})$ is a non-increasing function in d_{jp} which is the distance between Waypoint j and Target p. $T_{i,j,p}$ is the shaded area for Target p shown in Fig. 4-9, and $b_T(j,p)$ is a binary variable indicating whether the Waypoint j is within a tracking distance of Target p. $b_T(j,p) = 1$ means the UAV is expected to be able to keep tracking Target p if it travels to Waypoint j.

As the UAV keeps tracking the target (*i.e.*, maintains a short distance to the target), $|T_{i,j,p}|$ decreases, as well as $U(x,y,k+1)$. Hence, to maximize the objective function, the best strategy would be for the UAV to keep the target under track until it gains enough information to predict the behavior of the target in the future. At that point it is sufficiently confident that it can leave the target to start searching another region and be able to return later to find the target again. Now, the overall objective function is,

$$\bar{J}(i,j,k) = w_1\bar{J}_1 + w_2\bar{J}_2 + w_3\bar{J}_3 + w_4\bar{J}_4 + w_5\bar{J}_5 + w_6\bar{J}_6 \tag{4.15}$$

where $\bar{J}_i$ are the normalized objective functions.

4.4.3 Simulation Results

A simulation is presented here to compare the search & track algorithm with the random search & track. The scenario is shown in Fig. 4-11. The search region is again a 100×100 area, discretized into 1×1 cells. 3 UAVs start from the bottom left corner. The simulations are run for 1000 time steps. There are 6 static targets and 3 moving targets in the region. The number and type of the targets are not known to the UAVs. When a UAV finds a new target, after taking several steps (number of steps is pre-defined prior to the simulation) of sensor data from tracking, it will be able to classify the target as static or moving using the velocity estimates. The moving targets move roughly in piece-wise linear paths. The nominal paths are shown as dotted lines in Fig. 4-11. The targets move along its own path with bounded random perturbations in (x, y). Prior to the start of the mission, à priori information indicates that there is high probability of finding the target in the circular regions (around Targets 7,8 and 9), as shown in Fig. 4-12. We assume that the targets are moving with a certain speed from initial estimates. However, we do not have any reliable information about its heading direction. Hence, the circular regions grow (as well as the overall uncertainty) radially in all directions before Targets 7,8 and 9 are found. The parameters for the simulations are given as below,

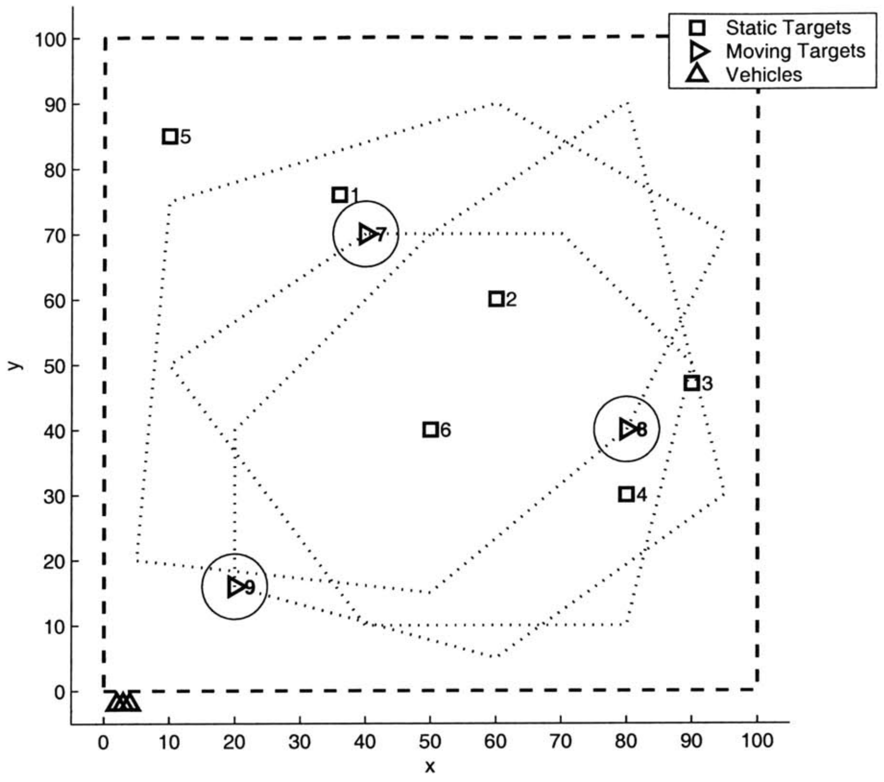
1. $v\Delta t = 5$
2. $\theta_{max} = \frac{\pi}{7.2}$
3. $m = 3$
4. $N_p = 3$
5. $N_e = 1$
6. $\zeta = 0.75$

7. $\Gamma = 0.85$
8. $w = [0.12 \quad 0.15 \quad 0.13 \quad 0.17 \quad 0.23 \quad 0.2]$
9. Number of tracking steps required to classify the type of the targets = 4
10. Tracking range = 10
11. $v\Delta t$ of moving targets $\approx$ 1–3

The speed (or $v\Delta t$) of the targets is lower than that of the UAVs so that they have a capability to re-acquire the targets. The UAVs also are assumed to have a larger maximum turning angle than in Section 4.3.3, so that they are more maneuverable, which helps keep the targets within tracking range. There are special path planning strategies for a UAV to track ground UAVs [36], but they are beyond the scope of this thesis.

Fig. 4-13 shows the average uncertainty at each simulation step. In contrast to Fig. 4-7 where the uncertainty keeps decreasing as the mission proceeds, the uncertainty in Fig. 4-13 increases when the targets are left untracked. On the other hand, when the UAVs re-acquire the target, there is a significant drop in uncertainty because the regions of suspicious presence of the targets are being "cleared up". If the work of the UAVs is efficient, the uncertainty would converge to some steady-state value. This reflects a good compromise between the search and the track action, such that it maintains a certain level of knowledge for the environment. It can be observed that the search & track algorithm is capable of converging the uncertainty to a steady-state value of around 30%. However, there is no clear convergence of uncertainty for the random search. The uncertainty oscillates with large amplitudes. Moreover, the search & track algorithm is able to find all targets (Fig. 4-14), and classify the static targets (Fig. 4-15) at an earlier stage of the mission.

Fig. 4-16 and Fig. 4-17 show the tracking history of the two algorithms. A "high" signal indicates that the target is being tracked by any of the UAVs at that time step. The uncertainty curve is also superimposed on the plot. It is shown that there is a significant drop in uncertainty when any target gets tracked. The drop is more significant when there is a large time gap between two consecutive track. The random search in Fig. 4-17 shows this behavior. However, this is not desirable since there will



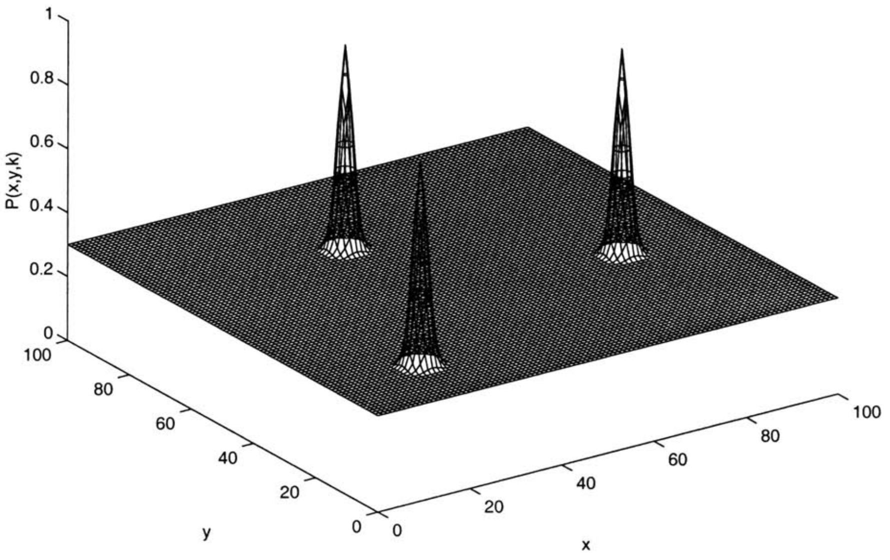


Figure 4-12: Initial probability map.

be a large fluctuation in information. On the other hand, Fig. 4-16 shows that the search & track algorithm has a more efficient tracking history and that the fluctuation in information is small after step 300. Continuous tracking of the same target does not produce much further decrease in uncertainty, however, it provides more information for predicting the behavior of the targets. The search & track algorithm has made use of this information and results in a more efficient tracking history.

Consider time steps 500–600 in Fig. 4-16, where the uncertainty has been kept at a low level over a relatively long period of time. There is a high frequency of tracked signal within this period. By looking at Fig. 4-14 it is clear that all targets are found after around step 500. As a result, the overall uncertainty is quite low. The high uncertainty only concentrates around the moving targets. Hence, the track action dominates over the search action. This reflects the behavior in which, when the UAVs believe they have searched over the entire region, the search action, or the "finding target" action is no longer necessary. Hence, they restrict their motion to be around the moving targets. The mission essentially becomes a "track-only" mission.

Fig. 4-18 shows the uncertainty plot with a similar scenario as before, but with one additional moving target. Hence, the number of moving targets is larger than the number of UAVs. It is shown that for the search & track algorithm, the uncertainty still tends to converge to a steady state value. Since the number of moving targets is now larger than the number of UAVs, the steady state value is larger (about 35%). Moreover, the oscillation in uncertainty increases since more moving targets results in more rapid growth of uncertainty. However, the search & track algorithm still shows its strength over the random algorithm for maintaining the information at a fairly constant level.

4.5 Conclusions

This chapter presents a multi-UAV search algorithm adopted from Refs. [32, 33, 34, 35]. The work is extended by incorporating the *track* action into the objective function

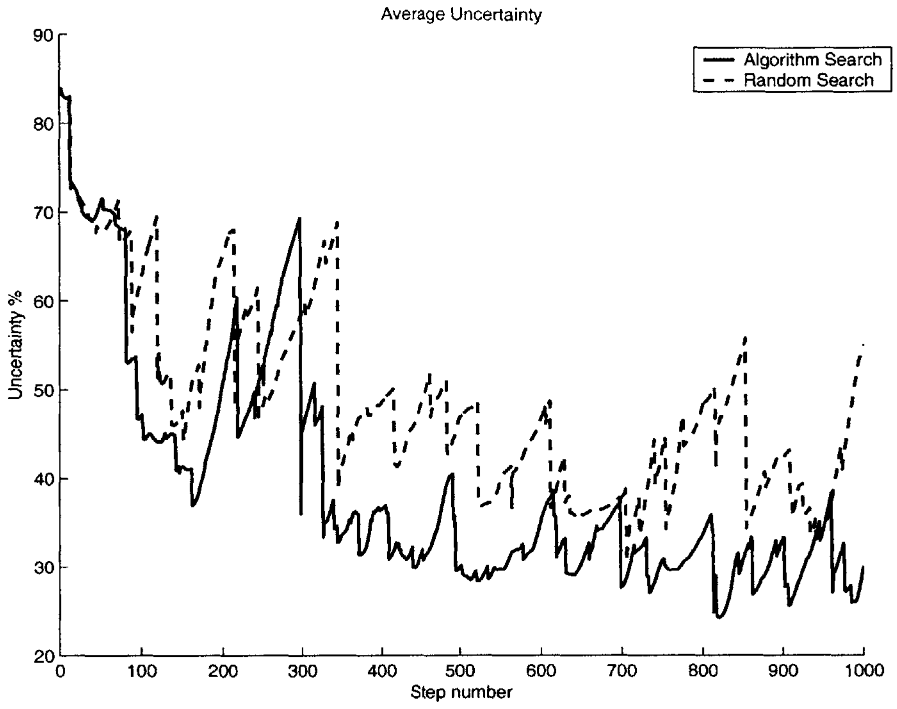


Figure 4-13: Average uncertainty for algorithm search and random search.

in order to handle moving targets. The algorithm has shown higher efficiency over the random algorithm given some predictable behavior of the targets. The simulation result shows the behavior of the UAV for loitering around the moving targets when it believes it is not likely to find new targets in the region (i.e. when the uncertainty is low). Moreover, simulation results also show the success of the algorithm to handle scenario when the number of moving targets is larger than the number of UAVs.

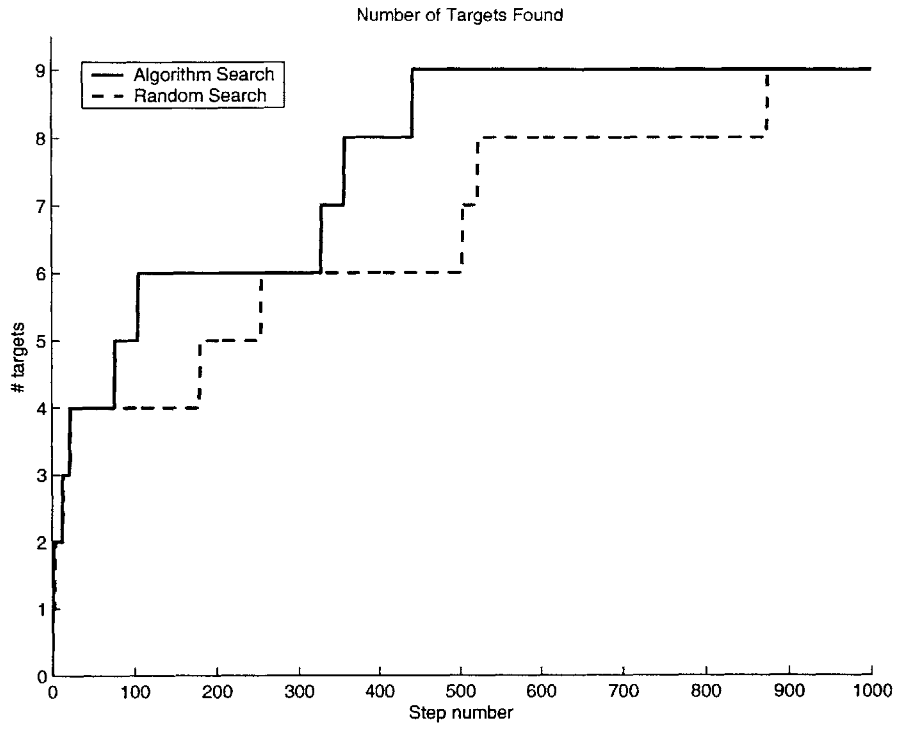


Figure 4-14: Number of targets found for algorithm search and random search.

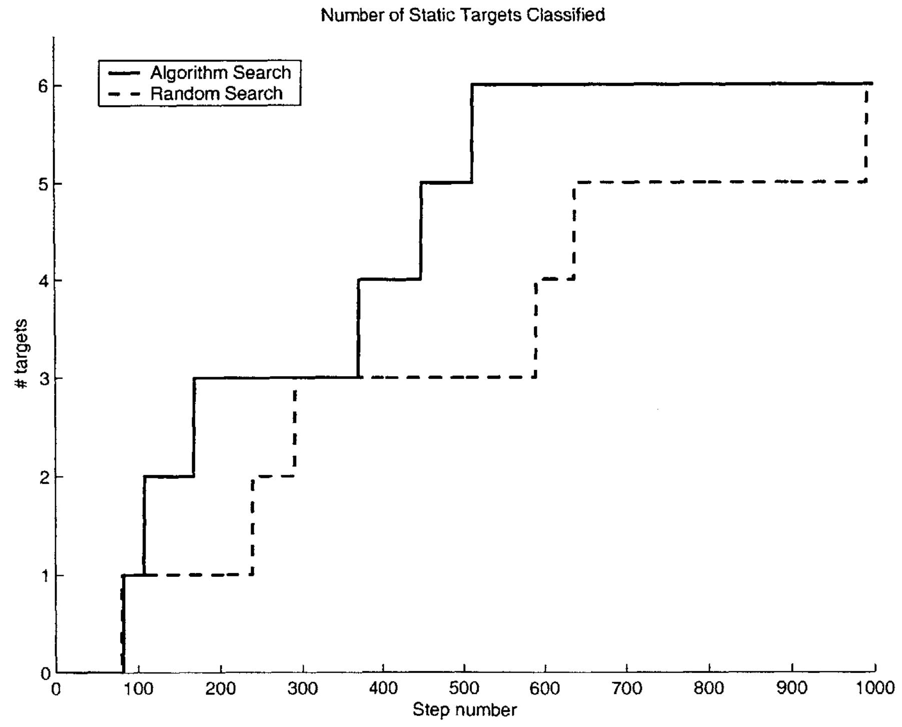
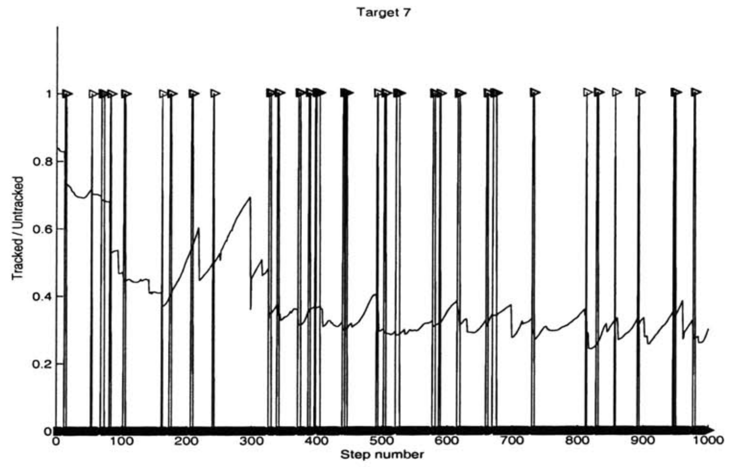
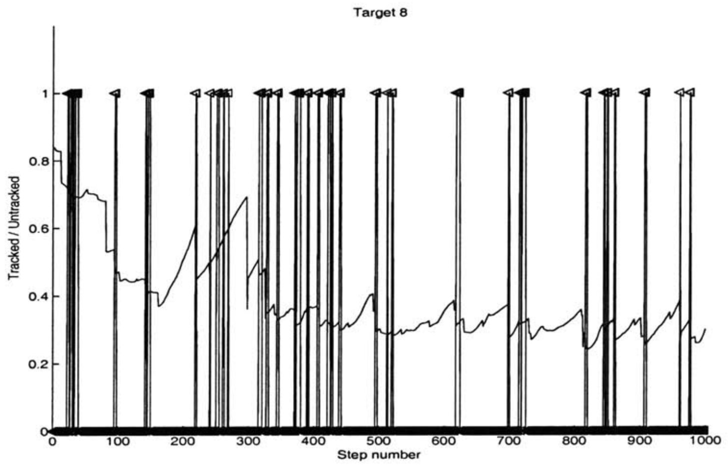


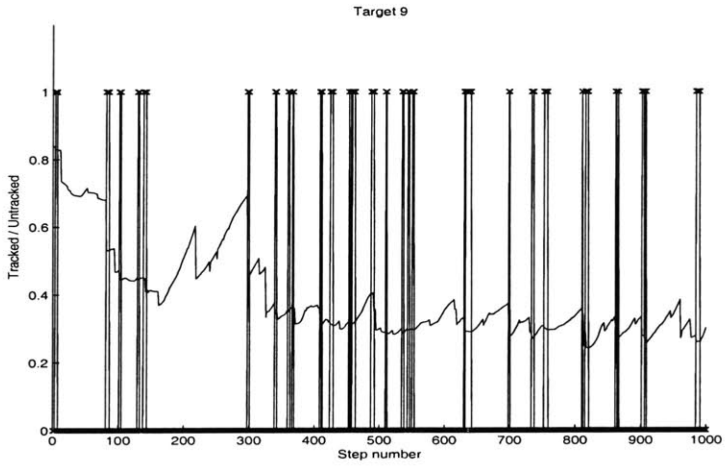
Figure 4-15: Number of static targets classified.



(a) Target 7.

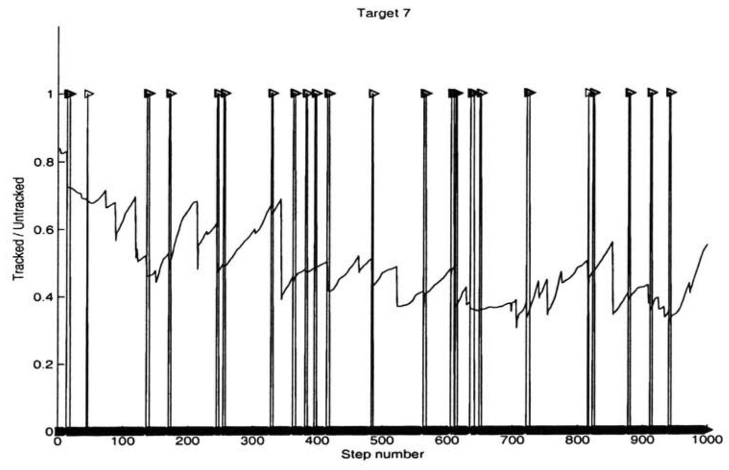


(b) Target 8.

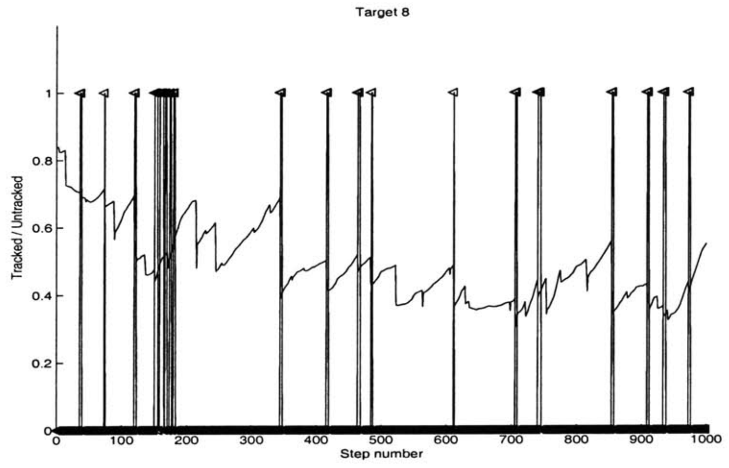


(c) Target 9.

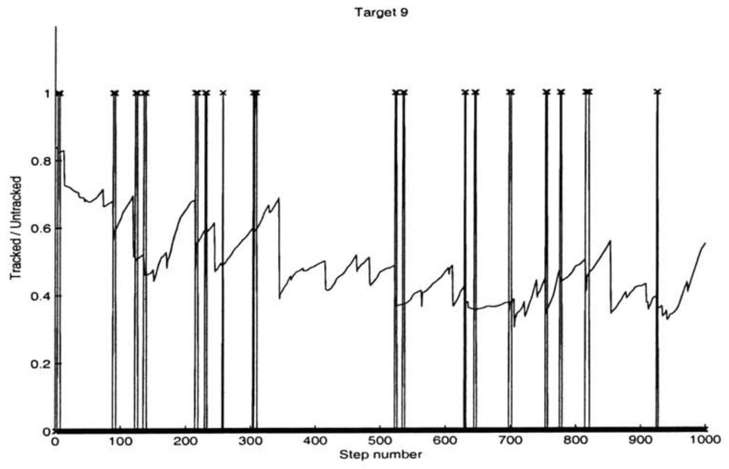
Figure 4-16: Tracking history of algorithm search.



(a) Target 7.



(b) Target 8.



(c) Target 9.

Figure 4-17: Tracking history of the random search approach.

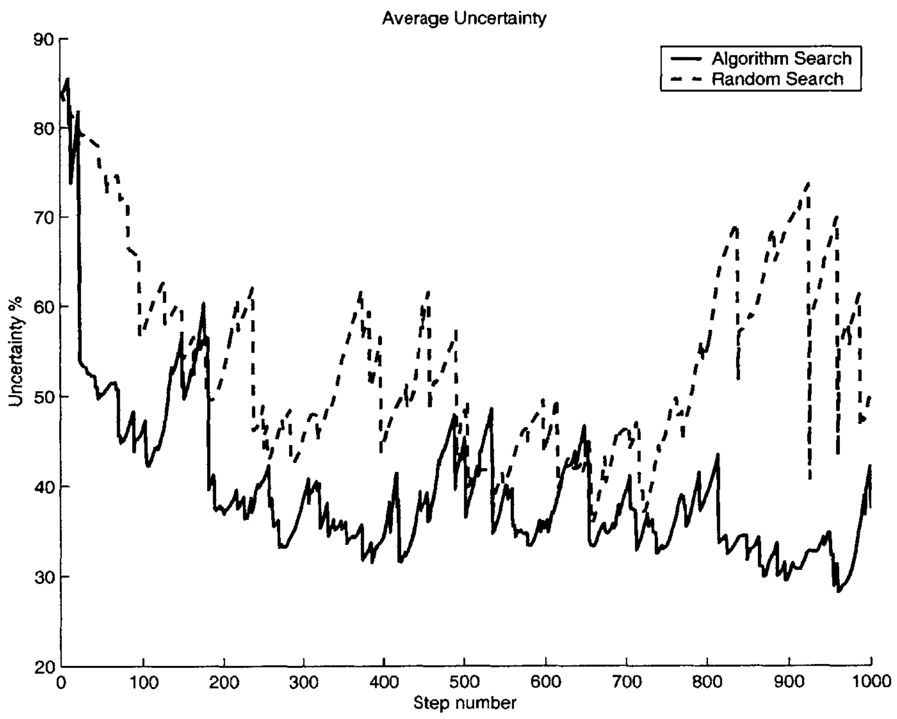


Figure 4-18: Average uncertainty for algorithm search and random search (4 moving targets).

Chapter 5

Testbed Implementation and Hardware Experiment

This chapter presents a rover testbed developed in the Aerospace Controls Lab. at MIT. The testbed has been used to demonstrate real-time implementation of the RH-MILP planning algorithm [25, 29]. Furthermore, a Graphic User Interface (GUI) tool has been recently developed to facilitate the design of complex and highly dynamic scenario, which can be readily implemented in both simulation and hardware experiment.

Section 5.1 introduces the rover testbed and the Indoor Positioning System (IPS), as well as the GUI. Section 5.2 shows an experimental result for implementing the ARSP algorithm on the rover testbed.

5.1 Hardware Testbed

This section introduces the hardware which consists of 8 rovers and an Indoor Positioning System. The rovers are constrained to move at constant speed with a minimum turning radius. Hence, they can be used to simulate typical UAV characteristics [13]. A precise Indoor Positioning System is used for localization of the rover. Fig. 5-1 shows the setup of the testbed with 2 rovers and 2 transmitters in the testing area.

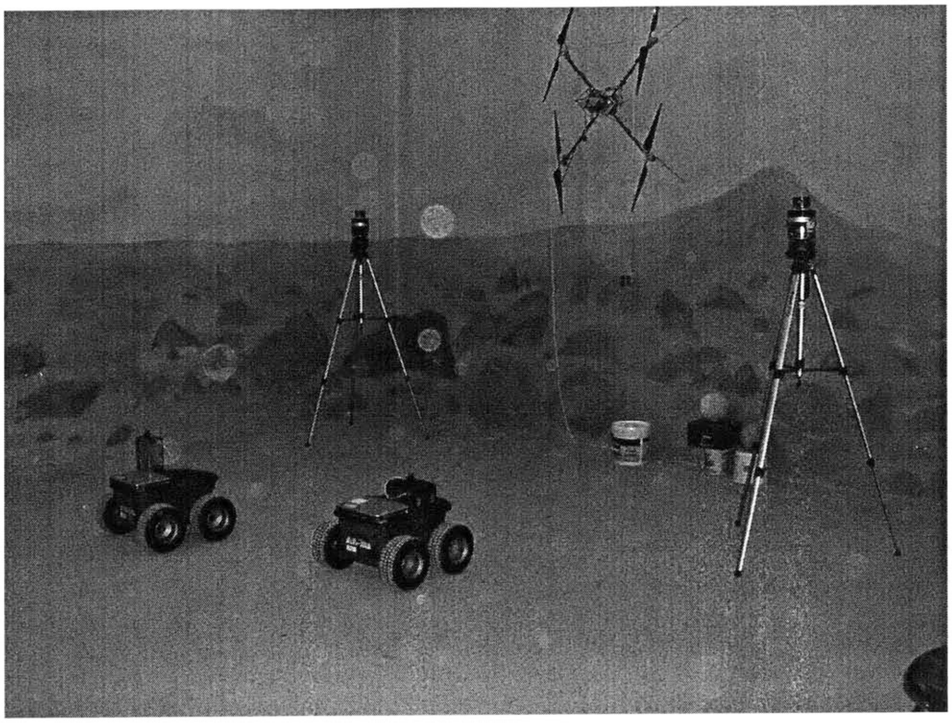


Figure 5-1: The setup of entire rover testbed (rovers and IPS).

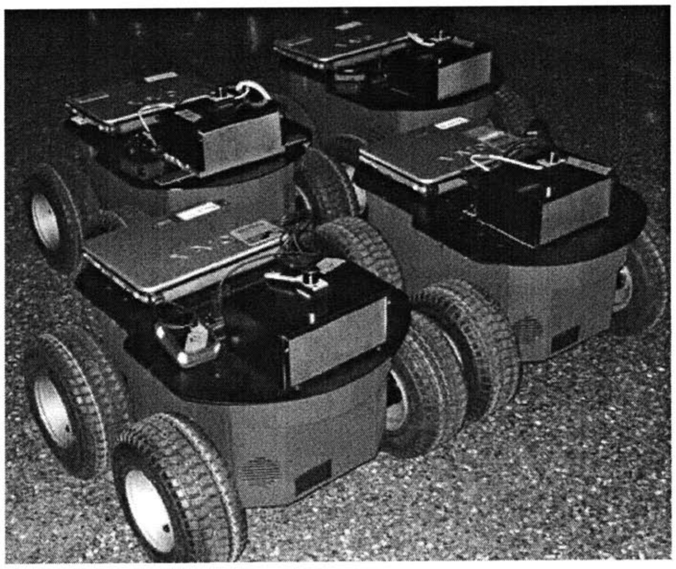


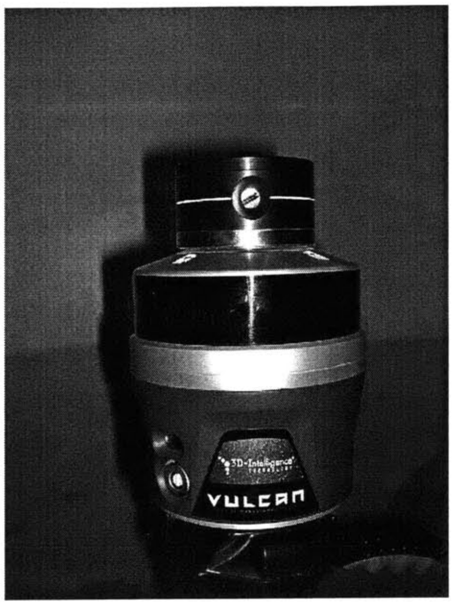
Figure 5-2: 4 of the 8 rovers.

5.1.1 Rovers

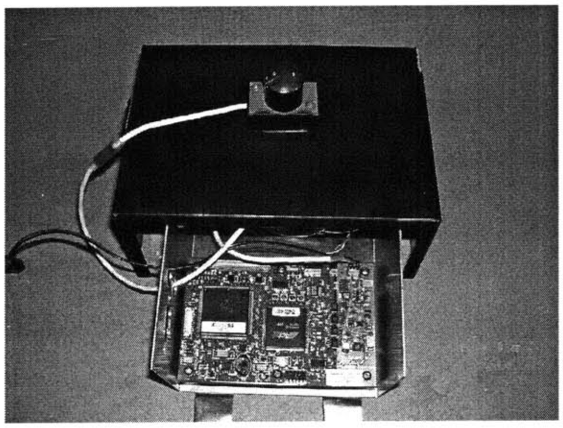
The rover testbed (Fig. 5-2) consists of a mixture of 8 Pioneer-2 and -3 power-steered vehicles manufactured by ActivMedia Robotics. The rovers operate under the ActivMedia Robotics Operating Systems (AROS) software, supplemented by the ActivMedia Robotics Interface for Applications (ARIA) software written in C++, which interfaces with the robot controller and facilitate incorporating user-developed code with the on board software [30]. An onboard Sony VAIO Pentium III (850MHz) sits on the rover to generates low level control commands, which are converted into PWM signals to drive the rover through serial communication. An IPS receiver is also installed on the rover for localization purpose. A Kalman filter is implemented to estimate the velocity and smooth the estimated position generated by the IPS.

5.1.2 Indoor Positioning System (IPS)

The IPS [31] is an ArcSecond 3D-i constellation metrology system which is comprised of 4 *transmitters* (Fig. 5-3(a)) and 12 *receivers* (Fig. 5-3(b)). A minimum of 2 transmitters are required to obtain position information, however, additional transmitters can increase visibility and range. The transmitter generates two vertically fanned *infrared (IR) laser beam* and a *LED strobe.* The fanned IR laser beam has an elevation angle of $\pm 30°$. Hence, any receivers coming too close to the transmitter cannot receive the signal from the transmitter. The position information are in Cartesian coordinate system (XYZ), with an uncertainty in measurement on the order of 0.4 mm (3σ). A PCE board connected to the receiver converts the IR beams and LED strobe information into timing pulses. Since each transmitter has a unique rotation speed, the period of the signals identify the transmitter. The position information are then calculated by the *Workbench* software, using the horizontal and vertical angle measurements. More detailed information is given in Ref. [29].



(a) The transmitter.



(b) The receiver and PCE board

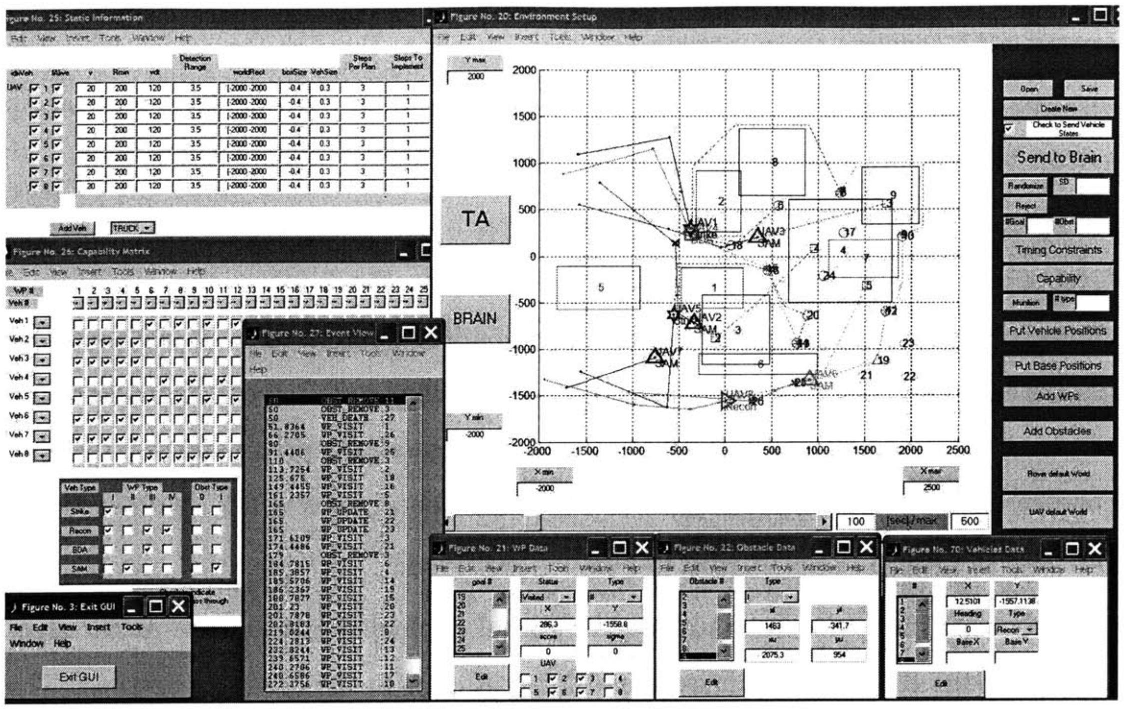
Figure 5-3: Indoor Positioning system (IPS).

5.1.3 Graphical User Interface (GUI)

A Graphical User Interface (GUI) software (Fig. 5-4) is developed under MATLAB 6.5. The GUI is a useful tool to facilitate the process of generating a large scale, complex and highly dynamic scenario, both for simulation and hardware experiment.

Previously, the information of the scenario is written in MATLAB m-file. To evaluate the setup of the scenario, we have to run the entire simulation every time and make the changes by trial-and-error. This is a very time-consuming procedure when the scenario is very complicated. Hence, a GUI that allows visualization of the scenario and step-by-step systematic way of setup of the scenario is very important.

The current GUI has included all of the functionality required to set up a scenario for the RH-MILP simulation and hardware-in-the-loop experiment. The user is can define vehicles to be used and their capability, place targets and obstacles, add timing constraints, and so on. Moreover, dynamic changes can be incorporated in the middle of simulation through the GUI. For instance, the user can "kill" a vehicle,



idxVeh	isAlive	v	Rmin	vdt	Detection Range	worldRect	boxSize	VehSize	Steps Per Plan	Steps To Implement
UAV ✓ 1	✓	20	200	120	3.5	[-2000 -2000	-0.4	0.3	3	1
✓ 2	✓	20	200	120	3.5	[-2000 -2000	-0.4	0.3	3	1
✓ 3	✓	20	200	120	3.5	[-2000 -2000	-0.4	0.3	3	1
✓ 4	✓	20	200	120	3.5	[-2000 -2000	-0.4	0.3	3	1
✓ 5	✓	20	200	120	3.5	[-2000 -2000	-0.4	0.3	3	1
✓ 6	✓	20	200	120	3.5	[-2000 -2000	-0.4	0.3	3	1
✓ 7	✓	20	200	120	3.5	[-2000 -2000	-0.4	0.3	3	1
✓ 8	✓	20	200	120	3.5	[-2000 -2000	-0.4	0.3	3	1

```
50         OBST_REMOVE :11
50         OBST_REMOVE :3
50         VEH_DEATH   :27
51.8364    WP_VISIT    :1
66.2705    WP_VISIT    :26
80         OBST_REMOVE :9
91.4406    WP_VISIT    :25
110        OBST_REMOVE :3
113.7254   WP_VISIT    :2
125.675    WP_VISIT    :18
149.4455   WP_VISIT    :16
161.2357   WP_VISIT    :5
165        OBST_REMOVE :8
165        WP_UPDATE   :21
165        WP_UPDATE   :22
165        WP_UPDATE   :23
171.6109   WP_VISIT    :3
174.4486   WP_VISIT    :21
179        OBST_REMOVE :3
184.7815   WP_VISIT    :6
185.3857   WP_VISIT    :4
185.5706   WP_VISIT    :14
186.2367   WP_VISIT    :19
188.7877   WP_VISIT    :15
201.23     WP_VISIT    :20
201.7878   WP_VISIT    :23
202.8183   WP_VISIT    :22
219.0244   WP_VISIT    :8
224.2813   WP_VISIT    :24
232.8244   WP_VISIT    :13
239.6571   WP_VISIT    :12
240.2706   WP_VISIT    :11
240.6586   WP_VISIT    :17
272.3756   WP_VISIT    :10
```

Figure 5-4: GUI.

add new targets or obstacles, which would trigger re-assignment. Fig. 5-4 shows an scenario with 8 vehicles, 26 targets, 5 obstacles and 4 SAMs, with timing constraints. Moreover, a couple of "events" (or dynamic changes) are defined in the scenario using the GUI as well. The GUI has greatly facilitated the procedure of setting up such a complex scenario.

5.2 Experiment Result

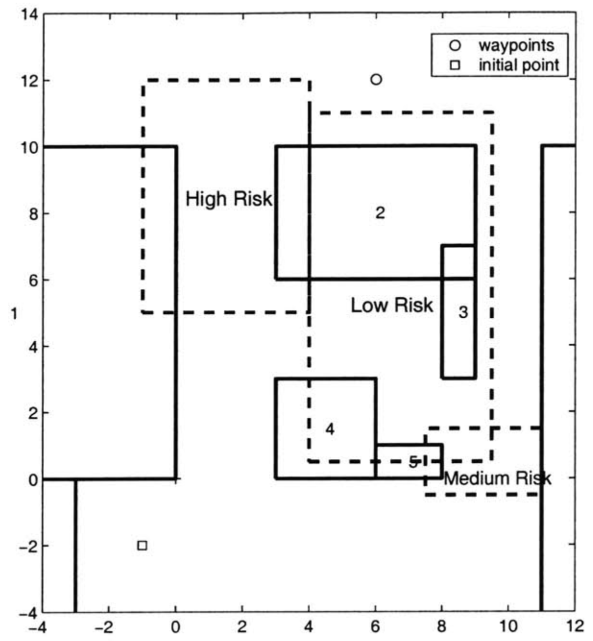
This section presents the results of hardware experiment with the ARSP algorithm (Chapter 3) implemented using the rover testbed described in the previous sections. The scenario is shown in Fig. 5-5(a). The rover starts at the bottom left corner. The highly constrained area consists of a number of obstacles and regions of different level of uncertainty. The goal is located at the top of the figure. Loosely speaking, the rover has three path options to reach the goal. The shortest path involves the maximum risk since it goes through the region of high uncertainty (Fig. 5-5(b)). The longest path involves the minimum risk since it passes through the region with low uncertainty (Fig. 5-5(d)). The third path has a medium distance and uncertainty compared with the other two choices (Fig. 5-5(c)).

By choosing different value for the parameter α, the path planner has a different level of concern about the uncertainty in the path. In Fig. 5-5(b) – Fig. 5-5(d), α is chosen to be 0, 0.1 and 0.5 respectively. The nominal solution ($\alpha = 0$) just picks the path of minimum distance (or minimum traveling time) and neglects any uncertainty involved. As we would expect, as the value of α increases, the path planner puts more concern on reducing the uncertainty in the path. Hence, for $\alpha = 0.5$, the rover takes the path of longest distance with minimum uncertainty.

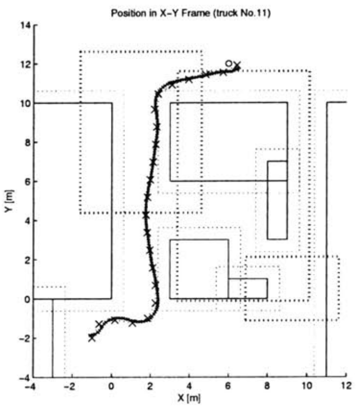
This experiment demonstrates the successful implementation of the ARSP algorithm into the RH-MILP framework and integration with the hardware testbed. The behavior of the rover with different value of α is consistent with what we would expect.

5.3 Conclusions

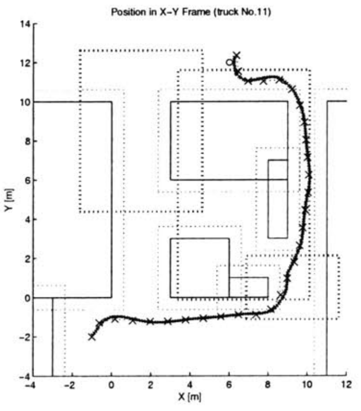
This chapter describes the rover testbed system used in the Aerospace Controls Lab., MIT. Moreover, a GUI tool is also described which help to set up more complex and more dynamic scenario for the purpose of testing with both simulation and hardware experiment. An experiment result with the rover testbed demonstrated successful implementation of the ARSP algorithm.



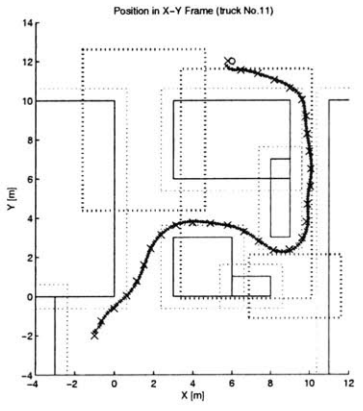
(a) The Scenario.



(b) $\alpha = 0$



(c) $\alpha = 0.1$



(d) $\alpha = 0.5$

Figure 5-5: Hardware experiment with different values of α

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis addresses the impact of environment dynamics and uncertainties on multi-UAV planning problem. Several algorithms were developed to modify the RH-MILP planning hierarchy to handle these changes and uncertainties. The thesis also demonstrated simulation results and hardware implementation of these algorithms.

Chapter 2 discussed the impact of dynamic changes in the environment on the computation time of RHTA. The re-computation of the cost map was identified as the limiting factor of speeding up the algorithm. An incremental method of updating the visibility graph during replan was presented. It successfully reduced the computation time for replan by 50%, as shown in a specific simulation result.

Chapter 3 presented a robust shortest algorithm, ARSP, for handling arc uncertainty. The algorithm is a modification of the Dijkstra's algorithm, so it is readily implemented in the RH-MILP formulation. The key benefit of ARSP over other robust algorithms [10, 11] is its low computational complexity. Moreover, a tuning knob adds flexibility, enabling the designer to adjust the level of risk acceptance in the planning.

Chapter 4 presented a planning algorithm which combines the search and track actions of UAVs. The algorithm extends the work in Refs. [32, 33, 34, 35] by incorporating the track mission for moving targets. A team of UAVs search an unknown region cooperatively and track the moving targets they have found. The algorithm has successfully included different mission objectives. In the presence of moving targets, the algorithm is able to converge the uncertainty to a steady-state value, even when there are more moving targets than UAVs.

Chapter 5 described the rover testbed developed in the Aerospace Controls Lab., MIT. Moreover, a GUI tool has been developed to facilitate creating complex and dynamic scenario for testing. An experiment result showed a successful implementation of the ARSP algorithm.

6.2 Future Work

In reality, UAV mission is uncertain and dynamic. The work in this thesis presented several algorithms to modify the RH-MILP planning hierarchy to handle uncertainty and dynamics. There are further improvements possible for them.

For UAV problem, the ARSP primarily incorporates the uncertainty as a result of variation in speed, hence traveling time. Other types of uncertainty can be addressed, for instance, the risk for flying over a SAM site. Here the risk addresses the compromise between traveling time and probability of being destroyed by the SAM. Furthermore, ambiguity in target and obstacle position leads to uncertainty in the problem as well. The uncertainty in obstacle position further addresses a question in robust constraints satisfaction.

The search & track algorithm serves as an information collecting stage for the UAV planning. It is generally done by the *reconnaissance* vehicles. The information is then sent to the *strike* vehicles for task assignment. Hence, further work will be required to combine the planning of these two types of UAVs. Moreover, in the presence of moving targets, the environment is rapidly changing. The incremental algorithm can be included to reduce the computational load during replanning.

Bibliography

[1] "Unmanned Aerial Vehicles Roadmap", *Office of the Secretary of Defense*, Dec 2002.

[2] P. R. Chandler, M. Pachter, D. Swaroop, J. M. Fowler, J. K. Howlett, S. Rasmussen,C. Schumacher, K. Nygard, "Complexity in UAV Cooperative Control", *Proceedings of the American Control Conference*, May 2002.

[3] D. Gillen, D. Jacques, "Cooperative Behavior Schemes for Improving the Effectiveness of Autonomous Wide Area Search Munitions", *Workshop on Cooperative Control and Optimization*, Dec, 2000.

[4] M. Pachter, P. Chandler, "Challengers of Autonomous Control", *IEEE Control Systems Magazine*, April, 1998.

[5] D. Jacques, R. Leblanc, "Effectiveness Analysis for Wide Area Search Munition", *Proceedings of the AIAA Missile Science Conference*, Nov 1998.

[6] D. Hristu, K. Morgansen, "Limited Communication Control", *Systems & Control Letters*, 1999.

[7] J. Bellingham, Y. Kuwata, and J. How, "Stable Receding Horizon Trajectory Control for Complex Environments", *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, August 2003.

[8] A. Richards, J. Bellingham, M. Tillerson, J. How, "Coordination and Control of Multiple UAV's", *Poceedings of the AIAA Guidance, Navigation and Control Conference*, Aug 2002.

[9] J. Bellingham, M. Tillerson, A. Richards, J. How, "Multi-Task Allocation and Path Planning for Cooperating UAV's", *Second Annual Conference on Cooperative Control and Optimization*, Nov 2001.

[10] D. Bertsimas, M. Sim, "Robust discrete optimization and network flows", *Operations Research Letters*, 2002.

[11] O. E. Karasan, M. C. Pmar, H. Yaman, "The Robust Shortest Path Problem with Interval Data", *Operations Research Letters*, 2003.

[12] P. Kouvelis, G. Yu, "Robust Discrete Optimization and Its Applications", *Kluwer Academic Publishers*, 1996.

[13] Y. Kuwata, "Real-time Trajectory Design for Unmanned Aerial Vehicles using Receding Horizon Control", *Master's Thesis, Massachusetts Institute of Technology*, June 2003.

[14] A. G. Richards, "Trajectory Optimization using Mixed-Integer Linear Programming", *Masters thesis, Massachusetts Institute of Technology*, June 2002.

[15] M. Alighanbari, "Task Assignment Algorithms for Teams of UAVs in Dynamic Environments", *Masters thesis, Massachusetts Institute of Technology*, June 2004.

[16] A. Richards, T. Schouwenaars, J. How, and E. Feron, "Spacecraft Trajectory Planning With Collision and Plume Avoidance Using Mixed-Integer Linear Programming", *Journal of Guidance, Control and Dynamics*, vol. 25, pp. 755V764, Aug 2002.

[17] A. Richards and J. P. How, "Aircraft Trajectory Planning With Collision Avoidance Using Mixed Integer Linear Programming", *Proceedings of the American Control Conference*, May 2002.

[18] V. V. Petrov, "Limits Theorems of Probability Theory: Independent Random Variables", *Oxford*, 1995.

[19] R. J. Larsen, M. L. Marx, "An Introduction to Mathematical Statistics and Its Applications", *Prentice Hall*, 1986.

[20] D. C. Montgomery, G. C. Runger, "Applied Statistics and Probability for Engineers", *John Wiley & Sons, Inc.*, 1999.

[21] D. P. Bertsekas, J. N. Tsitsiklis, "Introduction to Probability", *Notes for Class 6.041/6.431, Massachusetts Institute of Technology*, 2002.

[22] ILOG, *ILOG CPLEX Users guide*, 1999.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, "Introduction to Algorithms", *MIT Press*, 2nd Ed., 2001.

[24] H. Markowitz. "Portfolio Selection", *The Journal of Finance*, (7) No. 1, pp. 77-91, 1952.

[25] E. King, M. Alighanbari, Y. Kuwata, and J. How, "Coordination and Control Experiments on a Multi-vehicle Testbed", to appear at the *IEEE American Control Conference*, 2004.

[26] L. S. Buriol, M. G. C. Resende, M. Thorup, "Speeding Up Dynamic Shortest Path Algorithms", *AT&T Labs Research Technical Report TD-5RJ8B*, 2003.

[27] S. Koenig and M. Likhachev, "Incremental A^*", *Advances in Neural Information Processing Systems 14*, 2001.

[28] S. Koenig, M. Likhachev and D. Furcy, "Lifelong Planning A^*", *Technical Report, GIT-COGSCI-2002/2, College of Computing, Georgia Institute of Technology*, 2001.

[29] L. F. Bertuccelli, "Robust Planning for Heterogeneous UAVs in Uncertain Environments", *Master's Thesis, Massachusetts Institute of Technology*, 2004.

[30] ActivMedia Robotics, "Pioneer Operations Manual", January 2003. *http://robots.activmedia.com*

[31] ArcSecond, “Constellation 3D-i Error Budget and Specifications”, June, 2002. *http://www.arcsecond.com*

[32] M. M. Polycarpou, Y. Yang, K. M. Passino, “Cooperative Control of Distributed Multi-Agent Systems”, *IEEE Control Systems Magazine*, June, 2001.

[33] M. Flint, M. Polycarpou, E. Fernández-Gaucherand, “Cooperative Path-Planning for Autonomous Vehicles Using Dynamic Programming”, *IFAC*, 2002.

[34] Y. Jin, A. A. Minai, M. M. Polycarpou, “Cooperative Real-time Search and Task Allocation in UAV Teams”, *Proc. Conference on Decision and Control*, 2003.

[35] Y. Yang, A. A. Minai, M. M. Polycarpou, “Decentralized Cooperative Search in UAV's Using Opportunistic Learning”, *Proc. AIAA Guidance, Navigation and Control Conference*, 2002.

[36] J. Lee, R. Huang, A. Vaughn, X. Xiao, J. K. Hedrick, M. Zennaro, R. Sengupta, “Strategies of Path-Planning for a UAV to Track a Ground Vehicle”, *AINS*, 2003.

4534-44