A Priori and On-line Route Optimization for Unmanned Underwater Vehicles

by

Brian A. Crimmel

B.S., United States Coast Guard Academy (2007)

Submitted to the Sloan School of Management
in partial fulfillment of the requirements for the degree of

Masters of Science in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2012

Author .... [illegible]
Operations Research Center
May 18, 2012

Certified by ....
Eric D. Nelson
Senior Member of the Technical Staff
Autonomous Vehicle Systems
Charles Stark Draper Laboratory
Technical Supervisor

Certified by .... [illegible] ......
Professor Patrick Jaillet
Dugald C. Jackson Professor
Department of Electrical Engineering and Computer Science
Co-Director, Operations Research Center
Thesis Supervisor

Accepted by ....
Professor Dimitris J. Bertsimas
Boeing Professor of Operations Research
Co-Director, Operations Research Center

A Priori and On-line Route Optimization for Unmanned Underwater Vehicles

by

Brian A. Crimmel

Submitted to the Sloan School of Management
on May 18, 2012, in partial fulfillment of the
requirements for the degree of
Masters of Science in Operations Research

Abstract

The U.S. military considers Unmanned Underwater Vehicles (UUVs) a critical component of the future for two primary reasons — they are effective force multipliers and a significant risk-reducing agent. As the military's technology improves and UUVs become a reliable mission asset, the vehicle's ability to make intelligent decisions will be crucial to future operations.

The thesis develops various algorithms to solve the UUV Mission-Planning Problem (UUV-MPP), where the UUV must choose which tasks to perform in which sequence in a stochastic mission environment. The objective is to find the most profitable way to execute tasks with restrictions of total mission time, energy, time-restricted areas, and weather conditions. Since the UUV accumulates navigation error over time while maneuvering underwater, the UUV must occasionally halt operations to re-orient itself via a navigation fix. While a navigation fix takes time and increases the likelihood of exposing the vehicle's position to potential adversaries, a reduction in navigation error allows the UUV to perform tasks and navigate with a greater amount of certainty. The algorithms presented in this thesis successfully incorporate navigation fixes into the mission-planning process.

The thesis considers Mixed-Integer Programming, Exact Dynamic Programming, and an Approximate Dynamic Programming technique known as Rollout to determine the optimal *a priori* route that meets operational constraints with a specified probability. The thesis then shows how these formulations can solve and re-solve the UUVMPP on-line. In particular, the Rollout Algorithm finds task route solutions on average 96% of the optimal solution *a priori* and 98% of the optimal solution on-line compared to exact algorithms; with a significant reduction in computation run time, the Rollout Algorithm permits the solving of increasingly complex mission scenarios.

Technical Supervisor: Eric D. Nelson
Title: Senior Member of the Technical Staff
Autonomous Vehicle Systems
Charles Stark Draper Laboratory

Thesis Supervisor: Professor Patrick Jaillet
Title: Dugald C. Jackson Professor
Department of Electrical Engineering and Computer Science
Co-Director, Operations Research Center

Acknowledgments

This thesis would not have been possible without the support, love, and commitment offered to me during my time at MIT. I would like to thank...

- The U.S. Coast Guard, who afforded me the opportunity to take two years to pursue a degree which met my personal and professional development goals.
- Draper Laboratory, who provided the resources and exceptional people to create an outstanding research environment.
- My technical supervisor at Draper Laboratory, Eric Nelson, who provided hundreds of hours of help and guidance to develop my thesis simulations and report. Eric, thank you in particular for all the recommendations, edits, and fixes you made to the software that I wouldn't have been able to do myself. Your technical acumen and personable demeanor made it a pleasurable experience.
- My thesis and academic advisor at MIT, Patrick Jaillet, who gave substantial feedback and insight to my many proposed formulations. Dr. Jaillet, thank you for taking the time to meet regularly with me to ensure I followed some semblance of a rational plan.
- Jacob Cates, who spent countless hours of his own time helping me implement my first formulations and giving me feedback. Jake, thank you for making the transition from your thesis to mine as smooth as possible.
- My wonderful family, in particular Mom, Dad, and Nathan, whose strength and encouragement has been greatly appreciated throughout my life. Thank you for making the many road trips up here the last nine years.
- My beautiful girlfriend, Kaity, who provided support and love that proved invaluable during my time here. Kaity, you have made this experience a thousand times better than it would have been without you!
- My friends, especially my ORC classmates, Draper buddies, and MIT hockey teammates, who kept me grounded and sane throughout these hectic two years. I hope to work with you again in the future, whether in the military or in the "real" world.

It is a pleasure to have worked with many amazing people the last two years. I hope the next phase of my life is as enjoyable as the last two years have been.

Brian A. Crimmel, Lieutenant, U.S. Coast Guard
May 18, 2012

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

List of Tables

Chapter 1

Introduction

With the development of autonomous system technology, unmanned vehicles are capable of performing missions across the spectrum of military operations. According to The Navy Unmanned Undersea Vehicle Master Plan [15], the U.S. military considers unmanned vehicles a critical component of our future military for two primary reasons — they are effective force multipliers, and they are a significant risk reducing agent. As force multipliers, unmanned vehicles have the potential to perform similar or simplified tasks a manned craft could perform at a fraction of the cost. Furthermore, they can perform tasks that present great danger to manned craft, such as entering a waterway shortly after a nuclear attack, which would put personnel at an unacceptable level of risk.

The thesis focuses on the mission-planning algorithm for Unmanned Underwater Vehicles (UUVs) — vehicles that are able to operate underwater without a human occupant — termed as the Unmanned Underwater Vehicle Mission-Planning Problem (UUVMPP). The goal of the UUVMPP is to determine which tasks to perform in which sequence in a stochastic, or uncertain, mission environment. The Navy UUV Master Plan [15] expands on the nine high-priority UUV tasks, listed as follows:

- Intelligence, Surveillance, and Reconnaissance (ISR)
- Mine Countermeasures (MCM)
- Anti-Submarine Warfare (ASW)

- Inspection / Identification
- Oceanography
- Communication / Navigation Network Node (CN3)
- Payload Delivery
- Information Operations (IO)
- Time Critical Strike (TCS)

Since UUVs operate in an uncertain environment, plans that rely on estimated values can quickly become infeasible or suboptimal after a small change in operational parameters. The thesis develops formulations that incorporate stochasticity into the models which allows the mission planner to produce mathematically robust solutions, *i.e.* solutions that remain feasible and near optimal in a variety of situations.

The thesis considers Mixed-Integer Programming (MIP) formulations, Exact Dynamic Programming, and an Approximate Dynamic Programming technique known as the Rollout Algorithm to determine the optimal *a priori* UUV route for a given scenario, where *a priori* means solving prior to the start of the mission. The work then shows how these formulations can solve and re-solve the UUVMPP on-line, *i.e.* during the mission.

1.1 Motivation

UUVs are extremely limited in their ability to communicate to other personnel, both on the surface and underwater; as a result, human interaction is often not feasible and therefore a high level of autonomy must be achieved in the mission-planning process.

The goal of the UUVMPP is to find the task sequence that maximizes and/or minimizes some mission objective(s), such as stealth, risk, or number of tasks completed. In developing the best UUV route, the mission planner has to consider the limited resources onboard a UUV, most importantly mission time and energy capacity. In addition, since the UUV accumulates navigation error over time while maneuvering underwater, the UUV must occasionally halt operations to re-orient itself via a navigation fix to obtain updated positioning information. While a navigation fix takes time and increases the likelihood of exposing its position to potential adversaries, a reduction in navigation error allows the UUV to perform tasks and navigate with a greater amount of certainty. The algorithms presented in the thesis successfully incorporate navigation fixes into mission plans.

A UUV's movement through the mission environment consists of two actions: (1) **Travel**, the elapsed time it takes the UUV to travel from one latitude/longitude coordinate to another, and (2) **Task Execution**, the elapsed time it takes the UUV to complete a given task. The UUVMPP can potentially have time-varying travel times between two tasks and task execution times due to tides, currents, oceanic traffic, time-dependent no-travel zones, etc. Furthermore, travel times and task execution times may depend on the amount of navigation error the UUV has accumulated; *e.g.* if the UUV is tasked with completing an oceanographic survey of a particular area, the UUV may take longer to identify the region with reduced positional information. The thesis builds models that properly account for these factors.

Even if the mission planner develops a high-quality, robust *a priori* task route for the UUV, there is a chance the UUV's plan becomes obsolete during the mission when problem parameters are realized. The ability for the mission planner to know when and how to plan and re-plan on-line is vital to the UUVMPP. The research presents algorithms capable of solving the UUVMPP on-line and demonstrates its performance in a variety of realistic mission scenarios.

1.2 Outline

Chapter 2 formalizes the UUVMPP and provides a detailed analysis of the mathematical concepts presented in the thesis, including a discussion of relevant literature.

Chapter 3 introduces a Mixed-Integer Programming (MIP) formulation that accounts for time, energy, and navigation error to find an *a priori* task route solution for a given mission scenario. A second formulation with the Route Alteration Graph demonstrates a way the mission planner can develop a task route solution when the UUV encounters unlikely or unexpected situations. The chapter then walks through eight unique mission scenarios to illustrate the flexibility of the model. Chapter 3 ends with a run time analysis and characterization of optimal solutions with exact formulations and suggested heuristics.

Chapter 4 presents an Exact Dynamic Programming algorithm referred to as the Brute Force Method which formulates the UUVMPP as a Stochastic Shortest Path problem. The chapter also outlines an Approximate Dynamic Programming algorithm known as the Rollout Algorithm to show how an acceptable heuristic can be sequentially improved upon in an *a priori* or on-line setting. A run time analysis and characterization of optimal solutions for all methods presented in the thesis completes Chapter 4.

Chapter 5 argues the usefulness of the proposed formulations and whether they should be considered for application onboard UUVs. A discussion of possible future work concludes the thesis.

Chapter 2

Background

In the Unmanned Underwater Vehicle Mission-Planning Problem (UUVMPP), a basic scenario consists of:

- Start Mission Location: the latitude/longitude coordinate where the UUV begins its mission.
- Tasks: types of actions the UUV can perform at a particular map coordinate.
- Safe Navigation Locations: the coordinates the mission planner identifies as inherently safe for the UUV to take a navigation fix during the mission.
- End Mission Location: the coordinate where the UUV concludes its mission.

This chapter provides an overview of the UUVMPP, including a summary of mathematical techniques and a review of applicable literature.

2.1 Problem Overview

2.1.1 Traveling Salesman Problem

If the UUV is given unlimited resources to complete every available task, the mission planner chooses the task route that minimizes resource consumption. This problem closely resembles one of the most famous problems in discrete optimization, the Traveling Salesman Problem (TSP). A TSP is described by an *undirected graph* $G = (N, E)$ which consists of a set N of *nodes* and a set E of undirected *arcs* or *edges*, where an edge e is an *unordered pair* of distinct nodes, *i.e.* a two-element subset $\{i, j\}$ of N. Given the undirected graph $G = (N, E)$ and costs c_e for every edge $e \in E$, the objective of the TSP is to find a *tour* — a cycle that visits all nodes — of minimum cost [9]. Since the UUV has a distinct start and end mission location, to model the UUVMPP a *directed graph* $G = (N, A)$ replaces the directed graph above, where A represents a set of directed arcs, *i.e.* an *ordered pair* $\{i, j\}$ of distinct nodes.

2.1.2 Prize Collecting Traveling Salesman Problem

In a realistic mission, a UUV has various restrictions the mission planner must take into account. The UUV receives strict mission termination guidelines since it typically rendezvouses with a manned craft for retrieval, which leads to a time restriction on each mission denoted T_{MAX}. The UUV has a finite amount of energy at its disposal labeled E_{MAX}. The UUV must occasionally update its positioning information to maneuver in certain locations or perform specific tasks; therefore, the mission planner introduces a maximum allowable accumulated navigation error for the UUV's mission N_{MAX}. Due to these operational limitations, the mission planner often cannot execute all available tasks and must choose which tasks to complete to provide the greatest mission utility. This constrained combinatorial optimization problem resembles a Traveling Salesman Problem with Profits, otherwise known as a Prize Collecting TSP.

Prize Collecting TSPs can be viewed as bicriteria TSPs with two opposite objectives, one inciting the salesman to travel (and collect profit) and the other pushing the salesman to minimize travel costs. Most researchers address the Prize Collecting TSPs as a single-criterion version, where either one objective is constrained with a specified bound or the two objectives are weighted and combined linearly [12]. This thesis formulates the operational limitations as constraints which the UUV cannot exceed, where the ultimate goal is to find the task sequence that maximizes or minimizes some mission objective.

2.1.3 Vehicle Routing Problems with Time Windows

Since a UUV operates in a waterborne environment, there are numerous factors that affect a UUV's travel between tasks and dwell time at each task, including tides, currents, obstacles, and weather. A mission scenario can also contain time-varying obstacles known as time-dependent no-go zones; a shipping lane that is only active during daytime hours and a fishing area with heavy traffic during regular time intervals are two common examples. For these reasons, the UUVMPP must allow for problem parameters that depend on elapsed mission time.

Vehicle Routing Problems (VRP) with Time Windows explicitly address problems that contain time-varying problem parameters. Braysy and Gendreau [10] present a survey of the research in this subject area, which focuses on designing least cost routes from one depot to a set of geographically scattered customers with time-dependent service requirements. Bent and Hentenryck [3] describe a two-stage hybrid algorithm for a transportation problem consisting of service locations and time windows. Both [10] and [3] motivate the development of heuristics suggested in this thesis.

Due to the operational constraints and the time-varying problem parameters, the UUV-MPP can best be described as a Prize Collecting Traveling Salesman Problem with Time Windows.

2.1.4 Integer Knapsack Problem

In its most basic form, the UUVMPP is a *zero-one* (or *binary*) *integer programming* problem — tasks can only be completed 0 or 1 times and the mission planner must decide which tasks to complete to maximize the utility of the mission. The structure of the UUVMPP is similar to the Integer Knapsack Problem, where one must choose the items to place into a knapsack to maximize the value of the items while not exceeding available space or weight [9].

In the Integer Knapsack Problem, a person is given n items. The jth item has weight w_j and its value is c_j. Given a bound K on the weight that can be carried in a knapsack, a person would like to select items to maximize the total value. Let x_j represent the binary decision variable for each item, where $x_j = 1$ if the jth item is chosen and 0 otherwise. The formulation is as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} w_j x_j \leq K \\ & x_j \in \{0,1\}, \qquad j = 1, 2, \ldots, n. \end{aligned}$$

The Integer Knapsack Problem is analogous to the UUVMPP (where tasks are "items" and operational constraints are "weights") with one major exception — the Integer Knapsack Problem ignores the combinatorial complexity and assumes the sequence of item placement has no impact on the problem (*i.e.* placing item i then j in the knapsack is equivalent to placing item j then i). However, due to the time-varying problem parameters in the UUV-MPP, the mission planner must consider sequencing in developing the final route solution. The intuition for the heuristic found in Chapter 4 comes from a common greedy heuristic used to solve the Integer Knapsack Problem, the Nearest Neighbor algorithm.

2.1.5 Cates Method

Cates [11] models the UUVMPP as a Prize Collecting Traveling Salesman Problem with Time Windows and proposes various exact and heuristic formulations. The Mixed-Integer Programming (MIP) formulations presented in Chapter 3 extends Cates' work to account for the impact of navigation error on the UUV's mission, including when and how to execute navigation fixes. Cates introduces the Beta (β) constraint which guarantees a task route maintains an appropriate confidence level for satisfying operational limitations; this thesis utilizes the Beta constraint to develop robust solutions.

2.2 Network Flow Optimization

Vehicle Routing Problems are commonly formulated as Network Flow Problems due to the variety of general purpose solving methods available [9]. A *network* is a directed graph $G = (N, A)$ defined by a node set N and an arc set A. The formulation establishes linear constraints for the arcs and nodes which restricts the flow through the network. Each node $i \in N$ has some external *supply* denoted d_i; in particular, node i is called a *source* if $d_i > 0$ and a *sink* if $d_i < 0$. Each arc $(i, j) \in A$ has an associated flow variable x_{ij} that represents the amount of flow (*e.g.* UUV travel) through the arc. The formulation establishes the flow conservation law — *i.e.* the amount of flow into node i must equal the total flow out of node i — with the following constraint for each node:

$$\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = d_i, \quad \forall\, i \in N$$

The flow constraints can be concisely represented in matrix form as:

$$\mathbf{Bx} = \mathbf{d}$$

where $\mathbf{B}$ is the *node-arc incidence matrix* and $\mathbf{d}$ is the *demand vector.* In the node-arc incidence matrix, each row corresponds to a node and each column corresponds to an arc. The (i, j)th entry of matrix $\mathbf{B}$, denoted $\mathbf{B}^{(i,j)}$, is defined as follows:

$$\mathbf{B}^{(i,j)} = \begin{cases} 1, & \text{if } i \text{ is the start node of the } j\text{th arc} \\ -1, & \text{if } i \text{ is the end node of the } j\text{th arc} \\ 0, & \text{otherwise} \end{cases}$$

In a Vehicle Routing Problem, one unit of flow must travel from the node representing the start mission location (S in the UUVMPP) to the node representing the end mission location (E). Therefore, $\mathbf{d}$ is defined as $d_S = 1$, $d_E = -1$, and $d_i = 0$ for all other nodes. A more detailed analysis of network flow models and its application to Vehicle Routing Problems can be found in [1].

2.3 Dynamic Programming

Dynamic Programming (DP) decomposes a problem into a sequence of stages, where decisions are made at each stage. In the Vehicle Routing Problem, a stage is analogous to how many service locations (or tasks) have been visited in a given mission. The key benefit of Dynamic Programming is that the algorithm provides the optimal decision for every situation the mission planner could potentially encounter.

The basic DP model has two principal features: (1) an underlying discrete-time dynamic system, and (2) a cost function that is additive over time [4]. The evolution of the system depends on the decisions made at discrete instances of time, and is of the form:

$$x_{k+1} = f_k(x_k, u_k, w_k), \qquad k = 0, 1, \ldots, N-1$$

where:

- k indexes discrete time,
- x_k is the state of the system which summarizes past information relevant for future optimization,
- u_k is the control or decision to be chosen at time k,
- w_k is a random parameter, often the disturbance or noise in the system,
- N is the maximum number of stages in the system, and
- f_k is a function that describes how the system updates the state information.

Letting $g_k(x_k, u_k, w_k)$ represent the cost incurred at time k, the total cost of the dynamic system is:

$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)$$

where $g_N(x_N)$ is the terminal cost, or the cost incurred at the end of the process. Due to the randomness in the system w_k, the total cost is often expressed as an expectation.

The additive cost function summarizes the dynamic system in a simple recursive algorithm that relies on the *Principle of Optimality* to find the globally optimal solution. The Principle of Optimality is defined as follows [4]:

Definition. *Given some N-stage Dynamic Program, let* $\pi^* = \{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\}$ *be an optimal policy for the basic problem. Assume that when using* π^**, a given state* x_i *occurs at time* i *with positive probability. Consider the subproblem whereby we are at* x_i *at time* i *and wish to minimize the "cost-to-go" from time* i *to time* N

$$E\big[g_N(x_N) + \sum_{k=i}^{N-1} g_k(x_k, \mu(x_k), w_k)\big]$$

Then the truncated policy $\{\mu_i^*, \mu_{i+1}^*, \ldots, \mu_{N-1}^*\}$ *is optimal for this subproblem.*

The intuition behind the principle of optimality is that if the policy from state x_i were not optimal, then the controller would be able to further improve the solution by using the optimal policy once the system reaches x_i.

To better interpret this fundamental principle, consider the following analogy. Suppose the fastest driving route from New York City, New York to Boston, Massachusetts is known to be π^* and passes through Hartford, Connecticut, state i. The principle of optimality states that the leg from Hartford to Boston would also be the fastest route for a trip that began in Hartford and finishes in Boston, π_i^*. Figure 2-1 illustrates this example.

The principle of optimality implies that DP can construct an optimal policy in piecewise fashion. The algorithm first finds the optimal policy for the subproblem involving just the final stage. The algorithm then calculates the optimal policy for the subproblem involving the last two stages, continuing in this fashion until the optimal policy for the entire N-stage

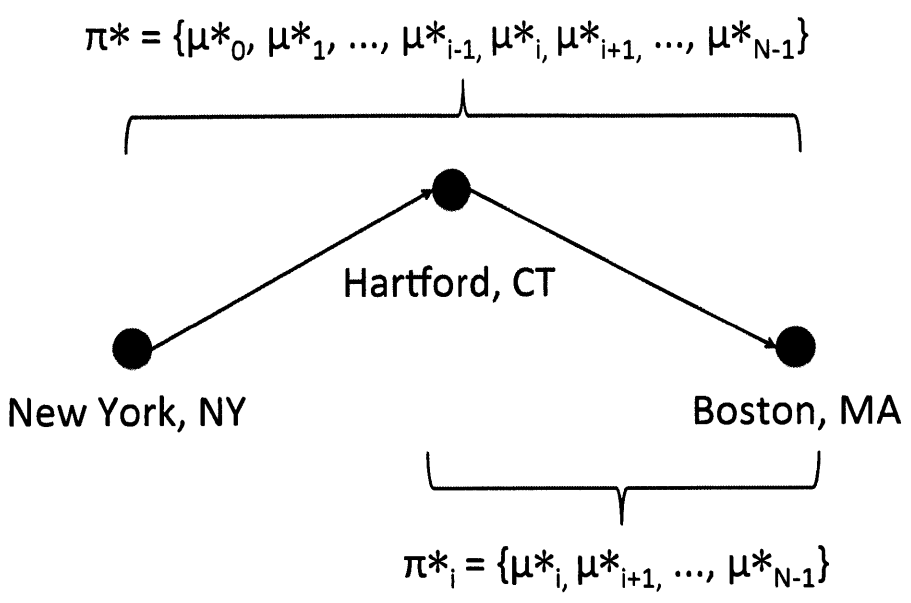

$\pi^* = \{\mu^*_0, \mu^*_1, \ldots, \mu^*_{i-1}, \mu^*_i, \mu^*_{i+1}, \ldots, \mu^*_{N-1}\}$
Hartford, CT
New York, NY
Boston, MA
$\pi^*_i = \{\mu^*_i, \mu^*_{i+1}, \ldots, \mu^*_{N-1}\}$

Figure 2-1: Principle Of Optimality Example

problem is determined. DP proceeds sequentially by solving all subproblems of a given time length, using the solution to the subproblems of shorter time length.

The main drawback of DP is its computational intractability. DP has an exponential increase in required computation as the problem's size increases, referred to as the "curse of dimensionality." Therefore, Approximate Dynamic Programming methods must be considered to solve problems of meaningful size. Secomandi's [13] and [14] demonstrate how an Approximate DP method known as the Rollout Algorithm can sequentially improve the performance of a base heuristic in the Vehicle Routing Problem.

2.4 Summary

This chapter outlined the mathematical concepts present in the development of the UUV-MPP as well as various solving methods. With the goal of finding robust UUV task route solutions, Chapter 3 models the UUVMPP as a Network Optimization problem with Mixed-Integer Programming formulations.

Chapter 3

Route Optimization with Mixed-Integer Programming Methods

3.1 Introduction

In finding the optimal task route in a given mission scenario, the mission planner faces a combinatorial optimization problem that becomes increasingly difficult to solve as the number of tasks grows. The Unmanned Underwater Vehicle Mission-Planning Problem (UUVMPP) is a variation of the Traveling Salesman Problem (TSP) with constraints that potentially restrict the completion of every task; in addition, the distances between tasks and dwell time at each task vary with time and are not deterministic. This modified problem is a Prize Collecting Traveling Salesman Problem with Time Windows. The Mixed-Integer Programming (MIP) formulations presented in this chapter consider these constraints to determine the task route *a priori* that provides the greatest expected utility and maintains a certain probability of successfully completing the mission.

This chapter discusses the evolution of the network flow graphs, constraints, and objective function to appropriately model the UUVMPP. A walkthrough of realistic mission scenarios and solutions demonstrate the flexibility of the MIP formulations and show how the problem scales with increasingly complex problems.

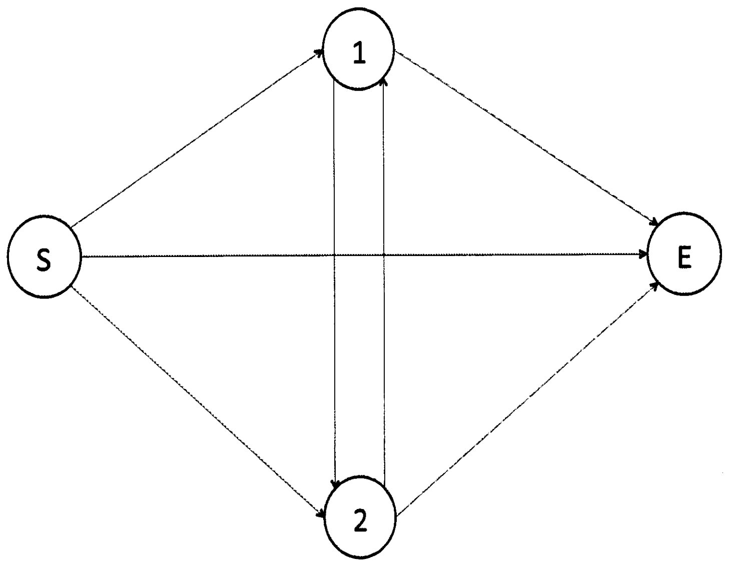


Figure 3-1: Basic Network Flow, Two-Task Example. Node S is the start mission location, node E is the end mission location, and nodes 1 and 2 represent tasks 1 and 2, respectively.

3.2 Network Flow Graphs

3.2.1 Basic Network Flow

The main objective of the UUVMPP is to find which tasks to perform, or not perform, and in which sequence to maximize the UUV's utility [11]. A basic mission scenario consists of a start mission location, a set of tasks, and an end mission location. This environment can be well-represented mathematically by a directed network flow model where a node represents each location and an arc connects each pair of nodes. The Basic Network Flow in Figure 3-1 illustrates a two-task example problem where node S is the start mission location, node E is the end mission location, and nodes 1 and 2 represent tasks 1 and 2, respectively. Let $\mathbf{L}$ represent the set of all available tasks, in this case $\{1, 2\}$.

Table 3.1 enumerates the five feasible *a priori* solutions to the Basic Network Flow with two tasks. A feasible solution represents a task sequence that reaches the end mission location and satisfies the given constraints; therefore, an infeasible solution occurs when the UUV cannot travel directly from the start location to the end location with sufficient mission resources, such as time or energy.

Solution Number	**Task Sequence**
1	$S \to E$
2	$S \to 1 \to E$
3	$S \to 2 \to E$
4	$S \to 1 \to 2 \to E$
5	$S \to 2 \to 1 \to E$

Table 3.1: Feasible UUVMPP Solutions with Basic Network Flow, Two-Task Example

The number of feasible solutions in the Basic Network Flow model is strictly dependent on the number of tasks in the mission scenario, n:

$$\sum_{k=0}^{n} \frac{n!}{k!} \tag{3.1}$$

There are two important assumptions with these formulations — each task can either be performed 0 or 1 times, and if the UUV travels to a task location it will execute that task with probability one. These formulations do not consider the possibility the UUV experiences a catastrophic failure of equipment, such as loss of propulsion or navigation system.

In Figure 3-1, the arcs represent both transit to the task locations and task execution. The Basic Network Flow is an oversimplified model in the UUVMPP since time and energy have a unique relationship when transiting versus executing a task. To correct this issue, consider Figure 3-2 where each task is split into two separate nodes. For each task $i \in \mathbf{L}$, node $i.0$ represents the vehicle being at the task prior to execution and node $i.1$ represents the vehicle being at the task after execution. When stochasticity is introduced into the model in Section 3.2.7, splitting the nodes in this fashion allows the MIP formulations to capture the variability in the travel times and task execution. The network flow model in Figure 3-2 is interpreted as follows:

- Node $S.1$ is the start location.

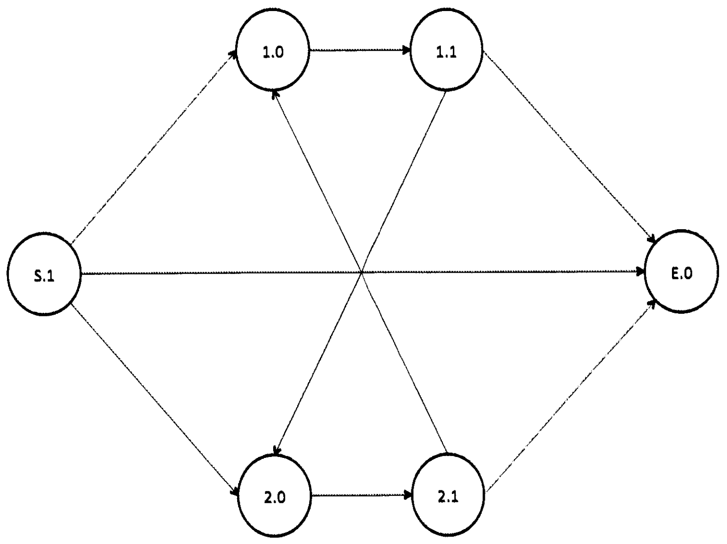


Figure 3-2: Basic Network Flow with Nodes Split, Two-Task Example. Node $S.1$ is the start location, node $E.0$ is the end location, nodes 1.0 and 2.0 represent being at task 1 and 2 **before** task execution, and nodes 1.1 and 2.1 represent being at task 1 and 2 **after** task execution. In Cates [11], the figure was termed the UUV Decision Graph.

- Node $E.0$ is the end location.
- Nodes 1.0 and 2.0 represent being at task 1 and 2 **before** task execution, respectively.
- Nodes 1.1 and 2.1 represent being at task 1 and 2 **after** task execution, respectively.
- Arcs $(i.1, j.0)$ represent transit from task i to task j.
- Arcs $(i.0, i.1)$ represent executing task i.

In Cates [11], Figure 3-2 was termed the UUV Decision Graph. However, with the incorporation of navigation fixes into the UUVMPP, Figure 3-2 does not illustrate the full set of decisions at the mission planner's disposal. The following section discusses various ways the network flow model can account for navigation fixes.

3.2.2 Modeling Navigation Fixes

As the UUV traverses underwater for long periods of time, the vehicle accumulates navigation error at some rate and becomes less confident in its position. The UUV occasionally

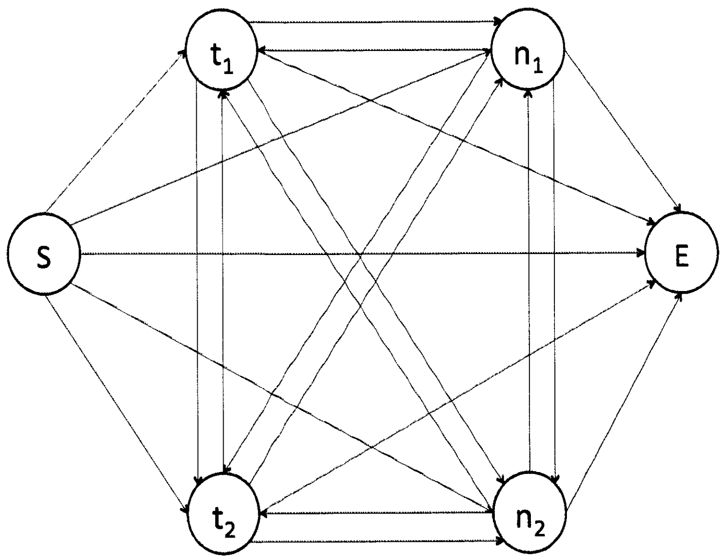


Figure 3-3: Modeling Navigation Fixes – Method 1. Nodes t_1 and t_2 represent the two tasks and nodes n_1 and n_2 represent the two safe navigation fix locations.

takes a navigation fix to re-orient and reset its accumulated navigation error. The Basic Network Flow can be altered to permit navigation fixes in the mission-planning process. The ultimate goal is to create a network flow model that illustrates all possible UUV task route combinations.

Method 1 – Nodes for All Navigation Fix Locations

Method 1 treats each navigation fix location as its own node in the Basic Network Flow. Let:

- $L_T = \{t_1, t_2, \ldots, t_n\}$ represent the set of n available tasks.
- $L_N = \{n_1, n_2, \ldots, n_p\}$ represent the set of p available navigation fix locations.

Figure 3-3 illustrates a two-task example with two navigation fix locations with Method 1.

Since navigation fixes can be visited an unlimited number of times, this formulation restricts arcs to the nodes $\in L_T$ to be binary (*i.e.* 0 or 1) while the arcs to the nodes $\in L_N$ have no restriction. Cycles are difficult to avoid in a formulation with these characteristics, and thus Method 1 is not a preferred network flow model to solve the UUVMPP.

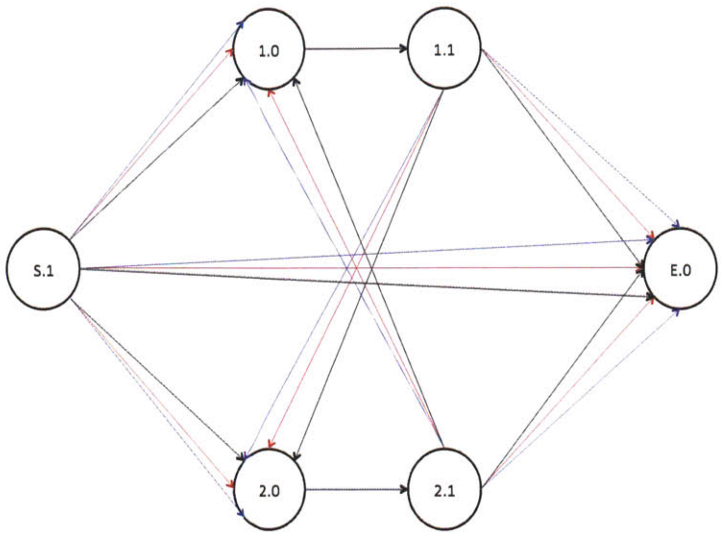


Figure 3-4: Modeling Navigation Fixes – Method 2. Black arcs of type $(i.1, j.0)$ represent direct transit from task i to task j, red arcs of type $(i.1, j.0)$ represent transit from task i to navigation fix location n_1 to task j, and blue arcs of type $(i.1, j.0)$ represent transit from task i to navigation fix location n_2 to task j.

Method 2 – Arcs for All Navigation Fix Locations

Method 2 depicts task locations as nodes (similar to the Basic Network Flow) and arcs are added to represent taking a navigation fix. Specifically, a UUV travel arc is added between every pair of tasks for each navigation fix location which indicates the mission planner's decision to take a navigation fix en route to the next task. Embedded in each fix arc ($i.1$, $j.0$) are the time and energy required to travel from task i to some pre-determined navigation fix location and then to task j. Figure 3-4 displays a two-task example with two navigation fix locations, where:

- Black arcs of type $(i.1, j.0)$ represent direct transit from task i to task j.
- Red arcs of type $(i.1, j.0)$ represent transit from task i to navigation fix location n_1 to task j.
- Blue arcs of type $(i.1, j.0)$ represent transit from task i to navigation fix location n_2 to task j.

If there are a large number of navigation fix locations, Method 2 adds significant complexity to the model due to the large increase in number of arcs. Between each pair of tasks, there is often a safe navigation fix location that intuitively makes the most sense (*e.g.* one location is closer than all the others) and all other fix arcs are unnecessary. Method 3 offers an improvement that takes this into consideration.

Method 3 – Arcs for a Subset of Navigation Fix Locations

Method 3 is identical to Method 2 except there are always two arcs between each pair of tasks:

- Red arcs of type $(i.1, j.0)$ represent direct transit from task i to task j.
- Blue arcs of type $(i.1, j.0)$ represent transit from task i to some navigation fix location to task j.

Method 3 requires additional calculations prior to the start of the mission to determine which safe navigation fix location is most prudent between each pair of tasks.

Method 3 is well-suited to portray the UUV's task route in the UUVMPP. The following section formalizes the construction of the network flow model in Method 3 referred to as the Decision Graph.

3.2.3 Decision Graph

The Decision Graph (Figure 3-5) allows the model to incorporate navigation fixes into the mission-planning process. The nodes have the same meaning as in the Basic Network Flow with Nodes Split (Figure 3-2), with arcs interpreted as follows:

- Red arcs $(i.1, j.0)$ represent direct transit from task i to task j **without** a navigation fix.
- Blue arcs $(i.1, j.0)$ represent transit from task i to task j **with** a navigation fix. Section 3.5.2 elaborates on how the model selects the navigation fix locations *a priori.*

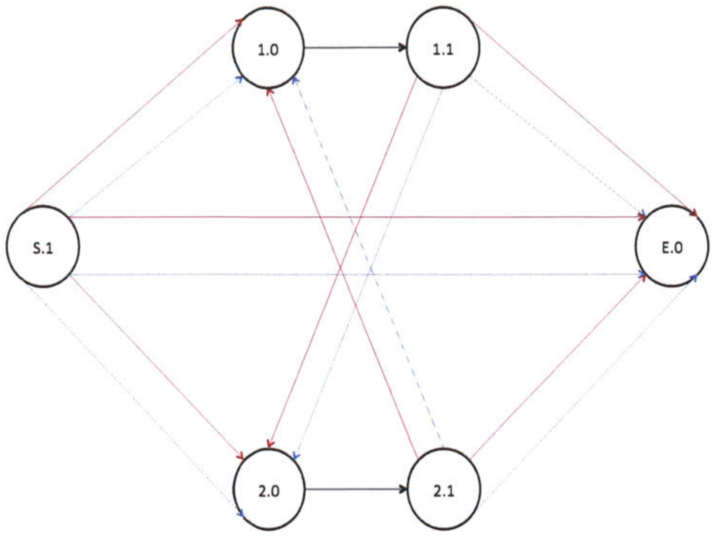


Figure 3-5: Decision Graph, Two-Task Example. Node $i.0$ represents being at the task prior to execution and node $i.1$ represents being at the task after execution. Red arcs represent direct travel **without** a fix, blue arcs represent travel **with** a fix, and black arcs represent task execution.

- Black arcs ($i.0$, $i.1$) represent executing task i.

The Decision Graph captures all possible routes for the UUV's mission. The ultimate goal of the UUVMPP is to decide how to maneuver through the mission environment represented by this network flow. The solutions to the MIP formulations presented in this chapter provide the task sequence in the Decision Graph that maximizes the overall mission's utility.

The addition of navigation fixes into the mission-planning process significantly increases the number of feasible solutions to the UUVMPP since the mission planner has to determine when to execute navigation fixes in addition to determining the optimal task sequence. From the Basic Network Flow model solutions found in Table 3.1, consider Solution 4, $S \rightarrow 1 \rightarrow 2 \rightarrow E$. By incorporating navigation fixes into the UUVMPP, the mission planner has eight separate ways to execute this particular task sequence, where **direct** represents traveling from one task to another without a navigation fix; see Table 3.2. In total, there are 20 feasible solutions in the two-task example problem modeled by the Decision Graph in Figure 3-5.

Solution Number	Possible Task Sequence
1	$S \xrightarrow{\text{direct}} 1 \xrightarrow{\text{direct}} 2 \xrightarrow{\text{direct}} E$
2	$S \xrightarrow{\text{direct}} 1 \xrightarrow{\text{direct}} 2 \xrightarrow{\text{fix}} E$
3	$S \xrightarrow{\text{direct}} 1 \xrightarrow{\text{fix}} 2 \xrightarrow{\text{direct}} E$
4	$S \xrightarrow{\text{direct}} 1 \xrightarrow{\text{fix}} 2 \xrightarrow{\text{fix}} E$
5	$S \xrightarrow{\text{fix}} 1 \xrightarrow{\text{direct}} 2 \xrightarrow{\text{direct}} E$
6	$S \xrightarrow{\text{fix}} 1 \xrightarrow{\text{direct}} 2 \xrightarrow{\text{fix}} E$
7	$S \xrightarrow{\text{fix}} 1 \xrightarrow{\text{fix}} 2 \xrightarrow{\text{direct}} E$
8	$S \xrightarrow{\text{fix}} 1 \xrightarrow{\text{fix}} 2 \xrightarrow{\text{fix}} E$

Table 3.2: Feasible UUVMPP Solutions with Decision Graph, Task Sequence $S \to 1 \to 2 \to E$

The number of feasible solutions in this model is again strictly dependent on n, but grows much faster than in the Basic Network Flow Model:

$$\sum_{k=0}^{n} \frac{n!}{k!} * 2^{k+1} \tag{3.2}$$

The Decision Graph is an appropriate model to solve the UUVMPP where the UUV's travel times and task execution times are constant and deterministic. The following model expands the Decision Graph to account for the time-changing mission parameters.

3.2.4 Time Expanded Deterministic Graph

While the Decision Graph serves as an acceptable way to view the task route solution, the Time Expanded Deterministic Graph (TEDG) presented in this section allows for a more realistic portrayal of the environment by accounting for tides, currents, weather, and time-dependent no-go zones. Time-dependent no-go zones can range from fishing areas with anticipatable traffic patterns to shipping lanes with greater activity during daytime hours.

The TEDG allows the mission planner to model both the travel times between tasks and the task execution times as a function of the elapsed mission time. The planner must have an efficient way to discretize the mission environment because the TEDG requires discrete time units. The TEDG has copies of each node (except the start node) in the Decision Graph (Figure 3-5) for each time step, T, with travel time and task execution arcs which depend on the elapsed time.

Figure 3-6 illustrates the expansion into the time dimension by copying each Decision Graph node for each time step until the maximum allowable mission time, T_{MAX}. This model employs a constant discretization ranging from mission time 0 to T_{MAX} for all nodes except the start node, since the model assumes the UUV starts at time 0 with probability one. The TEDG in Figure 3-6 has the following interpretation:

- Node $S.1.0$ is the start location at time 0.
- Node $E.0.t$ represents being at the end location at time t.
- Nodes $i.0.t$ represent being at task i **before** task execution at time t.
- Nodes $i.1.t$ represent being at task i **after** task execution at time t.
- Red arcs ($i.1.t$, $j.0.s$) represent direct transit from task i at time t to task j at time s **without** a navigation fix.
- Blue arcs ($i.1.t$, $j.0.s$) represent transit from task i at time t to task j at time s **with** a navigation fix.
- Black arcs ($i.0.t$, $i.1.s$) represent executing task i beginning at time t and finishing at time s.

The final step to build the Time Expanded Deterministic Graph is to insert a super sink node so all flow in the network ends at one common node. The super sink node $E.1.T_{MAX}$ connects to all nodes of type $E.0.t$, where t represents the elapsed time of the mission. Node $E.1.T_{MAX}$ represents the end mission location, similar to nodes of type $E.0.t$; the addition of the super sink node has no affect on the actual mission scenario and is only included for

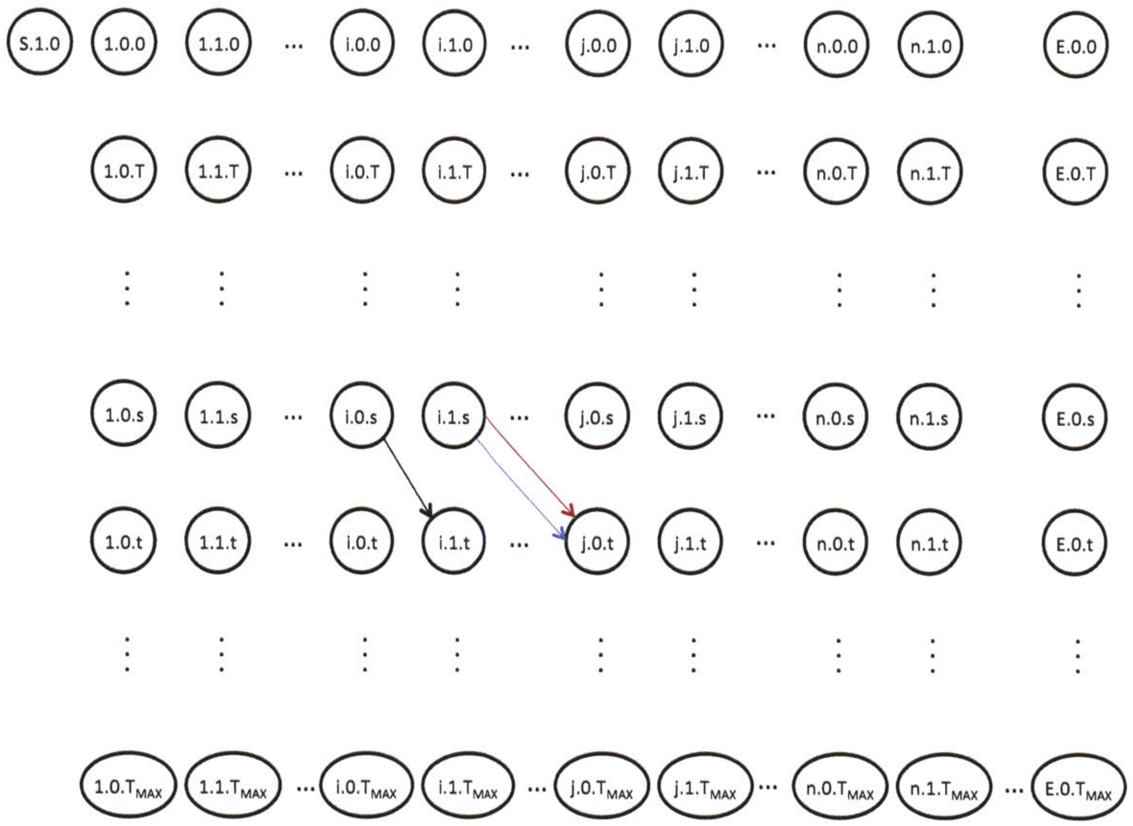


Figure 3-6: Time Expanded Deterministic Graph. Nodes $i.0.t$ represent being at task i before task execution at time t and nodes $i.1.t$ represent being at task i after task execution at time t. Red arcs ($i.1.t$, $j.0.s$) represent direct transit from task i at time t to task j at time s, blue arcs ($i.1.t$, $j.0.s$) represent transit from task i at time t to task j at time s with a navigation fix, and black arcs ($i.0.t$, $i.1.s$) represent executing task i beginning at time t and finishing at time s.

mathematical convenience. The insertion of this node and the adjoining arcs complete the construction of the TEDG. Figure 3-7 contains a two-task example where:

- $T_{MAX} = 6$ time units.
- $T = 1$ time unit.
- Direct travel between each pair of tasks **without** a navigation fix, represented by a red arc, is 1 time unit.
- Travel between each pair of tasks **with** a navigation fix, represented by a blue arc, is 2 time units.
- Each task takes 1 time unit to complete, represented by a black arc.

Although the example in Figure 3-7 assumes the travel times are the same for each time step, the TEDG can also handle situations where the mission time affects the UUV's travel. Consider the two-task example in Figure 3-8, where the parameters are the same as in Figure 3-7 except a shipping lane is present between the start location and task 1 from mission time 0 to time 3. As a result, it now takes the UUV four time units to reach task 1 from the start location direct and five time units with a navigation fix. This alters the network flow model as follows:

- Red arc (S.1.0, 1.0.1) in Figure 3-7 becomes red arc (S.1.0, 1.0.4) in Figure 3-8.
- Blue arc (S.1.0, 1.0.2) in Figure 3-7 becomes blue arc (S.1.0, 1.0.5) in Figure 3-8.

The TEDG amends the Decision Graph by accounting for the effects of a time-varying mission environment, but it does not capture the impact of the accumulated navigation error on the UUV's mission plan. To address this issue, Cates [11] suggests the mission planner could track the expected time since the last navigation fix with flow variables on the TEDG. While Cates' proposed method would not significantly alter the formulation's complexity, the method does not allow for the generation of mission scenarios where the amount of accumulated navigation error affects the UUV's performance, *e.g.* if a task takes longer

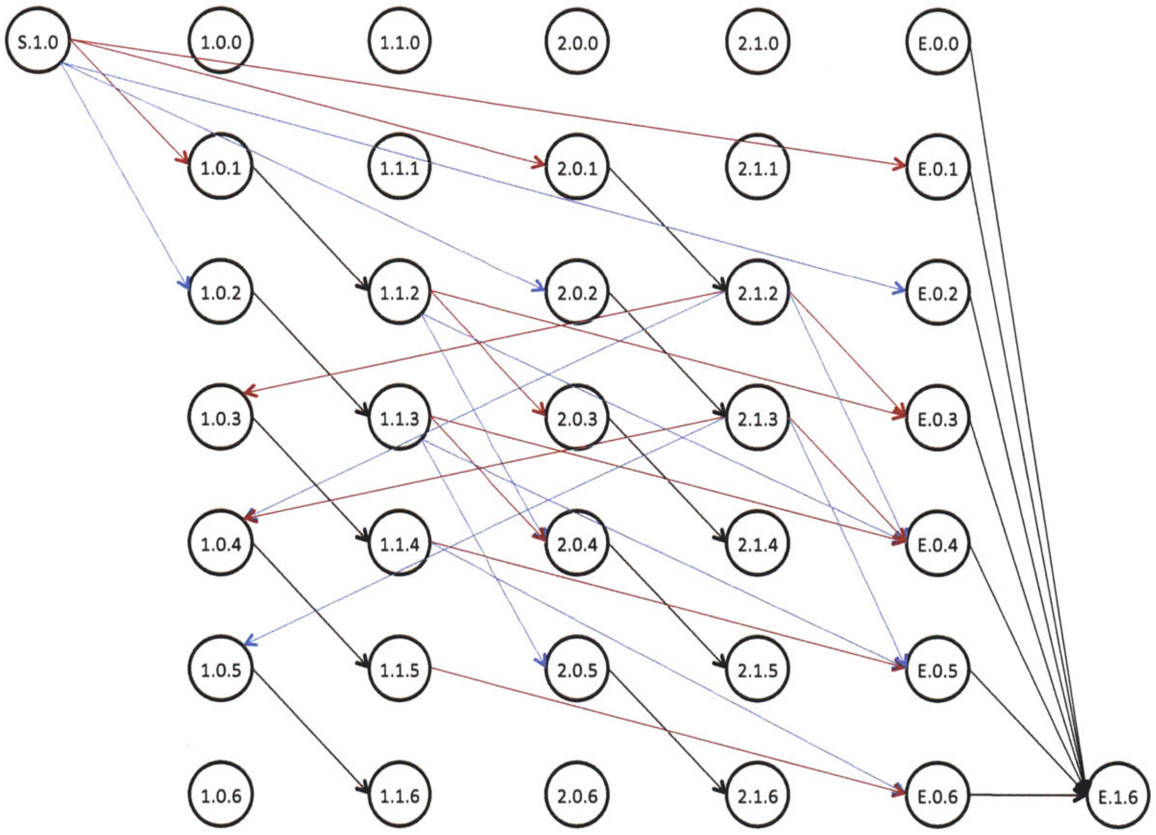


Figure 3-7: Time Expanded Deterministic Graph, Two-Task Example. $T_{MAX} = 6$ time units, $T = 1$ time unit, direct travel between each pair of tasks is 1 time unit, travel between each pair of tasks with a navigation fix is 2 time units, and each task takes 1 time unit to complete.

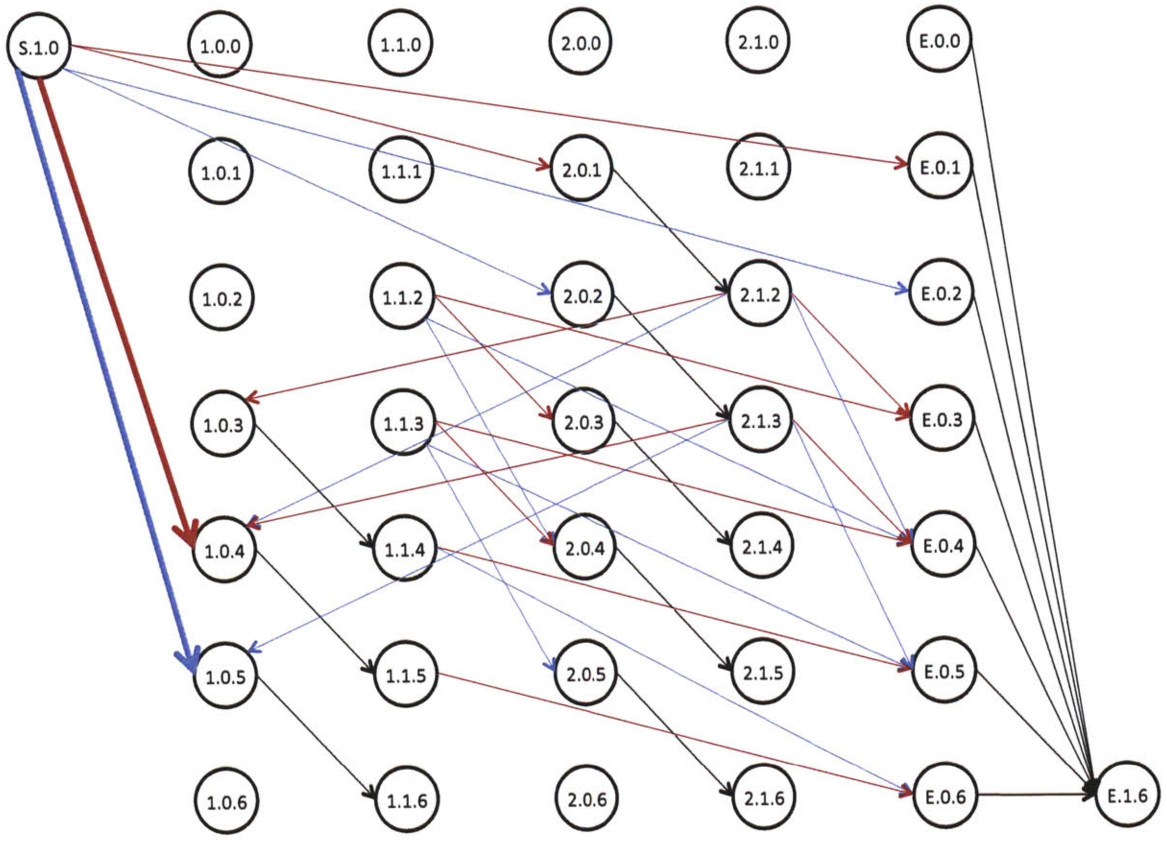


Figure 3-8: Time Expanded Deterministic Graph, Two-Task Example with Shipping Lane. The problem parameters are the same as in Figure 3-7, except note how the arcs representing travel from the start mission location to task 1 are affected by the presence of the shipping lane (highlighted above).

to perform with reduced positional certainty. Expansion into the accumulated navigation error dimension is required to handle these situations. The following section describes a modification of the Decision Graph known as the Navigation Expanded Deterministic Graph.

3.2.5 Navigation Expanded Deterministic Graph

In the Time Expanded Deterministic Graph, a blue arc in the network flow model represents the mission planner's decision to take a navigation fix between two tasks. In the UUVMPP, the UUV performs navigation fixes for two primary reasons:

1. The UUV cannot exceed a certain level of navigation error, N_{MAX}, to perform tasks or navigate in areas that mandate precise positional information.

2. A task has a greater chance of being completed efficiently when the UUV is more confident in its position.

The UUV accumulates navigation error at some rate as it maneuvers underwater for an extended period of time. Once the error becomes too great, the UUV must take a navigation fix to re-orient its position before continuing the mission. The Navigation Expanded Deterministic Graph (NEDG) provides a method to appropriately account for the effects of the accumulated navigation error on the mission plan.

The NEDG is an expansion on the Decision Graph that generates copies of each node (except the start node) for each time step, T, and then inserts travel time and task execution arcs which depend on the UUV's accumulated navigation error. In this graph, the nodes extend into the accumulated navigation error dimension starting at 0 and ending at N_{MAX}. Although this graph closely resembles the TEDG, the NEDG ignores the time factor and only tracks the amount of navigation error the UUV has accumulated at a particular moment in time. Figure 3-9 contains an overview of the NEDG network flow model where:

- Node $S.1.0$ is the start location with zero accumulated navigation error.

- Node $E.0.v$ represents being at the end location with v units of navigation error.

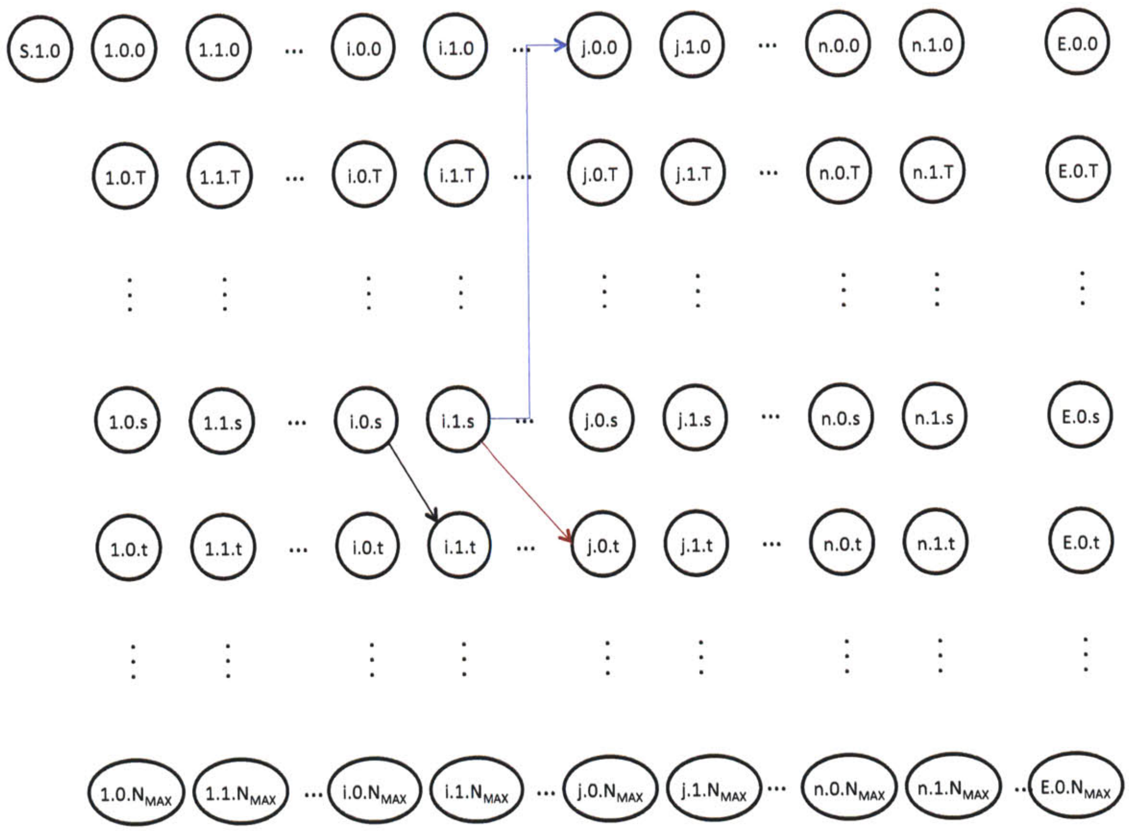


Figure 3-9: Navigation Expanded Deterministic Graph. Nodes $i.0.v$ represent being at task i before task execution with v units of navigation error and nodes $i.1.v$ represent being at task i after task execution with v units of navigation error. Red arcs of type $(i.1,\ j.0)$ represent direct transit from task i to task j, blue arcs of type $(i.1,\ j.0)$ represent transit from task i to task j with a navigation fix (resetting accumulated navigation error to zero), and black arcs represent task execution.

- Nodes $i.0.v$ represent being at task i **before** task execution with v units of navigation error.
- Nodes $i.1.v$ represent being at task i **after** task execution with v units of navigation error.
- Red arcs ($i.1.v$, $j.0.u$) represent direct transit from task i with v units of navigation error to task j with u units of navigation error **without** a navigation fix.
- Blue arcs ($i.1.v$, $j.0.0$) represent transit from task i with v units of navigation error to task j **with** a navigation fix, resetting the accumulated navigation error to zero.
- Black arcs ($i.0.v$, $i.1.u$) represent executing task i with v units of navigation error and finishing with u units of navigation error.

The super sink node $E.1.N_{MAX}$ and adjoining arcs are inserted to finish the construction of the NEDG, similar to the TEDG. Blue arc ($i.1.v$, $j.0.0$) in Figure 3-9 indicates that the navigation fix between task i and j resets the accumulated navigation error to zero. If the quality of the navigation fix is substandard or the navigation fix location is located far away from task j, then the accumulated navigation error may not necessarily be zero once the UUV reaches task j. In this situation, the mission planner can build the model with blue arc ($i.1.v$, $j.0.u$), where u is the expected amount of navigation error once the UUV arrives at task j.

Figure 3-10 contains a two-task example where:

- $N_{MAX} = 6$ navigation units.
- $T = 1$ navigation unit.
- Direct travel between each pair of tasks **without** a navigation fix, represented by a red arc, increases the accumulated navigation error by 2 units.
- Travel between each pair of tasks **with** a navigation fix, represented by a blue arc, resets the accumulated navigation error to zero.

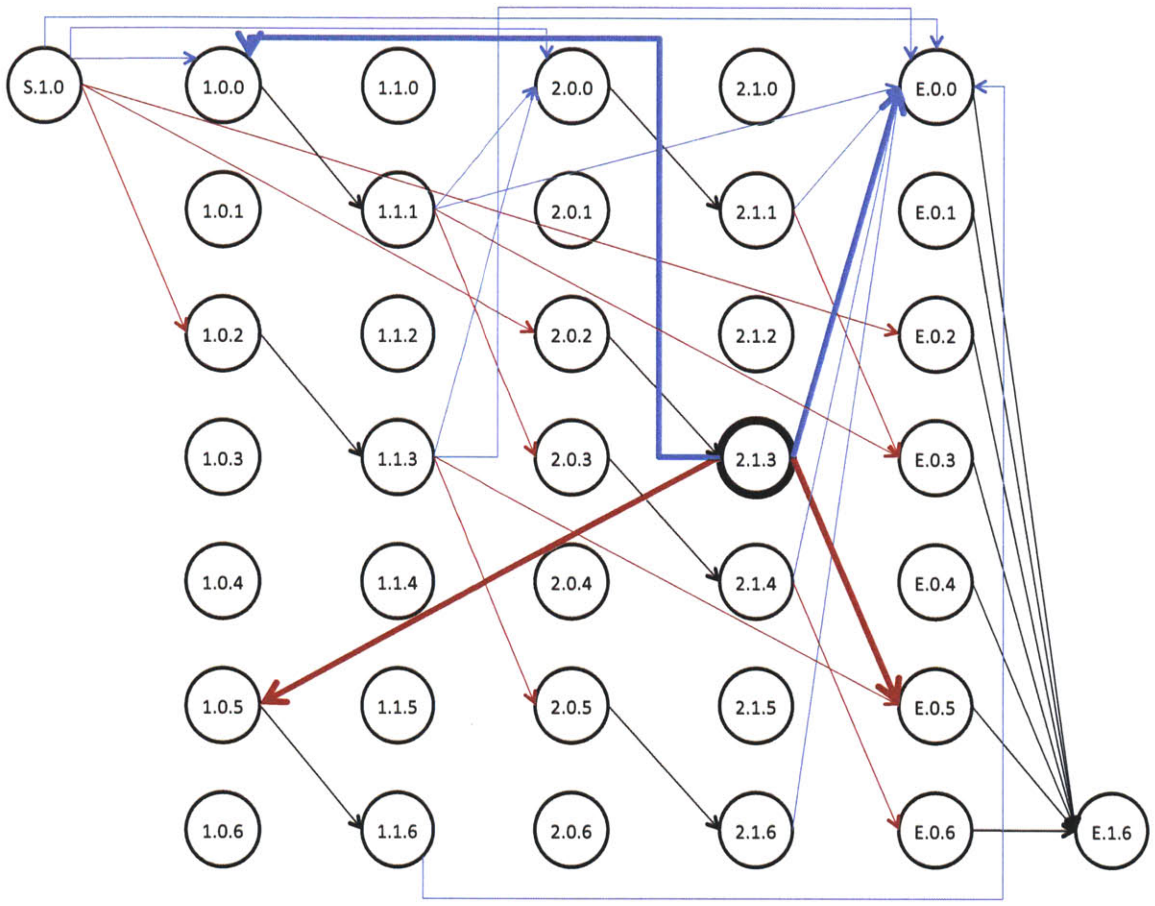


Figure 3-10: Navigation Expanded Deterministic Graph, Two-Task Example. $N_{MAX} = 6$ navigation units, $T = 1$ navigation unit, direct travel between each pair of tasks increases the accumulated navigation error by 2 units, travel between each pair of tasks with a navigation fix resets the accumulated navigation error to 0, and the execution of each task increases the accumulated navigation error by 1 unit. Table 3.3 summarizes the decisions the mission planner faces at node 2.1.3 — *i.e.* at task 2 after task completion with 3 units of navigation error — and the arcs associated with each decision (highlighted above).

Decision	Associated Arc
Travel to Task 1 with a navigation fix	Blue arc (2.1.3, 1.0.0)
Travel to Task 1 without a navigation fix	Red arc (2.1.3, 1.0.5)
Travel to End with a navigation fix	Blue arc (2.1.3, E.0.0)
Travel to End without a navigation fix	Red arc (2.1.3, E.0.5)

Table 3.3: NEDG Two-Task Example, At Node 2.1.3. Refer to Figure 3-10.

- The execution of each task increases the accumulated navigation error by 1 unit, represented by a black arc.

Consider node 2.1.3 in Figure 3-10, which means the mission planner is located at task 2 after task completion with three units of accumulated navigation error. Table 3.3 summarizes the decisions the mission planner faces at this node and the arcs that are related to each decision.

The NEDG is an accurate mathematical representation of the UUV's mission environment if time has a negligible effect. If tides and currents have minimal impact on the UUV's travel time and there are no time-dependent no-go zones in the operating area, then the expansion into the time dimension may not be necessary.

The TEDG and NEDG are two separate models that account for an important facet of the UUVMPP while ignoring other mission factors. By combining these two models into the Time and Navigation Expanded Deterministic Graph, the mission planner can adequately model the effects of time and accumulated navigation error together in the UUVMPP.

3.2.6 Time and Navigation Expanded Deterministic Graph

The Time and Navigation Expanded Deterministic Graph (TANEDG) combines the TEDG and NEDG into a three-dimensional network flow, accounting for both the time constraint and the UUV's accumulated navigation error constraint.

To construct the Time and Navigation Expanded Deterministic Graph, the model has copies of each node (except the start node) in the Decision Graph for each possible time from 0 to T_{MAX} (discretized by parameter T) and each possible accumulated navigation error from 0 to N_{MAX} (discretized by parameter T). Therefore, each node in the Decision Graph corresponds to $\frac{T_{MAX}}{T} * \frac{N_{MAX}}{T}$ nodes in the TANEDG, one for every possible combination of time and navigation error.

Since the model expands the nodes in the Decision Graph in two separate dimensions, the resulting TANEDG is a three-dimensional network flow model. The expansion **down** the page is the time dimension, and the expansion **into** the page is the accumulated navigation error dimension. Figure 3-11 portrays how node 1.0 from the Decision Graph expands to fit the TANEDG framework. Nodes are of the form $i.k.t.v$, where:

- i is the task label; $i \in \{S, 1, 2, \ldots, n, E\}$.
- $k \in \{0, 1\}$, where 0 and 1 represent being at task i before and after task execution, respectively.
- t is the elapsed time since mission start.
- v is the accumulated navigation error.

To build the full Time and Navigation Expanded Deterministic Graph, the model repeats the process outlined in Figure 3-11 for all nodes (except the start node) and adds arcs that depend on both time and the accumulated navigation error. The TANEDG has one super sink node $E.1.T_{MAX}.N_{MAX}$ that connects to all end mission location nodes of type $\{E.0.c.d\}$ where $c \in \{0, T, 2*T, \ldots, T_{MAX}\}$ and $d \in \{0, T, 2*T, \ldots, N_{MAX}\}$.

The top-most layer of the model is equivalent to the Time Expanded Deterministic Graph with zero accumulated navigation error, and each layer underneath represents the same TEDG with a different amount of accumulated navigation error. In Figure 3-11, nodes of form $\{1.0.c.0\}$ are on the first layer, nodes of form $\{1.0.c.\mathrm{T}\}$ are on the second layer, etc. The expansion continues until the bottom layer where all nodes representing task 1 before task completion are of the form $\{1.0.c.N_{MAX}\}$. The TANEDG is interpreted as follows:

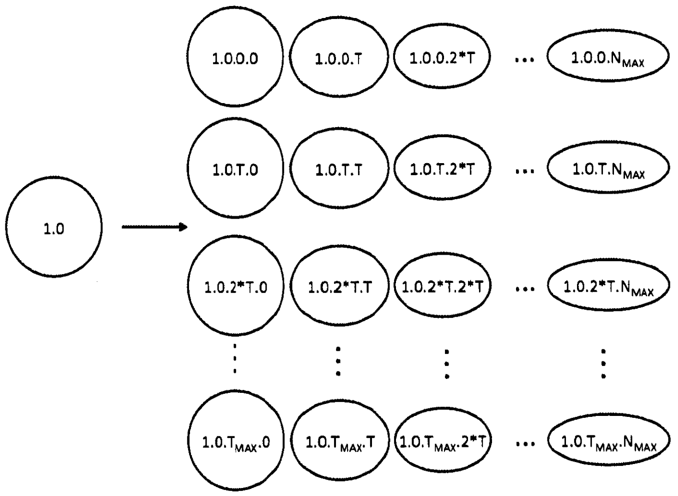


Figure 3-11: Time and Navigation Expanded Deterministic Graph, Expansion of Node 1.0. All states are of the form $1.0.t.v$, where t is the elapsed time since mission start and v is the accumulated navigation error. Note how as the accumulated navigation error increases, the nodes become smaller. This illustrates the depth of the model, where smaller nodes indicate a greater expansion into the accumulated navigation error dimension and go into the page.

- Node $S.1.0.0$ is the start location at time zero with zero accumulated navigation error.
- Node $E.0.t.v$ represents being at the end location at time t with v units of navigation error.
- Nodes $i.0.t.v$ represent being at task i **before** task execution at time t with v units of navigation error.
- Nodes $i.1.t.v$ represent being at task i **after** task execution at time t with v units of navigation error.
- Red arcs $(i.1.t.v,\ j.0.s.u)$ represent direct transit from task i at time t with v units of navigation error to task j at time s with u units of navigation error **without** a navigation fix.
- Blue arcs $(i.1.t.v,\ j.0.s.0)$ represent transit from task i at time t with v units of navigation error to task j at time s **with** a navigation fix, resetting the accumulated navigation error to zero.

From Node	**Arc**	**To Node**	**UUV's Action**
S.1.0.0	Blue arc (S.1.0.0, 1.0.3.0)	1.0.3.0	Travel, Start to Task 1 (fix)
1.0.3.0	Black arc (1.0.3.0, 1.1.4.1)	1.1.4.1	Execute Task 1
1.1.4.1	Red arc (1.1.4.1, E.0.6.3)	E.0.6.3	Travel, Task 1 to End (direct)
E.0.6.3	Black arc (E.0.6.3, E.1.10.3)	E.1.10.3	No action – mission complete

Table 3.4: TANEDG Two-Task Example, $S \xrightarrow{\text{fix}} 1 \xrightarrow{\text{direct}} E$. Refer to Figure 3-12.

- Black arcs ($i.0.t.v$, $i.1.s.u$) represent executing task i at time t with v units of navigation error and finishing at time s with u units of navigation error.

Figure 3-12 illustrates a two-task example with $T = 1$ time unit, $T_{MAX} = 10$ time units and $N_{MAX} = 3$ navigation units. Since $N_{MAX} = 3$ navigation units, the network flow in Figure 3-12 has four layers to the model representing accumulated navigation errors of 0, 1, 2, and 3 units, respectively. In addition:

- Direct travel between each pair of tasks **without** a navigation fix, represented by a red arc, takes 2 time units and increases the accumulated navigation error by 2 navigation units.

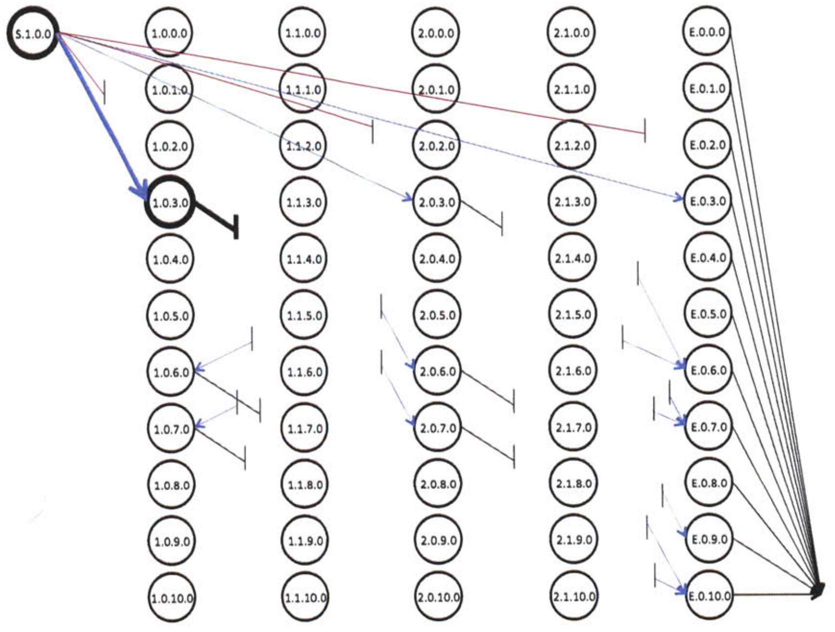
- Travel between each pair of tasks **with** a navigation fix, represented by a blue arc, takes 3 time units and resets the accumulated navigation error to zero navigation units.

- Each task takes 1 time unit and increases the accumulated navigation error by 1 unit, represented by a black arc.

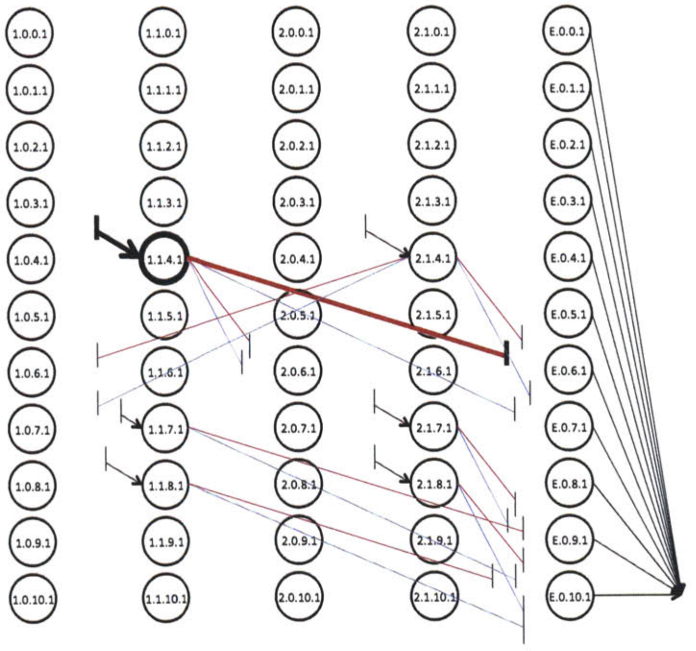
Table 3.4 outlines how the network flow represents the task sequence $S \xrightarrow{\text{fix}} 1 \xrightarrow{\text{direct}} E$.

The Time and Navigation Expanded Deterministic Graph is an appropriate model for the UUVMPP if all parameters are deterministic. This assumption may not be realistic in practice since it is difficult to anticipate the UUV's movement through the mission environment exactly. The mission planner's goal is to find a robust solution that performs well in

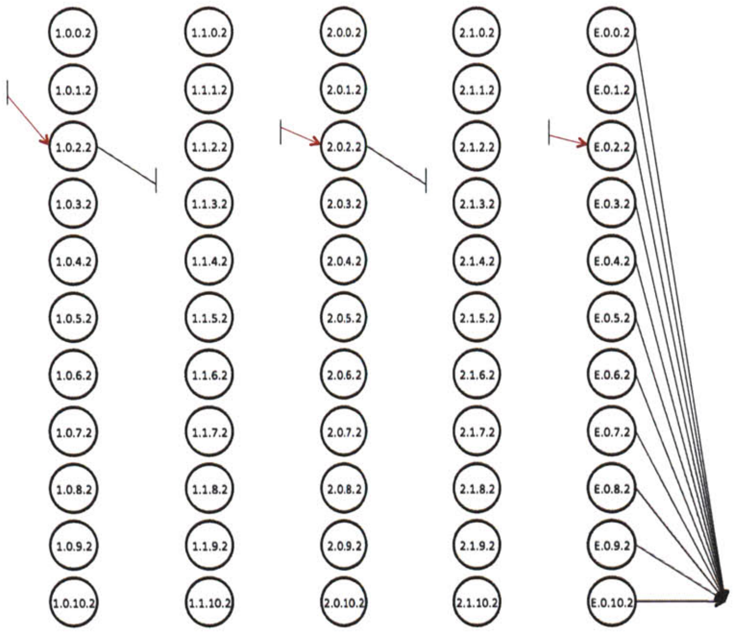
Figure 3-12: Time and Navigation Expanded Deterministic Graph, Two-Task Example



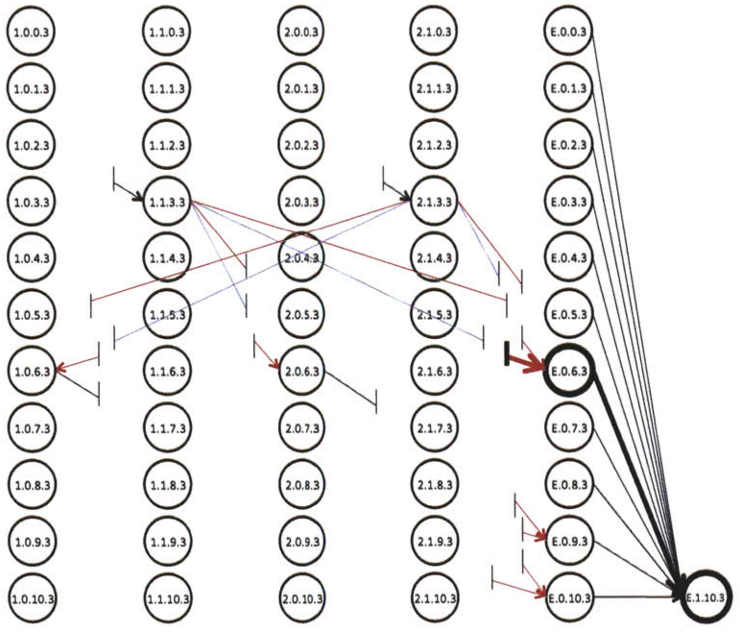
(a) 0 Units Navigation Error



(b) 1 Unit Navigation Error



(c) 2 Units Navigation Error



(d) 3 Units Navigation Error

a variety of situations with a certain amount of confidence the chosen task sequence will be successful; to account for this, stochasticity is introduced into the network flow model with the Time and Navigation Expanded Stochastic Graph.

3.2.7 Time and Navigation Expanded Stochastic Graph

The Time and Navigation Expanded Stochastic Graph (TANESG) is an extension of the TANEDG where the UUV's travel times and task execution times are random variables. This enables the mission planner to develop UUV route solutions that do not exceed the given time and energy constraints with a certain probability.

To construct the TANESG, the model has copies of each node in the Decision Graph for each possible time from 0 to T_{MAX} and each possible accumulated navigation error from 0 to N_{MAX}. The construction of the TANESG is similar to its deterministic counterpart except this model also has copies of the start node that represent the possibility of the UUV starting its mission at a different time. In addition, the model inserts two more nodes, the super source node $S.0.0.0$ and super sink node $E.1.T_{MAX}.N_{MAX}$, which represents the absolute start and end of the mission, respectively.

Similar to the TANEDG, the TANESG has arcs that depend on both time and the accumulated navigation error. To represent the UUV's stochastic movement through the mission environment, the TANESG utilizes a distribution of arcs to model the UUV's travel times and task execution times. For example, in the deterministic graph, if the mission planner leaves node $i.k.t.v$ and chooses to maneuver to a node of type $j.l$, then it will always reach a single node $j.l.s.u$ (*i.e.* the elapsed mission time and accumulated navigation error can be predicted exactly). In the stochastic graph, if the mission planner leaves node $i.k.t.v$ and chooses to maneuver to a node of type $j.l$, then it will arrive at a set of nodes of type $j.l.s.u$, $s \in \{0, T, \ldots, T_{MAX}\}$ and $u \in \{0, T, \ldots, N_{MAX}\}$, according to some specified probability distribution.

Figure 3-13 illustrates the conversion of arcs from the deterministic graph to the stochastic graph. In the deterministic case, the mission planner reaches node $j.l.s.u$ from node $i.k.t.v$

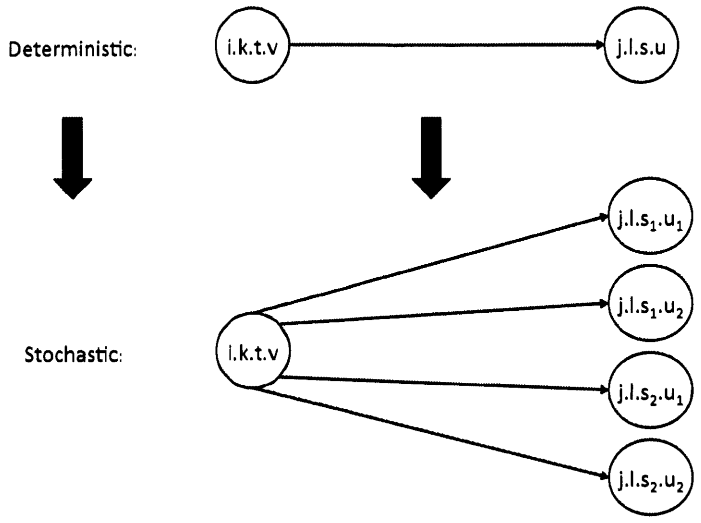


Figure 3-13: Time and Navigation Expanded Stochastic Graph, Arc Distributions. In the deterministic case, the mission planner reaches node $j.l.s.u$ from node $i.k.t.v$ with probability one. From node $i.k.t.v$ in the stochastic case, the mission planner either reaches node $j.l.s_1.u_1$, $j.l.s_1.u_2$, $j.l.s_2.u_1$, or $j.l.s_2.u_2$, where the sum of these probabilities equals 1.

with probability one. From node $i.k.t.v$ in the stochastic case, the mission planner either reaches node $j.l.s_1.u_1$, $j.l.s_1.u_2$, $j.l.s_2.u_1$, or $j.l.s_2.u_2$, where the sum of these probabilities equals 1.

The Time and Navigation Expanded Stochastic Graph adequately combines the time and accumulated navigation error components of the UUVMPP into one network flow model. The following section develops the MIP formulations that allow the mission planner to use the TANESG to determine an *a priori* task route robustly.

3.3 Mixed-Integer Programming Formulations

The ultimate goal of the UUVMPP is to find the task sequence that maximizes the expected benefit for a given UUV mission. MIP Formulations 1 and 2 utilize the Time and Navigation Expanded Stochastic Graph to find the best UUV route in the Decision Graph, repeated in Figure 3-14 for convenience.

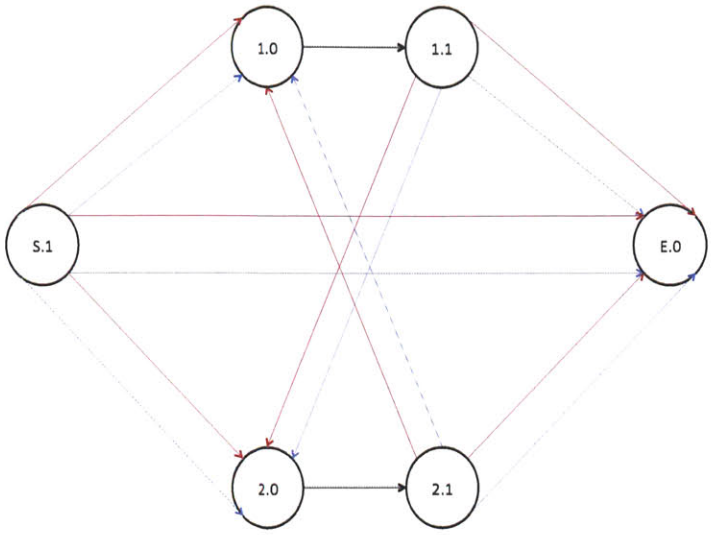


Figure 3-14: Decision Graph, Two-Task Example. Repeat of Figure 3-5.

3.3.1 MIP Formulation 1

Notation

This section formalizes the definitions from the network flow models, particularly the Decision Graph and the TANESG, to develop the notation required in MIP Formulation 1. The **Decision Graph**, G, has the following structure:

- N: the set of nodes $i.k$ in G; $i \in \{S, 1, 2, \ldots, n, E\}$ and $k \in \{0, 1\}$.
- A: the set of arcs $(i.k, j.l)$ in G; $i, j \in \{S, 1, 2, \ldots, n, E\}$ and $k, l \in \{0, 1\}$.
- $G = (N, A)$, the Decision Graph.
- $\mathbf{B}$: the node-arc incidence matrix for G.
- $\mathbf{d}$: the demand vector for G, where source $S.1$ has a supply of 1, sink $E.0$ has a demand of 1, and all other nodes $\in N$ have balanced flow.

To solve the network optimization problem on Decision Graph G, let:

- $\breve{y}_{i.k}^{j.l}$ = flow variable, the number of times the UUV transits on black arc $(i.k, j.l)$ in G; $i = j$, $k = 0$, and $l = 1$.

- $\dot{y}_{i.k}^{j.l}$ = flow variable, the number of times the UUV transits on red arc $(i.k, j.l)$ in G; $i \neq j$, $k = 1$, and $l = 0$.

- $\bar{y}_{i.k}^{j.l}$ = flow variable, the number of times the UUV transits on blue arc $(i.k, j.l)$ in G; $i \neq j$, $k = 1$, and $l = 0$.

- $y_{i.k}^{j.l} = \breve{y}_{i.k}^{j.l} \cup \dot{y}_{i.k}^{j.l} \cup \bar{y}_{i.k}^{j.l}$.

- $\mathbf{y}$: vector representing all feasible $y_{i.k}^{j.l}$ values.

The **Time and Navigation Expanded Stochastic Graph**, G_s, has the following structure:

- β: the confidence level of successfully completing the given task sequence; *i.e.* Pr{UUV reaches end mission location by T_{MAX} with the given task sequence} $\geq \beta$.

- N_s: the set of all nodes $i.k.t.v$ in G_s; $i \in \{S, 1, 2, \ldots, n, E\}$, $t \in \{0, T, \ldots, T_{MAX}\}$, $k \in \{0,\ 1\}$, and $v \in \{0, T, \ldots, N_{MAX}\}$.

- A_s: the set of all arcs $(i.k.t.v, j.l.s.u)$ in G_s; $i, j \in \{S, 1, 2, \ldots, n, E\}$, $k, l \in \{0,\ 1\}$, $t, s \in \{0, T, \ldots, T_{MAX}\}$, and $v, u \in \{0, T, \ldots, N_{MAX}\}$.

- $G_s = (N_s, A_s)$, the Time and Navigation Expanded Stochastic Graph.

- $\mathbf{B_s}$: the node-arc incidence matrix for G_s.

- $\mathbf{d_s}(\beta)$: the demand vector for G_s, where super source $S.0.0.0$ has a supply of 1, super sink $E.1.T_{MAX}.N_{MAX}$ has a demand of β, and all other nodes $\in N_s$ have balanced flow.

To solve the network optimization problem on the Time and Navigation Expanded Stochastic Graph G_s, let:

- $\breve{p}_{i.k.t.v}^{j.l.s.u}$ = Pr{UUV leaves node of type $i.k$ at time t with navigation error v, arriving at node of type $j.l$ at time s with navigation error u | at node $i.k.t.v$, the mission planner chooses to travel to node of type $j.l$}, where $k = 0$ and $l = 1$ (*i.e.* executing task i).

- $\dot{p}^{j.l.s.u}_{i.k.t.v} = \Pr\{$UUV leaves node of type $i.k$ at time t with navigation error v, arriving at node of type $j.l$ at time s with navigation error $u \mid$ at node $i.k.t.v$, the mission planner chooses to travel to node of type $j.l$ direct **without** a navigation fix$\}$, where $k = 1$ and $l = 0$ (*i.e.* travel from task i to task j).

- $\bar{p}^{j.l.s.u}_{i.k.t.v} = \Pr\{$UUV leaves node of type $i.k$ at time t with navigation error v, arriving at node of type $j.l$ at time s with navigation error $u \mid$ at node $i.k.t.v$, the mission planner chooses to travel to node of type $j.l$ **with** a navigation fix$\}$, where $k = 1$ and $l = 0$ (*i.e.* travel from task i to task j).

- $p^{j.l.s.u}_{i.k.t.v} = \check{p}^{j.l.s.u}_{i.k.t.v} \cup \dot{p}^{j.l.s.u}_{i.k.t.v} \cup \bar{p}^{j.l.s.u}_{i.k.t.v}$.

- $\mathbf{p}$: vector representing all feasible $p^{j.l.s.u}_{i.k.t.v}$ values.

- $\check{x}^{j.l.s.u}_{i.k.t.v}$ = flow variable, the probability the UUV transits on black arc $(i.k.t.v, j.l.s.u)$ in G_s; $i = j$, $k = 0$, and $l = 1$.

- $\dot{x}^{j.l.s.u}_{i.k.t.v}$ = flow variable, the probability the UUV transits on red arc $(i.k.t.v, j.l.s.u)$ in G_s; $i \neq j$, $k = 1$, and $l = 0$.

- $\bar{x}^{j.l.s.u}_{i.k.t.v}$ = flow variable, the probability the UUV transits on blue arc $(i.k.t.v, j.l.s.u)$ in G_s; $i \neq j$, $k = 1$, and $l = 0$.

- $x^{j.l.s.u}_{i.k.t.v} = \check{x}^{j.l.s.u}_{i.k.t.v} \cup \dot{x}^{j.l.s.u}_{i.k.t.v} \cup \bar{x}^{j.l.s.u}_{i.k.t.v}$.

- $\mathbf{x}$: vector representing all feasible $x^{j.l.s.u}_{i.k.t.v}$ values.

Constraints

MIP Formulation 1 contains two vectors of flow variables, $\mathbf{x}$ and $\mathbf{y}$. The goal of the UUV-MPP is to find the optimal route $\mathbf{y}$ on the Decision Graph G by calculating the transition probabilities $\mathbf{x}$ on the Time and Navigation Expanded Stochastic Graph G_s. First, the formulation must appropriately restrict the flow variables. Since no cycles are allowed in G and tasks can only can be completed once, then $\mathbf{y}$ is a binary variable:

$$\mathbf{y} \in \{0, 1\} \tag{3.3}$$

Because $\mathbf{x}$ is a probability in G_s, then:

$$0 \leq \mathbf{x} \leq 1 \tag{3.4}$$

The formulation captures the flow constraints on G — *i.e.* source $S.1$ has a supply of 1, sink $E.0$ has a demand of 1, and all other nodes $\in N$ have balanced flow — by Equation 3.5:

$$\mathbf{B}\mathbf{y} = \mathbf{d} \tag{3.5}$$

Likewise, the model accounts for the flow constraints on G_s — *i.e.* super source $S.0.0.0$ has a supply of 1, super sink $E.1.T_{MAX}.N_{MAX}$ has a demand of β, and all other nodes $\in N_s$ have balanced flow — by Equation 3.6. The model's choice of β allows the mission planner to find a route guaranteed to be successful with probability β, given accurate mission inputs.

$$\mathbf{B_s}\mathbf{x} \leq \mathbf{d_s}(\beta) \tag{3.6}$$

To tie graphs G and G_s together, Equation 3.7 restricts $\mathbf{y}$, the flow variable in G, to only consider routes where there is a nonzero probability in G_s of going from a node of type $i.k$ to a node of type $j.l$, summarized by the $\mathbf{x}$ flow variable. Summing over all values t, v, s, and u accounts for all ways flow can transition from an $i.k$ node to a $j.l$ node:

$$\sum_{t,v,s,u|(i.k.t.v,j.l.s.u)\in A_s} x_{i.k.t.v}^{j.l.s.u} \leq y_{i.k}^{j.l} \qquad \forall (i.k, j.l) \in A \tag{3.7}$$

Equation 3.8 forces the flow coming out of a node in G_s to be less than or equal to the flow coming into a node in G_s. More formally, when multiplying together all flow coming into some node $i.k.t.v$ in Graph G_s,

$$\sum_{i'.k'.t'.v' | (i'.k'.t'.v', i.k.t.v) \in A_s} x_{i'.k'.t'.v'}^{i.k.t.v}$$

and the conditional probability that the mission planner chooses a node of type $j.l$ from node $i.k.t.v$ and arrives at time s with u units of navigation error given the UUV is located at node $i.k.t.v$,

$$p_{i.k.t.v}^{j.l.s.u}$$

the product is the flow coming from node $i.k.t.v$ to node $j.l.s.u$. Since the formulation requires the flow coming into a node to be greater than or equal to the flow coming out of the node, Equation 3.8 forces the flow from node $i.k.t.v$ towards node $j.l.s.u$ to be greater than or equal to the actual flow arriving at node $j.l.s.u$ from node $i.k.t.v$, or

$$x_{i.k.t.v}^{j.l.s.u} \, .$$

As a result, Equation 3.8 states:

$$p_{i.k.t.v}^{j.l.s.u} \times \left(\sum_{i'.k'.t'.v' | (i'.k'.t'.v', i.k.t.v) \in A_s} x_{i'.k'.t'.v'}^{i.k.t.v} \right) \geq x_{i.k.t.v}^{j.l.s.u} \qquad \forall (i.k.t.v, j.l.s.u) \in A_s \qquad (3.8)$$

Equation 3.9 is an extension of Equation 3.8 for the flow from the super source node $S.0.0.0$ to all nodes of type $S.1$. Since the planner begins a mission with probability one, the super source node has a supply of one unit; therefore, the value 1 replaces the summation in Equation 3.8. Equation 3.9 is as follows:

$$p_{S.0.0.0}^{S.1.t.v} \geq x_{S.0.0.0}^{S.1.t.v} \qquad \forall t, v \text{ such that } (S.0.0.0, S.1.t.v) \in A_s \qquad (3.9)$$

Equations 3.3 through 3.9 combine to form the constraints necessary to solve the UUVMPP with MIP Formulation 1.

Objective Function

For each UUV mission, the planner must decide which objective to maximize or minimize, *e.g.* maximize the number of completed tasks, minimize risk, maximize stealth, etc. MIP

Formulation 1 optimizes a chosen linear objective function subject to the constraints specified in Section 3.3.1 to find the best *a priori* task route. This section provides examples of objective functions that maximize the expected utility of a given route, which can be extended to any chosen mission objective.

Let r_i be the reward for completing task i, where $r_i \geq 0 \, \forall \, i$. Equation 3.10 proposes the objective function denoted **Full Reward** which assumes the mission planner obtains the full reward r_i for completing task i. To maximize the expected reward gained for a UUV's route with the above parameters, the formulation uses the $\mathbf{x}$ flow variables to capture whether or not task i has been chosen for the UUV's route. The resulting objective function is:

$$\max \quad \sum_{i=1}^{n} \left(r_i \times \left(\sum_{t,v,s,u \mid (i.0.t.v, i.1.s.u) \in A_s} x_{i.0.t.v}^{i.1.s.u} \right) \right) \tag{3.10}$$

Suppose the mission planner desires an objective function that depends on when the UUV arrives at task i. As an example, consider the **Linear Decrease** reward function in Equation 3.11. The Linear Decrease objective function penalizes the UUV for having a large accumulated navigation error when arriving at task i, since a smaller amount of positional certainty could reduce the benefit of performing task i. To model this situation, one possibility is to add the term $\frac{N_{MAX}-v}{N_{MAX}}$, where the reward for completing task i reduces linearly depending on the accumulated navigation error v when arriving at task i. The Linear Decrease objective function is as follows:

$$\max \quad \sum_{i=1}^{n} \left(r_i \times \left(\sum_{t,v,s,u \mid (i.0.t.v, i.1.s.u) \in A_s} x_{i.0.t.v}^{i.1.s.u} \times \frac{N_{MAX} - v}{N_{MAX}} \right) \right) \tag{3.11}$$

To generalize, let $f(i,t,v,s,u)$ indicate a function dependent on the task i and the parameters t, v, s, and u. Equation 3.12 represents the universal objective function in the UUVMPP:

$$\max \quad \sum_{i=1}^{n} \left(r_i \times \left(\sum_{t,v,s,u \mid (i.0.t.v, i.1.s.u) \in A_s} x_{i.0.t.v}^{i.1.s.u} \times f(i,t,v,s,u) \right) \right) \tag{3.12}$$

By combining the objective function in Equation 3.12 and the constraints in Equations 3.3 through 3.9, MIP Formulation 1 is complete, stated as follows:

MIP Formulation 1

$$\max \quad \sum_{i=1}^{n}\left(r_i \times \left(\sum_{t,v,s,u|(i.0.t.v,i.1.s.u)\in A_s} x_{i.0.t.v}^{i.1.s.u} \times f(i,t,v,s,u)\right)\right)$$

s.t.

$$\mathbf{By} = \mathbf{d}$$

$$\mathbf{B_s x} \le \mathbf{d_s}(\beta)$$

$$\sum_{t,v,s,u|(i.k.t.v,j.l.s.u)\in A_s} x_{i.k.t.v}^{j.l.s.u} \le y_{i.k}^{j.l} \qquad \forall (i.k,j.l) \in A$$

$$p_{i.k.t.v}^{j.l.s.u} \times \left(\sum_{i'.k'.t'.v'|(i'.k'.t'.v',i.k.t.v)\in A_s} x_{i'.k'.t'.v'}^{i.k.t.v}\right) \ge x_{i.k.t.v}^{j.l.s.u} \qquad \forall (i.k.t.v, j.l.s.u) \in A_s$$

$$p_{S.0.0.0}^{S.1.t.v} \ge x_{S.0.0.0}^{S.1.t.v} \qquad \forall t,v \in A_s$$

$$0 \le \mathbf{x} \le 1$$

$$\mathbf{y} \in \{0,1\}$$

MIP Formulation 1 finds the *a priori* task route that maximizes the expected reward and guarantees the UUV reaches the end mission location within T_{MAX} with probability β. However, if the UUV experiences a situation where it will not reach the end location within T_{MAX} (with probability $1-\beta$), then MIP Formulation 1 does not allow the mission planner to re-plan, *i.e.* re-solve the UUVMPP during the execution of a mission plan. MIP Formulation 2 proposes a modification to the TANESG that permits route alteration.

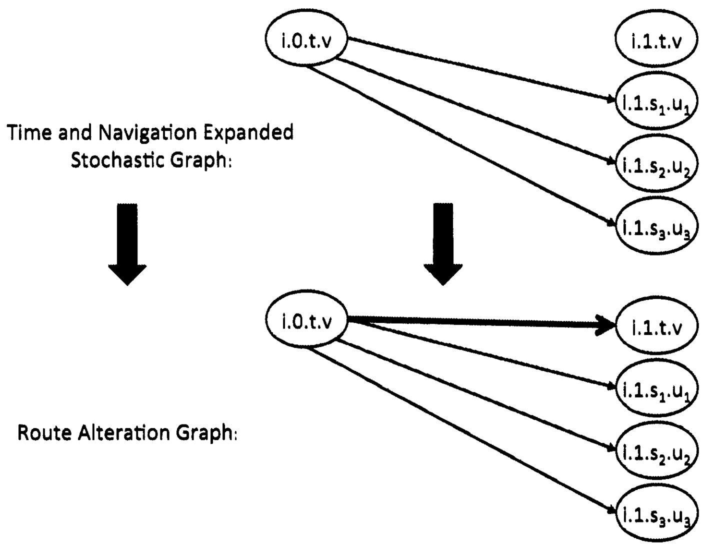


Figure 3-15: Skip Task Arcs in Route Alteration Graph. The Route Alteration Graph adds "Skip Task" arcs $(i.0.t.v, i.1.t.v)$ $\forall$ i, t, and v to G_s.

3.3.2 MIP Formulation 2 (Re-planning)

Notation

For MIP Formulation 2, the Route Alteration Graph G_a amends the Time and Navigation Expanded Stochastic Graph to account for situations where the UUV realizes worst-case disturbances; *e.g.*, an unpredicted weather system enters the mission environment and causes the UUV to maneuver at a slower than predicted speed. G_a allows the mission planner to change its course in one of two ways:

1. The mission planner can skip the next task $\Leftrightarrow$ add "Skip Task" arcs $(i.0.t.v, i.1.t.v)$ $\forall$ i, t, and v to G_s; see Figure 3-15.

2. The mission planner can end the mission and travel directly to the end location $\Leftrightarrow$ add "Mission End" arcs $(i.1.t.v, E.1.T_{MAX}.N_{MAX})$ $\forall$ i, t, and v to G_s; see Figure 3-16.

As a result, the **Route Alteration Graph** has the following structure:

- N_a: the set of all nodes $i.k.t.v$ in G_a; $N_a = N_s$.

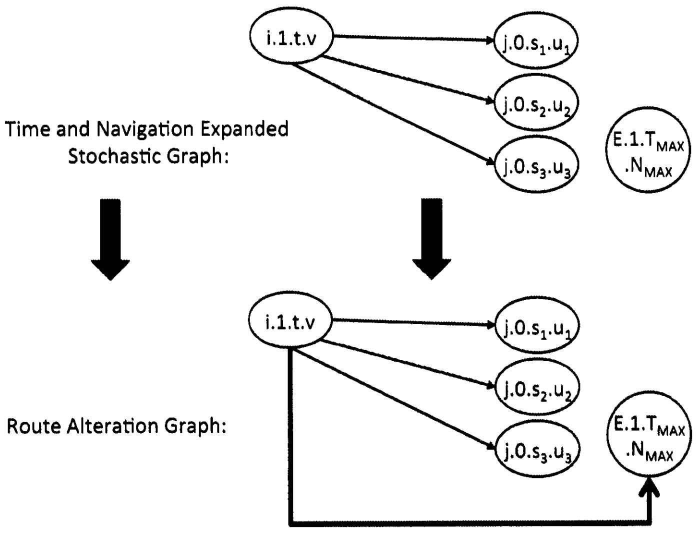


Figure 3-16: Mission End Arcs in Route Alteration Graph. The Route Alteration Graph adds "Mission End" arcs $(i.1.t.v, E.1.T_{MAX}.N_{MAX})$ $\forall$ i, t, and v to G_s.

- A_a: the set of all arcs $(i.k.t.v, j.l.s.u)$ in G_a; $i, j \in \{S, 1, 2, \ldots, n, E\}$, $k, l \in \{0,\ 1\}$, t, s $\in \{0, T, \ldots, T_{MAX}\}$, and $v, u \in \{0, T, \ldots, N_{MAX}\}$.
- $G_a = (N_a, A_a)$, the Route Alteration Graph.
- $\mathbf{B_a}$: the node-arc incidence matrix for G_a.
- $\mathbf{d_a}(\beta)$: the demand vector for G_a, where super source $S.0.0.0$ has a supply of 1, super sink $E.1.T_{MAX}.N_{MAX}$ has a demand of β, and all other nodes $\in N_a$ have balanced flow.

To solve the network optimization problem on the Route Alteration Graph, the model introduces the flow variable $\mathbf{z}$. Let:

- $z_{i.0.t.v} = \begin{cases} 1, & \text{if mission planner skips performing task } i \text{ at node } i.0.t.v \\ 0, & \text{otherwise} \end{cases}$
- $z_{i.1.t.v} = \begin{cases} 1, & \text{if mission planner goes to end mission location from node } i.1.t.v \\ 0, & \text{otherwise} \end{cases}$

- **z**: vector representing all feasible $z_{i.0.t.v}$ and $z_{i.1.t.v}$ values.

Constraints

MIP Formulation 2 contains three vectors of flow variables, **x**, **y**, and **z**. The restrictions on **x** and **y** remain the same — $0 \leq \mathbf{x} \leq 1$ and **y** binary — while **z** is also a binary flow variable. To satisfy the flow constraints on G and G_a, MIP Formulation 2 utilizes:

$$\mathbf{By} = \mathbf{d}$$

$$\mathbf{B_a x} \leq \mathbf{d_a}(\beta)$$

To tie graphs G and G_a together, MIP Formulation 2 amends Equation 3.7 by constraining the arcs in G, $(i.k, j.l)$, such that $j.l \neq E.0$ (*i.e.* all arcs representing travel except the arcs leading to the end node). The formulation only restricts the values $y_{i.k}^{j.l}$ that do not lead to the end mission location to allow unrestricted flow in G_a over arcs $(i.1.t.v, E.1.T_{MAX}.N_{MAX})$, which in turn permits the mission planner to take the "Mission End" arcs:

$$\sum_{t,v,s,u|(i.k.t.v,j.l.s.u)\in A_a} x_{i.k.t.v}^{j.l.s.u} \leq y_{i.k}^{j.l} \qquad \forall (i.k, j.l) \in A | j.l \neq E.0$$

Since the flow coming out of a node in G_a is less than or equal to the flow coming into a node in G_a, Formulation 2 contains the constraints found in Equations 3.8 and 3.9:

$$p_{i.k.t.v}^{j.l.s.u} \times \left(\sum_{i'.k'.t'.v'|(i'.k'.t'.v',i.k.t.v)\in A_a} x_{i'.k'.t'.v'}^{i.k.t.v} \right) \geq x_{i.k.t.v}^{j.l.s.u} \qquad \forall (i.k.t.v, j.l.s.u) \in A_a$$

$$p_{S.0.0.0}^{S.1.t.v} \geq x_{S.0.0.0}^{S.1.t.v} \qquad \forall t, v \text{ s.t. } (S.0.0.0, S.1.t.v) \in A_a$$

To incorporate the flow variables **z** corresponding to the "Skip Task" arcs into MIP Formulation 2, consider the mission planner only skips task i at node $i.0.t.v$ if and only if $z_{i.0.t.v} = 1$.

This leads to the following two constraints:

$$x_{i.0.t.v}^{i.1.t.v} \leq z_{i.0.t.v} \qquad \forall i.0.t.v \in N_a | i.0 \neq E.0$$

$$\sum_{s>t|(i.0.t.v,i.1.s.u)\in A_a} x_{1.0.t.v}^{i.1.s.u} \leq 1 - z_{i.0.t.v} \qquad \forall i.0.t.v \in N_s$$

Likewise, the mission planner only uses "Mission End" arcs at node $i.1.t.v$ if and only if $z_{i.1.t.v} = 1$. The resulting constraints are:

$$\sum_{(i.1.t.v,E.1.T_{MAX}.N_{MAX})\in A_a} x_{i.1.t.v}^{E.1.T_{MAX}.N_{MAX}} \leq z_{i.1.t.v} \qquad \forall i.1.t.v \in N_s$$

$$\sum_{s,u,j\neq E|(i.1.t.v,j.0.s.u)\in A_a} x_{i.1.t.v}^{j.0.s.u} \leq 1 - z_{i.1.t.v} \qquad \forall i.1.t.v \in N_s$$

The final constraint in Formulation 2 establishes a threshold value t for each node of type $i.0.t.v$ in G_a such that route alteration is not permitted before t. The intuition is that the mission planner should not alter its route at time t if it does not alter its route at time $t+1$. As a result:

$$z_{i.k.t.v_1} \leq z_{i.k.t+1,v_2} \quad \forall\, i.k.t.v_1, i.k.t+1.v_2 \in N_a; v_1, v_2 \in \{0, T, \ldots, N_{MAX}\}$$

Objective Function

The generalized objected function for MIP Formulation 2 (Equation 3.13) only considers the flow variables relating to task execution (since the mission planner collects reward by successfully completing tasks), while ignoring the flow variables that dictate skipping a task. Thus, the summation restricts the time s to be strictly greater than time t in the flow variable $x_{i.0.t.v}^{i.1.s.u}$. The objective function in MIP Formulation 2 is:

$$\max \quad \sum_{i=1}^{n} \left(r_i \times \left(\sum_{t,v,s>t,u|(i.0.t.v,i.1.s.u)\in A_a} x_{i.0.t.v}^{i.1.s.u} \times f(i,t,v,s,u) \right) \right) \qquad (3.13)$$

Compiling the above constraints and objective function, MIP formulation 2 is as follows:

MIP Formulation 2

$$\max \quad \sum_{i=1}^{n} \left(r_i \times \left(\sum_{t,v,s>t,u|(i.0.t.v,i.1.s.u)\in A_a} x_{i.0.t.v}^{i.1.s.u} \times f(i,t,v,s,u) \right) \right)$$

s.t.

$$\mathbf{By} = \mathbf{d}$$

$$\mathbf{B_a x} \leq \mathbf{d_a}(\beta)$$

$$\sum_{t,v,s,u|(i.k.t.v,j.l.s.u)\in A_a} x_{i.k.t.v}^{j.l.s.u} \leq y_{i.k}^{j.l} \qquad \forall (i.k, j.l) \in A | j.l \neq E.0$$

$$p_{i.k.t.v}^{j.l.s.u} \times \left(\sum_{i'.k'.t'.v'|(i'.k'.t'.v',i.k.t.v)\in A_a} x_{i'.k'.t'.v'}^{i.k.t.v} \right) \geq x_{i.k.t.v}^{j.l.s.u} \qquad \forall (i.k.t.v, j.l.s.u) \in A_a$$

$$p_{S.0.0.0}^{S.1.t.v} \geq x_{S.0.0.0}^{S.1.t.v} \qquad \forall t, v \in A_a$$

$$x_{i.0.t.v}^{i.1.t.v} \leq z_{i.0.t.v} \qquad \forall i.0.t.v \in N_a | i.0 \neq E.0$$

$$\sum_{s>t|(i.0.t.v,i.1.s.u)\in A_a} x_{1.0.t.v}^{i.1.s.u} \leq 1 - z_{i.0.t.v} \qquad \forall i.0.t.v \in N_s$$

$$\sum_{(i.1.t.v,E.1.T_{MAX}.N_{MAX})\in A_a} x_{i.1.t.v}^{E.1.T_{MAX}.N_{MAX}} \leq z_{i.1.t.v} \qquad \forall i.1.t.v \in N_s$$

$$\sum_{s,u,j\neq E|(i.1.t.v,j.0.s.u)\in A_a} x_{i.1.t.v}^{j.0.s.u} \leq 1 - z_{i.1.t.v} \qquad \forall i.1.t.v \in N_s$$

$$z_{i.k.t.v_1} \leq z_{i.k.t+1,v_2} \qquad \forall\ i.k.t.v_1, i.k.t+1.v_2 \in N_a$$

$$0 \leq \mathbf{x} \leq 1$$

$$\mathbf{y} \in \{0,1\}$$

$$\mathbf{z} \in \{0,1\}$$

MIP Formulation 2 allows route alteration in the UUVMPP to mitigate the risk of reaching the end mission location after time T_{MAX}. There are a few significant drawbacks to this formulation. By the construction of the Route Alteration Graph G_a, the mission planner can only skip a task after it maneuvers to the task. Additionally, if the mission planner skips a task, then the mission planner cannot revisit the task later. MIP Formulation 2 also does not account for situations where the mission planner realizes best-case disturbances and has time to complete more tasks. Chapter 4 presents an algorithm that allows for complete route alteration on-line, including skipping tasks, performing additional tasks, re-ordering the sequence of the task route, and terminating the mission early.

3.3.3 Extendability

Cates [11] describes a method to augment the MIP formulations with additional resource constraints (*e.g.* energy, risk, stealth, etc.) using dual variables. Cates shows that adding the appropriate linear constraints to the formulations effectively models the resource consumption without significantly increasing the problem's complexity.

For example, consider the mission planner wants to account for energy; *i.e.* the mission planner desires to find a task route with MIP Formulation 1 or 2 that does not cause the UUV to exceed its maximum energy, E_{MAX}. Let $\mathbf{u}$ be the vector of *node potentials* in the stochastic graph (G_s for MIP Formulation 1, G_a for MIP Formulation 2), where the node potentials represent the maximum amount of energy the UUV can consume before reaching a given node. Then,

$$u_{E.1.T_{MAX}.N_{MAX}} \leq E_{MAX}$$

represents the constraint that the UUV cannot exceed its maximum energy before reaching the end mission location.

For a complete list of constraints and variables required to alter MIP Formulations 1 and 2, see [11].

3.4 ϵ-Rounding Heuristic

The ϵ-Rounding Heuristic is an effective algorithm that reduces the number of possible task sequences for the optimal UUV route. The heuristic solves a relaxed version of MIP Formulations 1 and 2 (*i.e.* $0 \leq \mathbf{x} \leq 1$, $0 \leq \mathbf{y} \leq 1$, and $0 \leq \mathbf{z} \leq 1$) called the Relaxed

Formulation, and uses the result to restrict the set of feasible solutions to the UUVMPP. The proposed heuristic finds all paths in the Decision Graph G that has a flow greater than ϵ in the Relaxed Formulation and selects the path that produces the greatest expected reward. Let P represent the path formed by the ϵ-Rounding Heuristic, where $P = \{S = i_1, i_2, \ldots, i_M = E\}$.

Begin:

Solve the Relaxed Formulation.

Let $U = \{P|P$ is a path from $S.1$ to $E.0$ in $\mathbf{y}$ with flow greater than ϵ, and is a feasible UUV route$\}$

Find $P_{MAX} \in U$ such that reward P_{MAX} is greater than or equal to all paths in U.

P_{MAX} is the path returned by the heuristic.

End

The ϵ-Rounding Heuristic can be interpreted in a separate but equivalent way. After solving the Relaxed Formulation, the heuristic eliminates the arcs $(i.k, j.l) \in A$ such that $y_{i.k}^{j.l} \leq \epsilon$ and the corresponding arcs $(i.k.t.v, j.l.s.u) \in A_s$. Then, the mission planner solves MIP Formulation 1 or 2 with the reduced-size network flow models. The intuition is if there is a small amount of flow over arcs in the Relaxed Formulation, then it is likely the arc is not included in the true optimal UUV task route.

By eliminating arcs in graphs G and G_s, the ϵ-Rounding Heuristic allows MIP Formulations 1 and 2 to consider fewer task routes when solving the UUVMPP. The correct choice of ϵ is critical to the performance of the heuristic; a larger ϵ eliminates more arcs in G and G_s which in turn improves the algorithm's run time, but it also produces an answer further from the optimal solution attained with exact formulations.

3.5 Experimental Setup

This section discusses the various assumptions for the simulations performed in the thesis. The UUVMPP simulations focus on the stochastic time and navigation expanded formulations, both with the exact algorithm and suggested heuristic.

3.5.1 Navigation Error Accumulation

The mission planning algorithm successfully incorporates navigation fixes into the mission-planning process. To simplify the construction of the network flow models for each simulation, the formulation generates mission inputs that assume accumulated navigation error is linear with time. Under this assumption, the term "accumulated navigation error" is identical to "time since the last navigation fix." It is important to note the formulations do not rely on the linearity between time and navigation error to solve the UUVMPP; the formulations would experience similar performance and run time results without the linearity assumption.

Consider the following example: if the mission planner takes its first navigation fix 7,200 seconds after the mission start time, the UUV's accumulated navigation error at that moment would also be 7,200 seconds. Once the mission planner performs the navigation fix, the vehicle's accumulated navigation error resets to 0 seconds. The navigation error then increases from 0 seconds in a 1-to-1 correspondence with time until the next navigation fix.

3.5.2 Navigation Fixes & Selection

The formulations can handle two distinct types of navigation fix locations: (1) safe navigation points and (2) safe navigation areas. A safe navigation point is a latitude/longitude coordinate on the map the mission planner identifies as inherently safe for the UUV to take a navigation fix, while a safe navigation area is any such region.

Each navigation fix is assumed to provide equal benefit by contributing zero reward to the objective function and resetting the accumulated navigation error to zero. If the algorithm expands to include multiple types of navigation fixes (*e.g.* map-matching and GPS), then the network flow models would have to be modified accordingly.

If safe navigation areas are present in the mission environment, the algorithm reduces each safe navigation area to a collection of safe navigation points. The below pseudo-code explains the procedure:

```
Begin:
For each pair of tasks $t_1$ and $t_2$
    If one of the tasks is inside a safe navigation area
        Place a new safe navigation point at that task.
    Else
        For each line segment $S_i$ bounding the safe navigation areas
            Find minimum distance, $d_i$, from $t_1 \rightarrow S_i \rightarrow t_2$.
        End
        Choose point that produces minimum distance, $d_i^* = \arg\min_i d_i$.
        Place a new safe navigation point at location $d_i^*$.
    End
End
```

The minimum distance from task t_1 to line segment S_i to task t_2 will either be one of the end points of line segment S_i or the point on S_i which creates an equal angle between each of the tasks and the line segment. Figure 3-17 illustrates this intuition; in this situation, the minimum distance from t_1 to S_i to t_2 will either be point X, Y, or Z, where Z is chosen to make equal angles θ. Although the minimum distance does not necessarily minimize elapsed time (since it does not consider tides, currents, etc.), this procedure provides a good estimate of the true best navigation fix location.

Figure 3-18 illustrates a two-task example containing a safe navigation area. The figure shows the original mission input and displays how the procedure reduces the safe navigation area to a collection of safe navigation points. The procedure generates six safe navigation points, since there are $\binom{4}{2} = \frac{4!}{2!*2!} = 6$ different pairs of task location combinations in a two-task example (including the start and end mission location). Note that all the created safe navigation points lie on the top-most line segment.

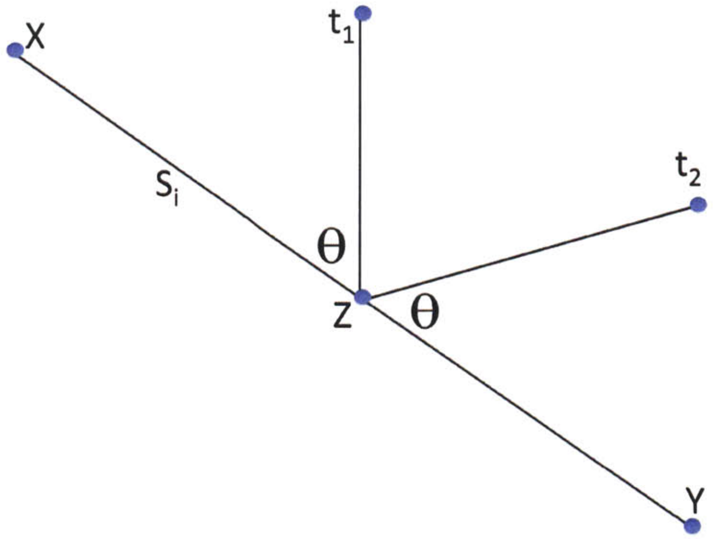


Figure 3-17: Safe Navigation Area Geometry. The goal is to find the point on line segment S_i that minimizes the distance from task t_1 to line segment S_i to task t_2. The solution will be one of the two end points of S_i (X or Y) or the point on S_i which creates an equal angle between each of the tasks and the line segment (Z).

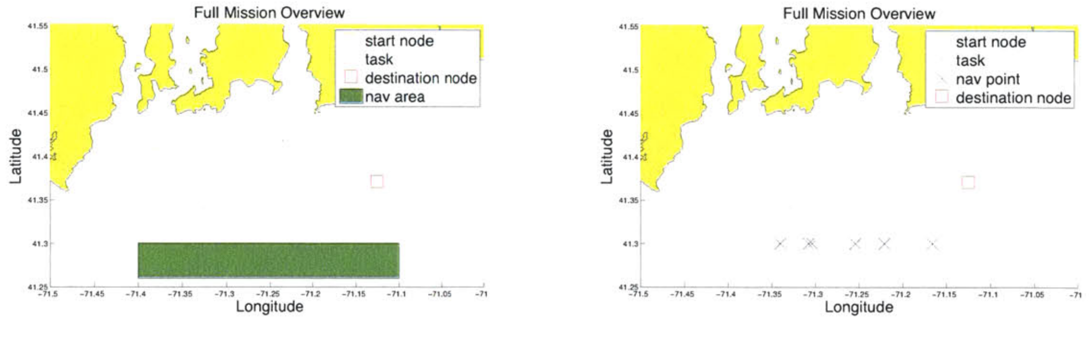


Figure 3-18: Safe Navigation Area Example. Note how the procedure reduces the safe navigation area to six safe navigation points, all lying on the top-most line segment.

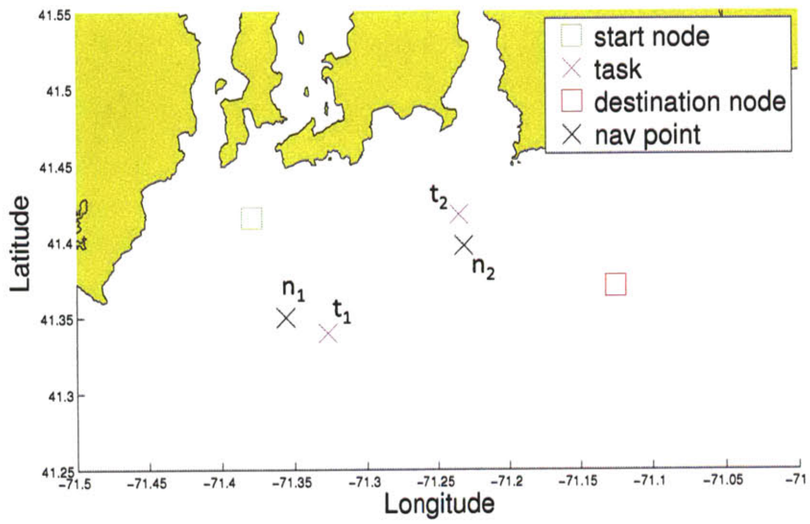


Figure 3-19: Navigation Fix Selection Example. Table 3.5 contains the optimal navigation fix location between each pair of tasks.

Once all safe navigation areas have been reduced to safe navigation points, the algorithm chooses the best navigation fix location between each pair of tasks for every time step *a priori*. When the mission planner has more than one navigation fix location to select from, the mission planner must decide which navigation fix is most prudent given the mission situation. Since each navigation fix is assumed to provide the same benefit to the overall mission and the UUV is constrained by time, the formulations assume the navigation fix point that causes the least penalty to the constraints is ideal.

Consider the example problem in Figure 3-19 which contains tasks $\langle t_1, t_2 \rangle$ and navigation fixes $\langle n_1, n_2 \rangle$. Table 3.5 displays the optimal navigation fix location between each pair of tasks should the mission planner decide to take a navigation fix.

From Task	To Task	Optimal Navigation Fix Location
S	t_1	n_1
S	t_2	n_2
S	E	n_2
t_1	t_2	n_2
t_1	E	n_2
t_2	t_1	n_2
t_2	E	n_2

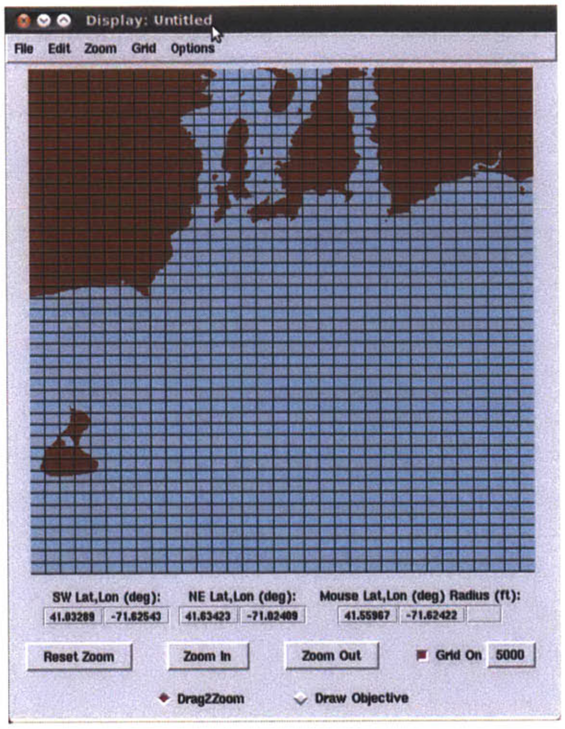
Table 3.5: Navigation Fix Selection, Two-Task Example. Refer to Figure 3-19.

3.5.3 Travel Time Planner

A user-developed Travel Time Planner aided the problem generation for the simulations contained within the thesis. The software allows the mission planner to place a start mission location, end mission location, tasks, safe navigation locations, safe navigation areas, and time-dependent no-go zones in a given mission environment. With this information, the Travel Time Planner simulates the tide and current data from a given start time and calculates the mean and standard deviation for travel times and energy usage between every pair of tasks. The software performs this calculation for every time step for the duration of the mission, automatically avoiding no-go zones and land masses.

The operating area for the randomly generated simulations in the thesis is the Naragansett Bay South of Rhode Island, United States, within the following latitude/longitude coordinates: (41.15°N, 71.4°W), (41.35°N, 71.4°W), (41.35°N, 71.2°W), (41.15°N, 71.2°W). Figure 3-20 illustrates the UUV's operating area on the Travel Time Planner.

Table 3.6 provides the Travel Time Planner's mean run time for one randomly generated mission as a function of the number of tasks. The table displays the results from running 100 random scenarios for each task value with two safe navigation fix points, T_{MAX} = 72,000 seconds, and T = 3,600 seconds.



SW Lat,Lon (deg): 41.03289 -71.62543
NE Lat,Lon (deg): 41.63423 -71.02409
Mouse Lat,Lon (deg) 41.55967 -71.62422 Radius (ft):
■ Grid On 5000

Figure 3-20: Snapshot of Travel Time Planner Software. The figure shows the operating area for the simulations in the thesis, the Naragansett Bay South of Rhode Island, United States.

Number of Tasks	Mean Run Time (seconds)
2	3.965×10^2
3	4.066×10^2
4	4.678×10^2
5	5.578×10^2
6	5.696×10^2
7	6.380×10^2
8	7.495×10^2
9	7.946×10^2
10	9.026×10^2
15	1.188×10^3
20	1.556×10^3
25	1.875×10^3
30	2.222×10^3

Table 3.6: Travel Time Planner Run Time

3.5.4 Discretization

The Time and Navigation Expanded Stochastic Graph relies on discretized problem inputs to solve MIP Formulations 1 and 2. This section describes the assumptions made for the resolution of this problem (time step) and the discretization of continuous random variables within the network flow model for both UUV travel and task execution times.

Resolution

The resolution of the model depends on the mission planner's choice of time step T. A small T produces a finer resolution (as $T \rightarrow 0$ the model approaches the continuous time model), while a large T represents a coarser resolution.

The choice of T directly affects the accuracy and complexity of the model. As T becomes extremely small, the optimal solution of the discretized model approaches the solution to the continuous time model. Reducing T adds more nodes and arcs in the formulation to represent a similar mission environment, which conversely affects run time and potential mission complexity. Table 3.7 illustrates how the number of time steps affect the number of nodes and arcs in a two-task example represented by Figure 3-21; T_{MAX} = 72,000 seconds and N_{MAX} = 36,000 seconds are held constant while T varies to adjust the number of time steps in the model. The network flow model could not be built with current software with T equal to 60 and 300 seconds, and thus the number of nodes and arcs for these T values in Table 3.7 are estimates.

In order to balance accuracy and model complexity, the thesis considers T = 3,600 seconds (*i.e.* one hour) an acceptable choice for the time step to permit the solving of larger, more interesting problems with longer mission duration T_{MAX}. Unless otherwise specified, all simulations are run with T = 3,600 seconds.

Arc Distributions

The simulations in the thesis assume that the travel times and task execution distributions are normally distributed (truncated to disallow negative duration). Since MIP Formula-

Time Step (sec)	# Of Time Steps	# Of Nodes	# Of Arcs
60	1201	$\sim 1.329 \times 10^6$	$\sim 1.387 \times 10^8$
300	241	$\sim 1.772 \times 10^5$	$\sim 6.936 \times 10^6$
600	121	4.429×10^4	9.908×10^5
1200	61	1.135×10^4	1.498×10^5
1800	41	5.168×10^3	5.258×10^4
3600	21	1.388×10^3	9.653×10^3
7200	11	3.98×10^2	2.163×10^3
14400	6	1.10×10^2	5.44×10^2

Table 3.7: Time Step vs. Model Characteristics for example problem in Figure 3-21.

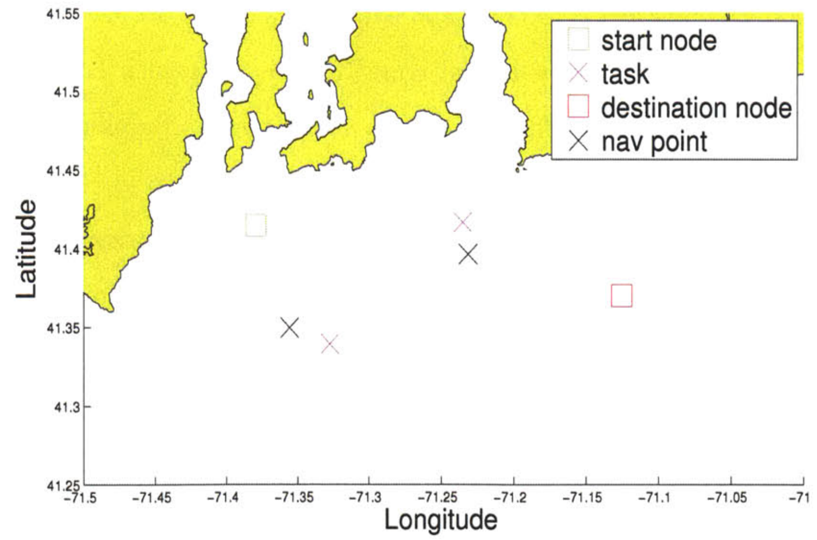


Figure 3-21: Time Step Resolution Example. Refer to Table 3.7.

tions 1 and 2 require discrete probability distribution inputs, there is an additional step to transform the continuous random variables into an acceptable format.

To discretize the continuous random variable for travel time between the pair of tasks i and j at time t, let:

- $\omega^j_{i.t}$ be the continuous random variable for travel time between tasks i and j at time t.
- $\bar{\omega}^j_{i.t}$ be the discrete random variable to be derived from $\omega^j_{i.t}$.
- $\mu^j_{i.t}$ be the mean travel time between tasks i and j at time t.
- $\sigma^j_{i.t}$ be the standard deviation of travel time between tasks i and j at time t.
- x_{MIN} be the minimum time for UUV transit given $\mu^j_{i.t}$ and $\sigma^j_{i.t}$.
- x_{MAX} be the maximum time for UUV transit given $\mu^j_{i.t}$ and $\sigma^j_{i.t}$.
- T be the time step.
- r be the radius.

The radius is analogous to the spread of the discretized distribution, where a larger r means a greater number of arcs are needed to represent the continuous distribution (*i.e.* bigger spread). For example, a radius of two forces the continuous distribution to be represented by four discrete pulses, two on either side of the mean $\mu^j_{i.t}$.

Then:

$$\Pr\{\bar{\omega}^j_{i.t} = x\} = \begin{cases} \Pr\{\omega^j_{i.t} \leq T * (\lfloor \frac{\mu^j_{i.t}}{T} \rfloor - r + \frac{1}{2})\}, & x = x_{MIN} \\ \Pr\{x - \frac{T}{2} \leq \omega^j_{i.t} \leq x + \frac{T}{2}\}, & x_{MIN} < x < x_{MAX} \\ 1 - \Pr\{\omega^j_{i.t} \leq T * (\lfloor \frac{\mu^j_{i.t}}{T} \rfloor + r - \frac{1}{2})\}, & x = x_{MAX} \\ 0, & \text{otherwise} \end{cases}$$

Where:

- $x_{MIN} = \max\{0, T * (\lfloor \frac{\mu^j_{i.t}}{T} \rfloor - r)\}$
- $x_{MAX} = T * (\lfloor \frac{\mu^j_{i.t}}{T} \rfloor + r)$

- $\bar{\omega}^j_{i.t} \in \{x_{MIN}, T * x_{MIN}, 2T * x_{MIN}, \ldots, x_{MAX}\}$.

The above methodology transforms the continuous random variable representing travel time into a discrete random variable that only takes values that are multiples of the time step; the task execution random variables can be discretized in a similar fashion. The simulations in the thesis set the radius for the travel time and task execution distributions equal to two. Figure 3-22 illustrates a discretization example where $\omega^j_{i.t}$ is normally distributed with $\mu^j_{i.t}$ = 8200 seconds and $\sigma^j_{i.t}$ = 2050 seconds. With a radius of two, the discrete probability mass function $\bar{\omega}^j_{i.t}$ has pulses at $x = \{3600, 7200, 10800, 14400\}$.

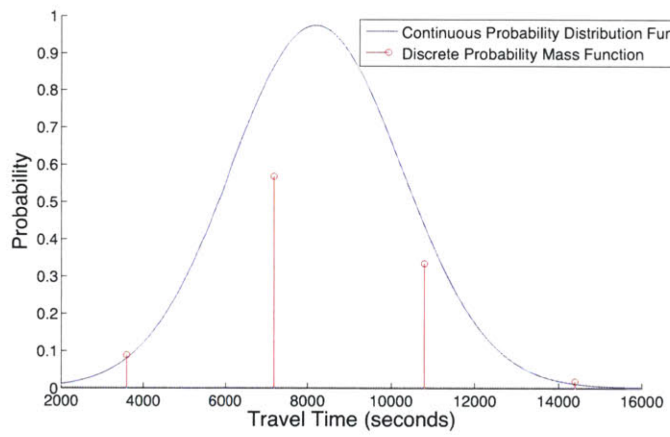


Figure 3-22: Travel Time Discretization Example, where $\omega^j_{i.t}$ is normally distributed with $\mu^j_{i.t}$ = 8200 seconds and $\sigma^j_{i.t}$ = 2050 seconds. With a radius of two, the discrete probability mass function $\bar{\omega}^j_{i.t}$ has pulses at $x = \{3600, 7200, 10800, 14400\}$.

3.5.5 Scenarios

This section presents eight diverse scenarios that demonstrate the flexibility of MIP Formulation 1. In each UUVMPP mission, the model determines *a priori* the task sequence that maximizes the expected reward. The following parameters remain constant for all scenarios:

- Confidence level, $\beta = 0.9$.

- Reward for completing each task, $r_i = 1 \; \forall \; i \in \{1, 2, \ldots, n\}$.

THIS PAGE INTENTIONALLY LEFT BLANK

Scenario 1

(a) Mission Parameters

# of tasks	2
# of navigation points	1
# of navigation areas	0
# of avoidance zones	0
T_{MAX}	41 time steps
N_{MAX}	41 time steps
T	3,600 seconds
Objective function	Full Reward

(b) Task Parameters

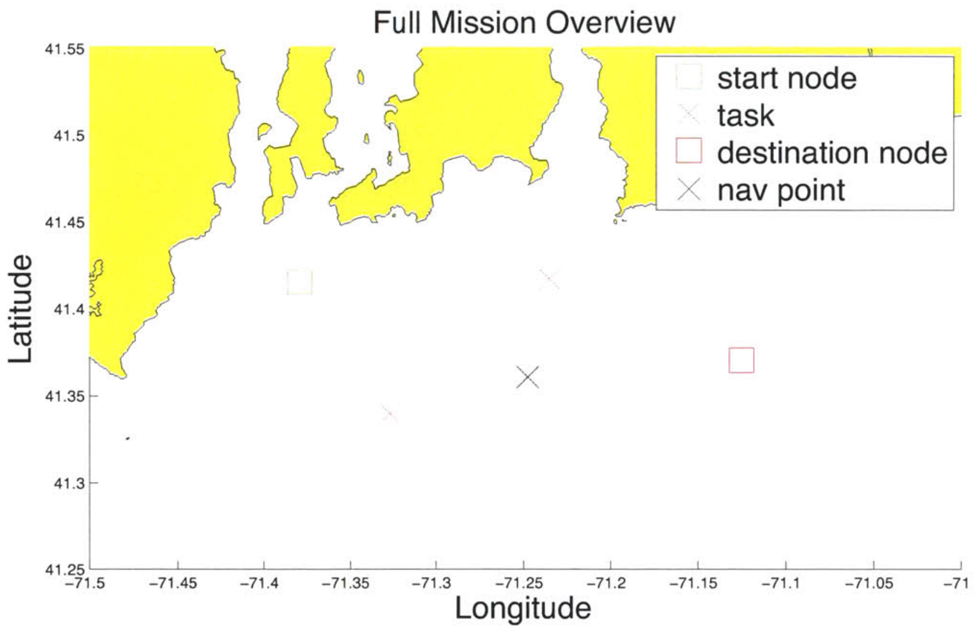
Task #	Mean execution	Fix?
1	3,600 seconds	no
2	3,600 seconds	no

(c) Scenario 1 Statistics

Optimal route	$S \xrightarrow{\text{fix}} 1 \xrightarrow{\text{direct}} 2 \xrightarrow{\text{direct}} E$
Objective value	2.0
# of arcs	91845
# of nodes	10088
Scenario 1 run time	43.3 seconds
Route expected completion time	13 time steps = 46,800 seconds
Route worst case completion time	19 time steps = 68,400 seconds

Table 3.8: Scenario 1 Set-Up and Results

Scenario 1 is a two-task problem where the UUV is given a substantial amount of time and is permitted to accumulate a large amount of navigation error (*i.e.* under-constrained). Since the scenario uses the Full Reward objective function (Equation 3.10), the solution has multiple optimal solutions; any task sequence that completes both tasks will produce an objective function value of 2.0. Every task sequence is feasible due to the loose time and accumulated navigation error constraints and will be completed with probability one, so the formulation arbitrarily chooses the task sequence $S \xrightarrow{\text{fix}} 1 \xrightarrow{\text{direct}} 2 \xrightarrow{\text{direct}} E$ as optimal.



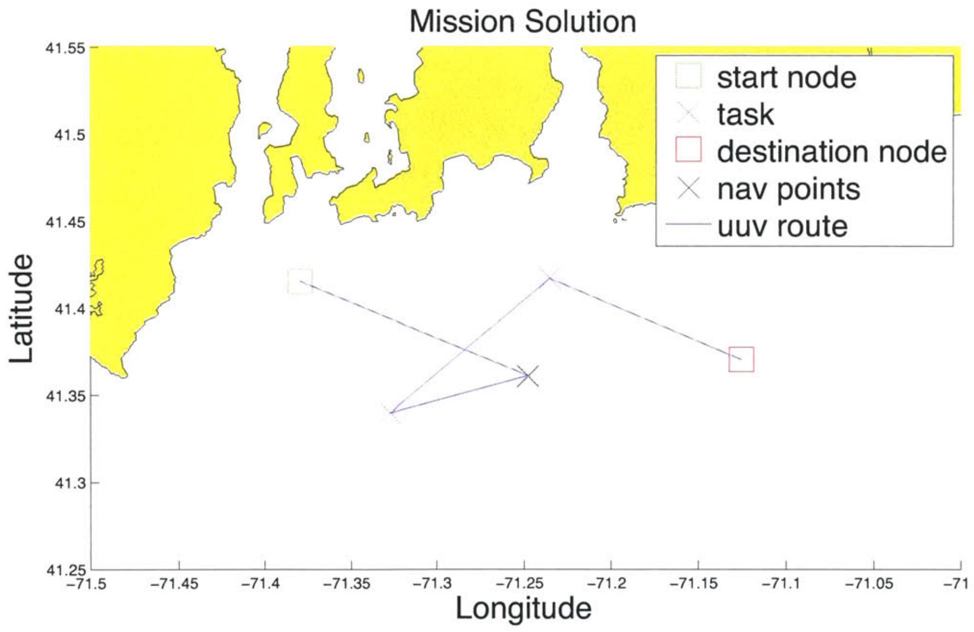


Figure 3-23: Scenario 1 Results

Scenario 2

(a) Mission Parameters

# of tasks	2
# of navigation points	1
# of navigation areas	0
# of avoidance zones	0
T_{MAX}	41 time steps
N_{MAX}	9 time steps
T	3,600 seconds
Objective function	Full Reward

(b) Task Parameters

Task #	Mean execution	Fix?
1	3,600 seconds	no
2	3,600 seconds	no

(c) Scenario 2 Statistics

Optimal route	$S \xrightarrow{\text{direct}} 2 \xrightarrow{\text{fix}} 1 \xrightarrow{\text{fix}} E$
Objective value	2.0
# of arcs	16940
# of nodes	2216
Scenario 2 run time	4.8 seconds
Route expected completion time	14 time steps = 50,400 seconds
Route worst case completion time	20 time steps = 72,000 seconds

Table 3.9: Scenario 2 Set-Up and Results

Scenario 2 is identical to Scenario 1 except the UUV is restricted in the amount of navigation error it can accumulate. Since the UUV has ample time to complete both tasks within T_{MAX}, the mission planner chooses to take two navigation fixes to sufficiently satisfy the restriction on N_{MAX}.

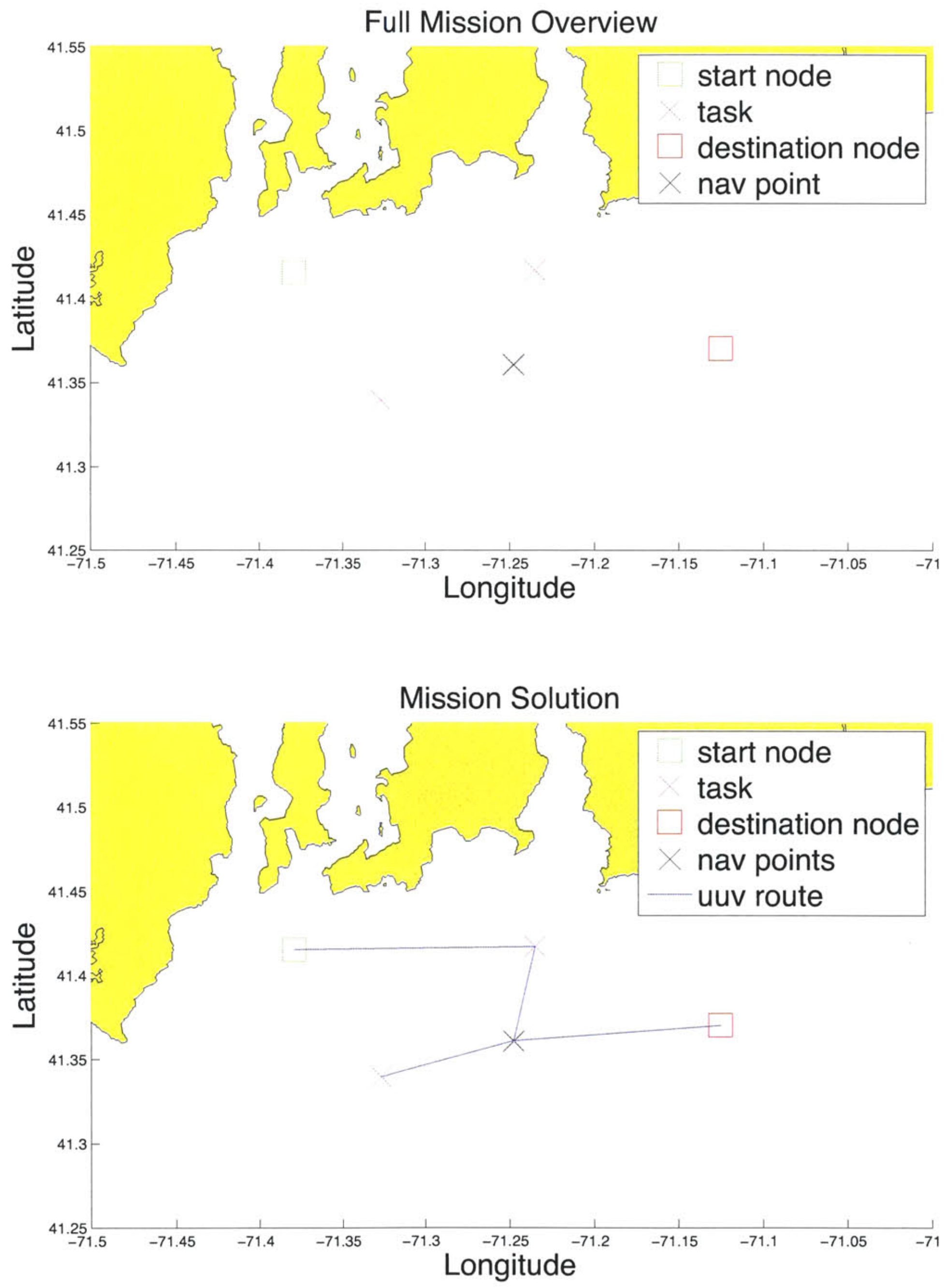


Figure 3-24: Scenario 2 Results

Scenario 3

(a) Mission Parameters

# of tasks	2
# of navigation points	1
# of navigation areas	0
# of avoidance zones	0
T_{MAX}	11 time steps
N_{MAX}	9 time steps
T	3,600 seconds
Objective function	Full Reward

(b) Task Parameters

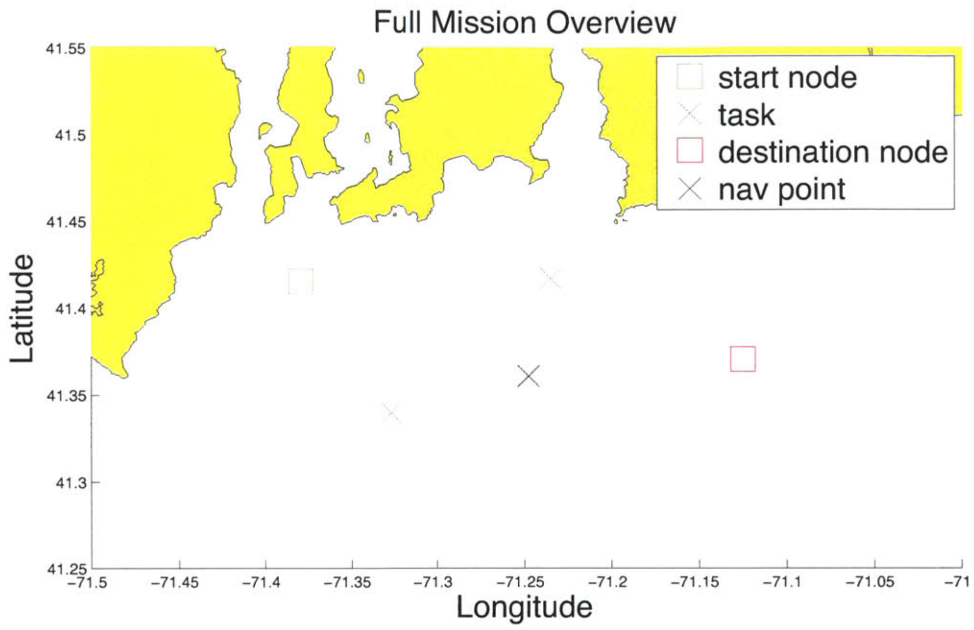
Task #	Mean execution	Fix?
1	3,600 seconds	no
2	3,600 seconds	no

(c) Scenario 3 Statistics

Optimal route	$S \xrightarrow{\text{direct}} 1 \xrightarrow{\text{fix}} E$
Objective value	1.0
# of arcs	2965
# of nodes	596
Scenario 3 run time	0.9 seconds
Route expected completion time	8 time steps = 28,800 seconds
Route worst case completion time	10 time steps = 36,000 seconds

Table 3.10: Scenario 3 Set-Up and Results

Scenario 3 contains the same task locations as the first two scenarios, where now the UUV has a tight restriction on both total time and accumulated navigation error. Due to these constraints, the mission planner determines the UUV cannot complete both tasks within T_{MAX}; in addition, the mission planner must take a navigation fix to satisfy the restriction on N_{MAX}. Task sequences $S \xrightarrow{\text{direct}} 1 \xrightarrow{\text{fix}} E$ and $S \xrightarrow{\text{direct}} 2 \xrightarrow{\text{fix}} E$ are both feasible solutions and have the same objective function value, 1.0. Since the expected completion for $S \xrightarrow{\text{direct}} 1 \xrightarrow{\text{fix}} E$ is 3,220 seconds quicker than $S \xrightarrow{\text{direct}} 2 \xrightarrow{\text{fix}} E$, the mission planner chooses $S \xrightarrow{\text{direct}} 1 \xrightarrow{\text{fix}} E$ to increase the likelihood of successful mission completion.



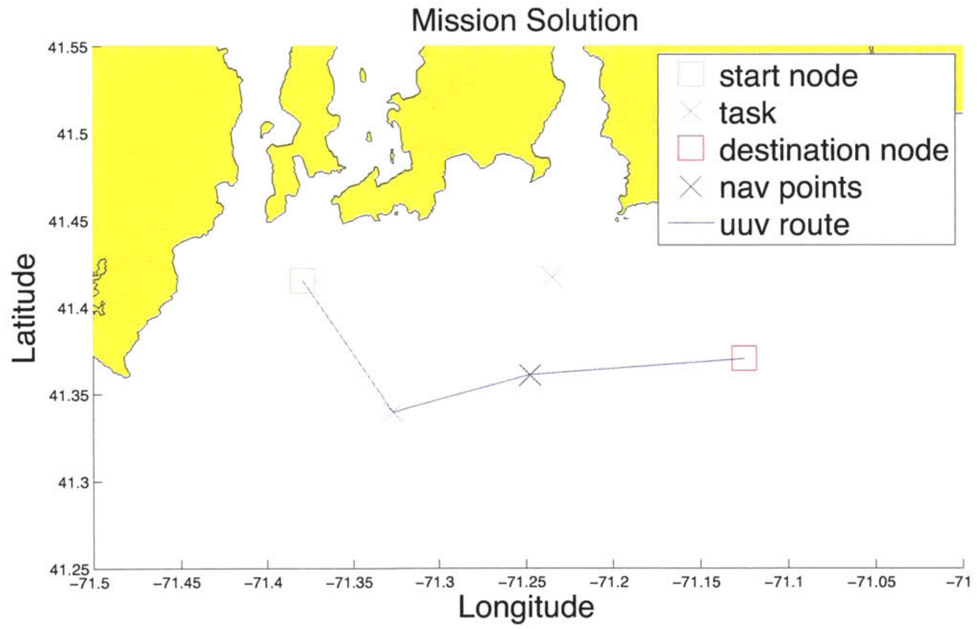


Figure 3-25: Scenario 3 Results

Scenario 4

(a) Mission Parameters

# of tasks	2
# of navigation points	2
# of navigation areas	0
# of avoidance zones	0
T_{MAX}	11 time steps
N_{MAX}	9 time steps
T	3,600 seconds
Objective function	Full Reward

(b) Task Parameters

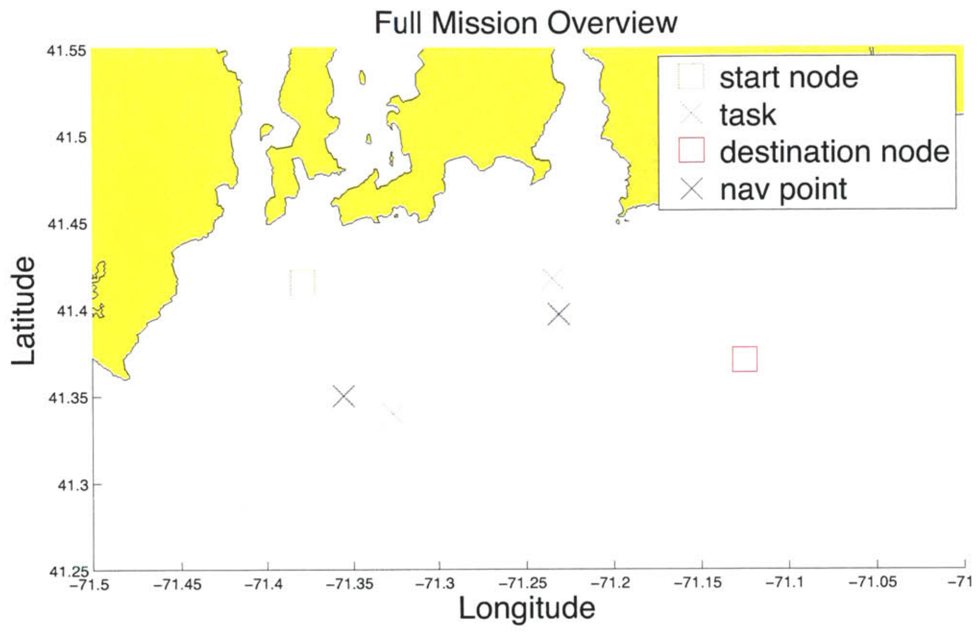
Task #	Mean execution	Fix?
1	3,600 seconds	no
2	3,600 seconds	no

(c) Scenario 4 Statistics

Optimal route	$S \xrightarrow{\text{direct}} 1 \xrightarrow{\text{fix}} E$
Objective value	1.0
# of arcs	3019
# of nodes	596
Scenario 4 run time	0.8 seconds
Route expected completion time	8 time steps = 28,800 seconds
Route worst case completion time	9 time steps = 32,400 seconds

Table 3.11: Scenario 4 Set-Up and Results

Scenario 4 introduces two navigation fixes into the mission environment with the same task locations. The optimal route remains $S \xrightarrow{\text{direct}} 1 \xrightarrow{\text{fix}} E$ which achieves an objective function value of 1.0. Although there is a navigation fix location closer to task 1, the planner chooses to take the navigation fix at the location between task 1 and the end mission location since the UUV does not have to deviate too far from track to execute the navigation fix. As a result, this decision minimizes the expected route completion time and thus increases the mission planner's confidence in successful mission completion.



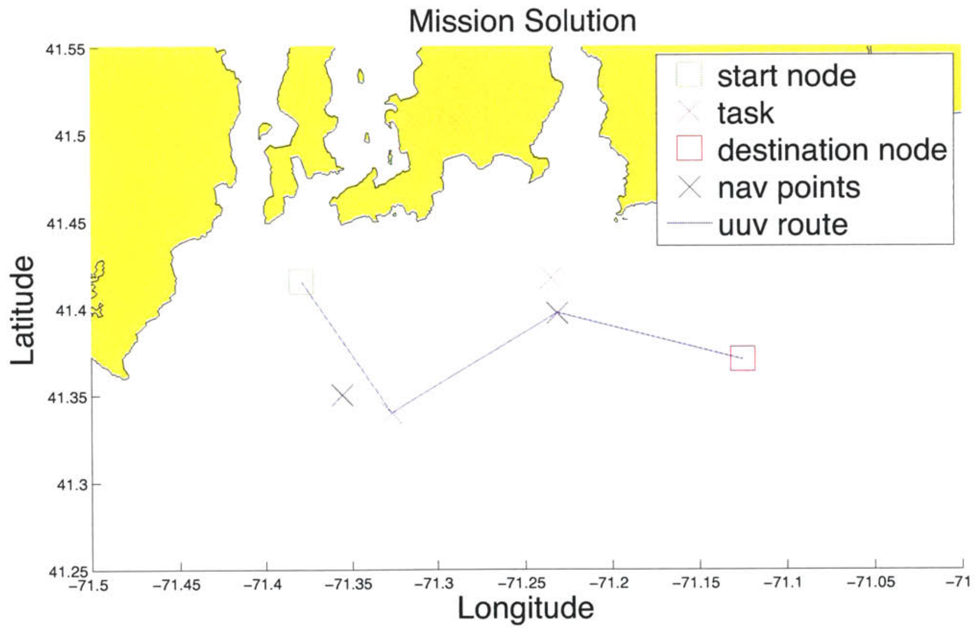


Figure 3-26: Scenario 4 Results

Scenario 5

(a) Mission Parameters

# of tasks	2
# of navigation points	2
# of navigation areas	0
# of avoidance zones	0
T_{MAX}	11 time steps
N_{MAX}	9 time steps
T	3,600 seconds
Objective function	Linear Decrease

(b) Task Parameters

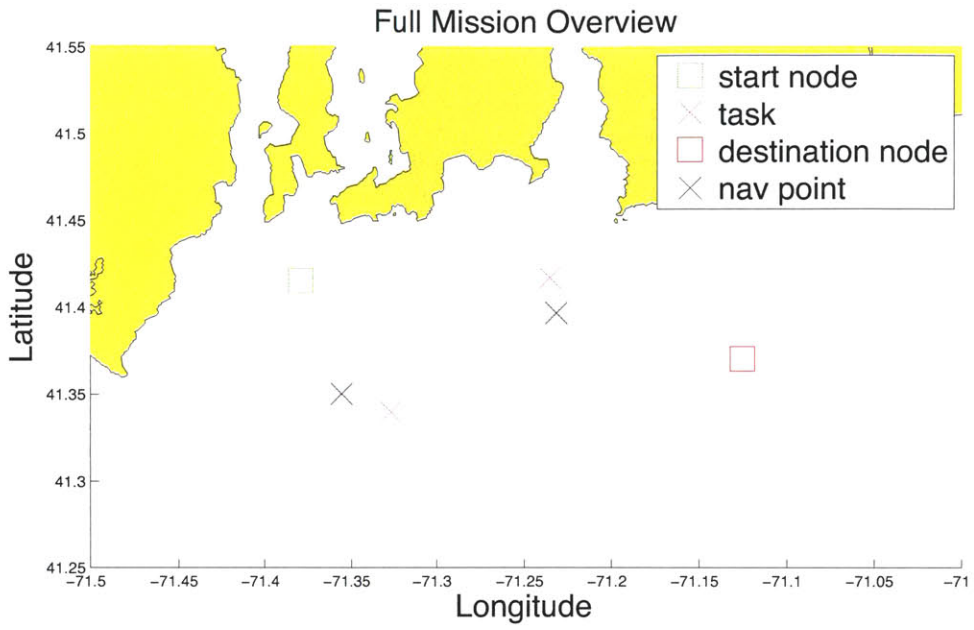
Task #	Mean execution	Fix?
1	7,200 seconds	no
2	7,200 seconds	no

(c) Scenario 5 Statistics

Optimal route	$S \xrightarrow{\text{fix}} 1 \xrightarrow{\text{fix}} E$
Objective value	0.875
# of arcs	3073
# of nodes	596
Scenario 5 run time	0.9 seconds
Route expected completion time	8 time steps = 28,800 seconds
Route worst case completion time	11 time steps = 39,600 seconds

Table 3.12: Scenario 5 Set-Up and Results

Scenario 5 contains tasks with longer expected duration; additionally, the algorithm incorporates the Linear Decrease objective function (Equation 3.11). Because the UUV is incentivized to take a navigation fix prior to executing a task, the mission planner chooses to take a navigation fix between the start mission location and task 1. Since the UUV can only confidently execute one of the two tasks, the mission planner decides task 1 achieves the greatest expected mission utility.



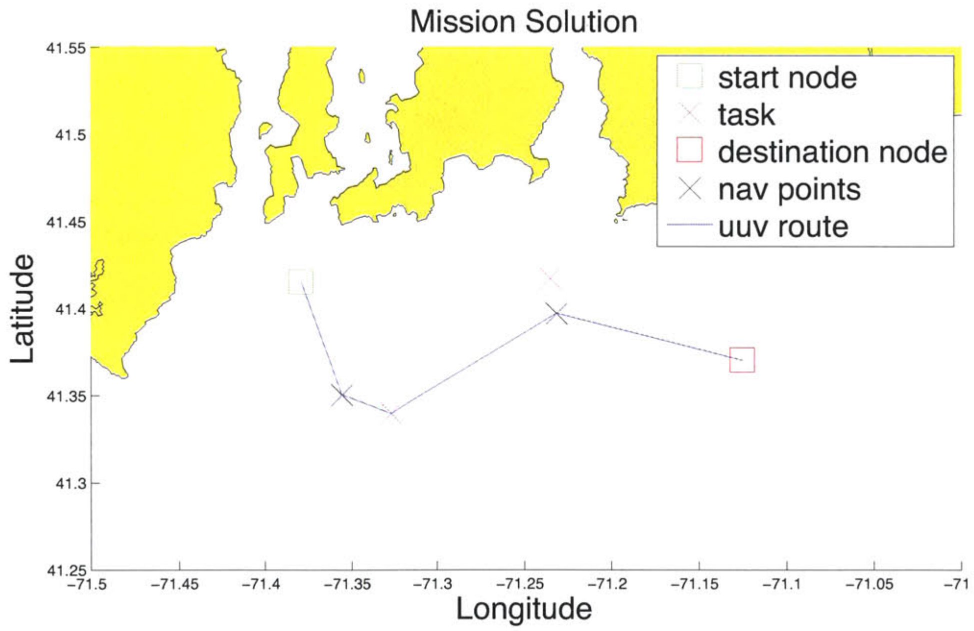


Figure 3-27: Scenario 5 Results

Scenario 6

(a) Mission Parameters

# of tasks	2
# of navigation points	2
# of navigation areas	0
# of avoidance zones	0
T_{MAX}	11 time steps
N_{MAX}	9 time steps
T	3,600 seconds
Objective function	Linear Decrease

(b) Task Parameters

Task #	Mean execution	Fix?
1	7,200 seconds	yes
2	7,200 seconds	yes

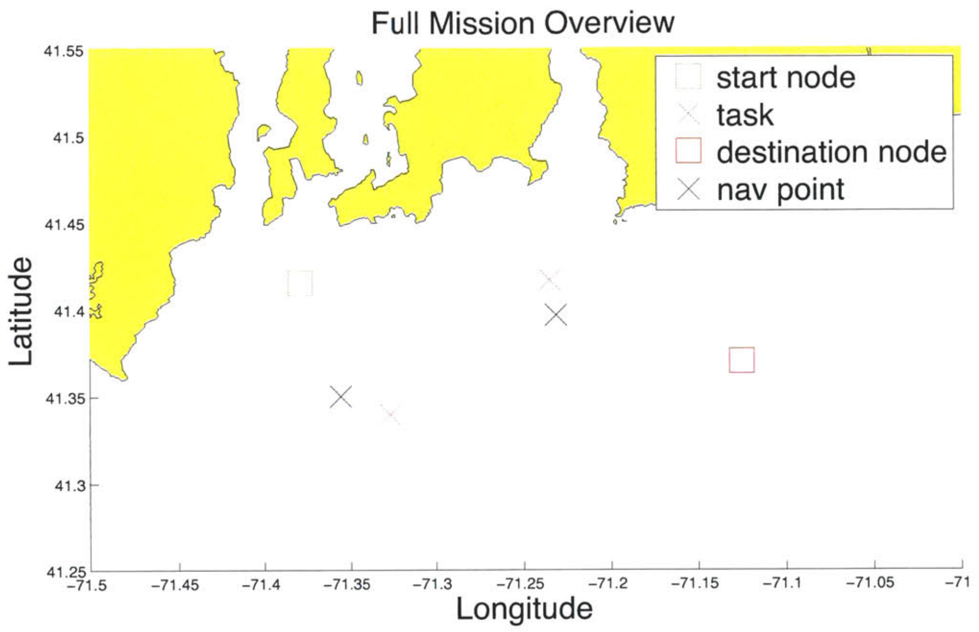
(c) Scenario 6 Statistics

Optimal route	$S \xrightarrow{\text{direct}} 1 \text{ with fix} \xrightarrow{\text{fix}} E$
Objective value	1.0
# of arcs	3183
# of nodes	596
Scenario 6 run time	1.0 seconds
Route expected completion time	7 time steps = 25,200 seconds
Route worst case completion time	9 time steps = 32,400 seconds

Table 3.13: Scenario 6 Set-Up and Results

In Scenario 6, the UUV can take a navigation fix while completing tasks. The UUV is given enough time to complete one of the two tasks, so the mission planner chooses to execute task 1 to minimize the total time of the mission. Since the UUV has ample time to reach the end mission location within T_{MAX}, the mission planner decides to take a navigation fix between task 1 and the end to increase the confidence in not exceeding N_{MAX}.

The mission planner greatly values the tasks with navigation fix capabilities for two reasons: (1) the mission planner receives the full task reward, and (2) the UUV's accumulated navigation error resets to zero without adding any time to the mission route.



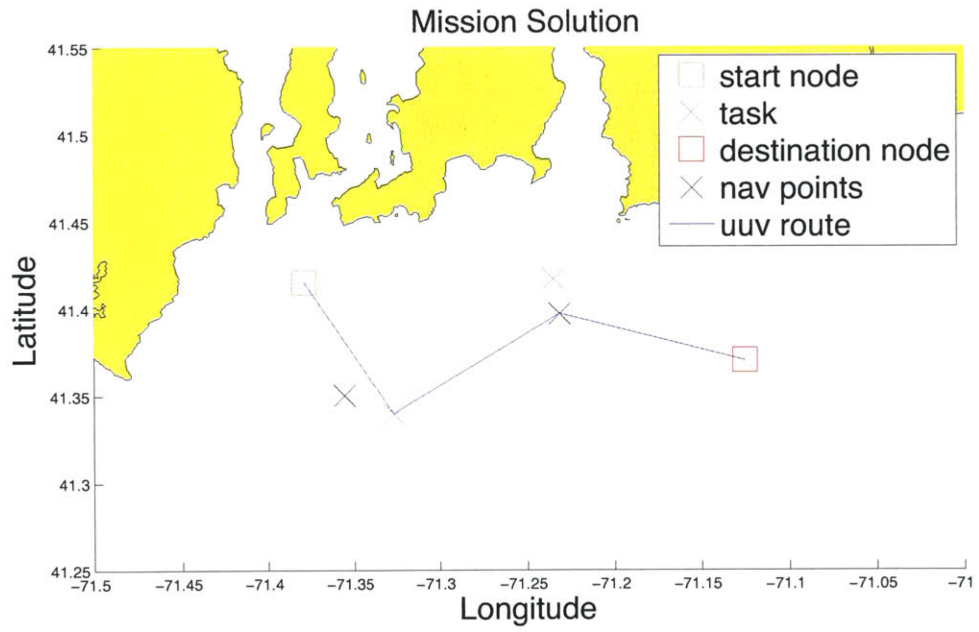


Figure 3-28: Scenario 6 Results

Scenario 7

(a) Mission Parameters

# of tasks	2
# of navigation points	0
# of navigation areas	2
# of avoidance zones	1
T_{MAX}	15 time steps
N_{MAX}	11 time steps
T	3,600 seconds
Objective function	Linear Decrease

(b) Task Parameters

Task #	Mean execution	Fix?
1	7,200 seconds	no
2	7,200 seconds	no

(c) Scenario 7 Statistics

Optimal route	$S \xrightarrow{\text{direct}} 1 \xrightarrow{\text{fix}} E$
Objective value	0.77546
# of arcs	6858
# of nodes	1058
Scenario 7 run time	1.6 seconds
Route expected completion time	11 time steps = 39,600 seconds
Route worst case completion time	14 time steps = 50,400 seconds

Table 3.14: Scenario 7 Set-Up and Results

Scenario 7 illustrates a two-task example with the presence of an avoidance zone and safe navigation area. Since the avoidance zone inhibits the UUV from traveling directly from task 1 to the end, the mission planner chooses to go around the avoidance zone instead and simultaneously collect a navigation fix along the way. Because there is no longer a navigation fix location en route to task 1, the mission planner chooses to complete task 1 first instead of traveling to the safe navigation area and then returning to task 1. In this scenario, the UUV only has enough time to complete one task, and the mission planner chooses task 1 since it is closer to the safe navigation area.

It is important to note the addition of avoidance zones and safe navigation areas to the mission environment does not increase the model's complexity. Avoidance zones and safe navigation areas alter the travel time distributions, but they do not add any arcs or nodes to the network flow model. As a result, the algorithm run time is of a similar order as other scenarios with the same number of tasks.

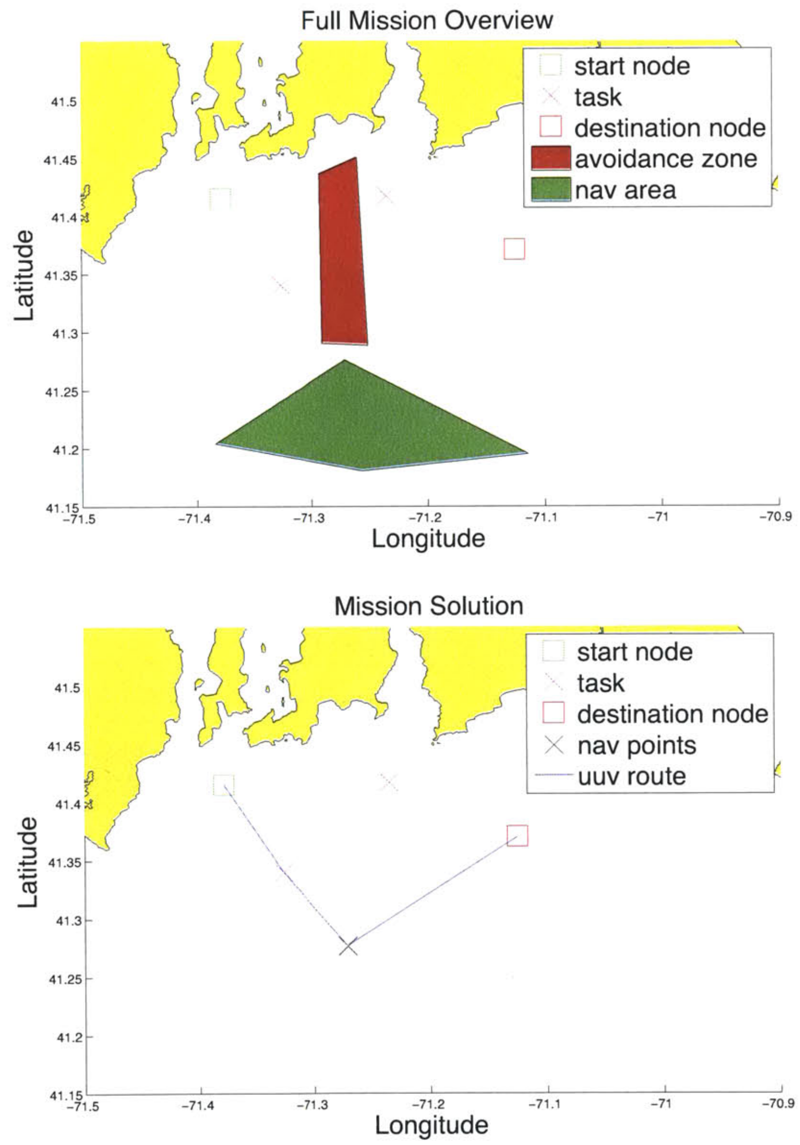


Figure 3-29: Scenario 7 Results

Scenario 8

(a) Mission Parameters

# of tasks	8
# of navigation points	1
# of navigation areas	2
# of avoidance zones	0
T_{MAX}	21 time steps
N_{MAX}	11 time steps
T	3,600 seconds
Objective function	Linear Decrease

(b) Task Parameters

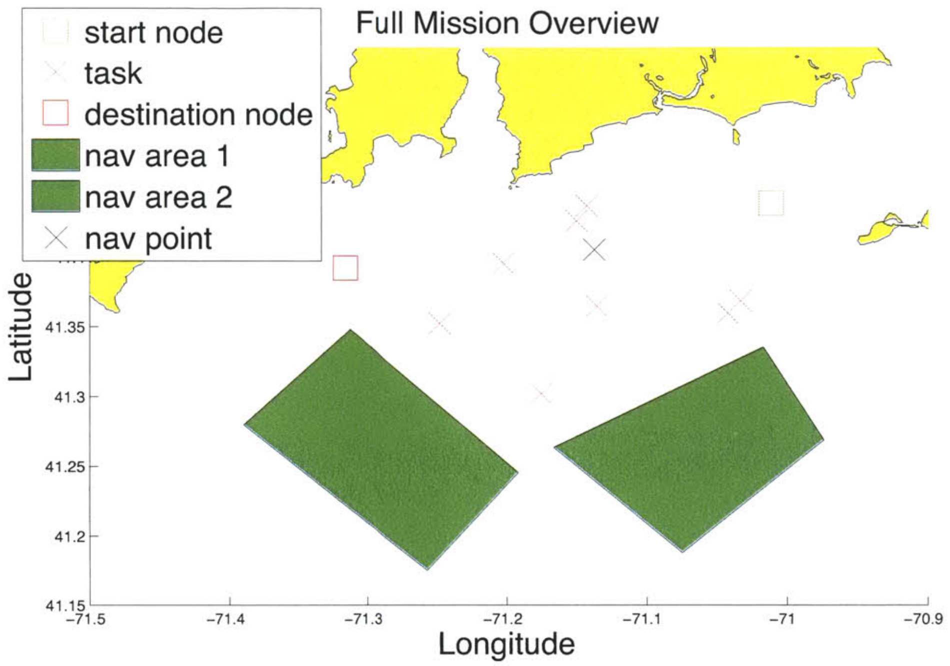
Task #	Mean execution	Fix?
1	7,200 seconds	yes
2	7,200 seconds	no
3	7,200 seconds	yes
4	7,200 seconds	yes
5	7,200 seconds	no
6	7,200 seconds	yes
7	7,200 seconds	no
8	7,200 seconds	no

(c) Scenario 8 Statistics

Optimal route	$S \xrightarrow{\text{direct}} 1$ with fix $\xrightarrow{\text{fix}} 3$ with fix $\xrightarrow{\text{direct}}$ 4 with fix $\xrightarrow{\text{fix}} 6$ with fix $\xrightarrow{\text{fix}} E$
Objective value	4.0
# of arcs	94725
# of nodes	4160
Scenario 8 run time	19.4 hours
Route expected completion time	18 time steps = 64,800 seconds
Route worst case completion time	22 time steps = 79,200 seconds

Table 3.15: Scenario 8 Set-Up and Results

Scenario 8 demonstrates MIP Formulation 1's ability to handle a diverse mission environment. The algorithm highly values tasks in which a navigation fix can be taken during execution; since the UUV only has enough time to complete four tasks, the mission planner chooses all four tasks of this type. The mission planner chooses to take three navigation fixes during the mission plan, but only if the UUV does not have to deviate too far from its intended course.



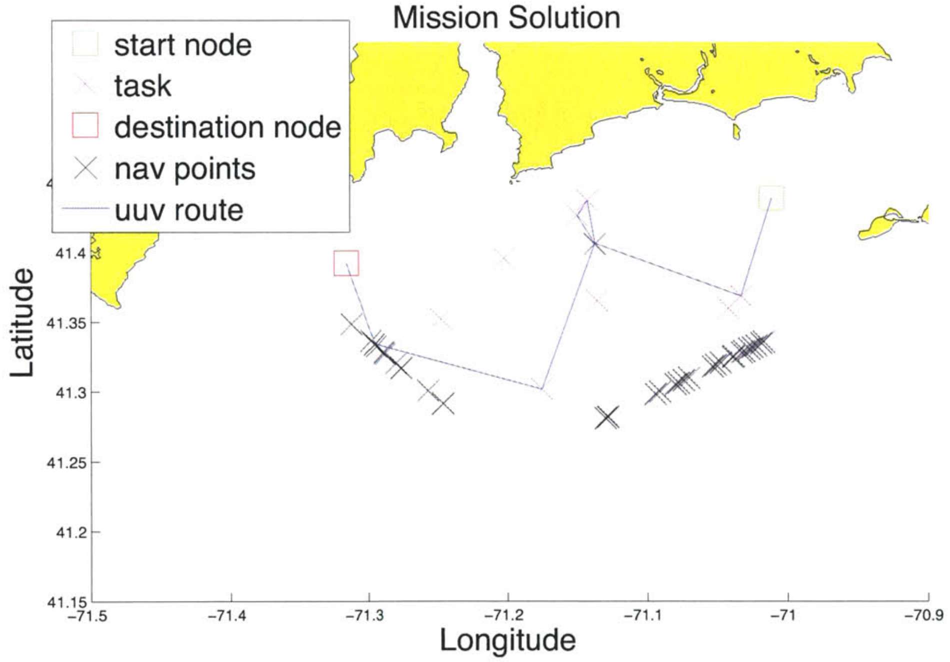


Figure 3-30: Scenario 8 Results

# of tasks	variable
# of navigation points	2
# of navigation areas	0
# of avoidance zones	0
T_{MAX}	21 time steps
N_{MAX}	11 time steps
T	3,600 seconds
Objective function	Linear Decrease

Table 3.16: MIP Formulation 1 Simulation Parameters

3.6 Results

The results section tests the size and scalability of the model for increasingly complex mission scenarios. The various simulations demonstrate the performance of MIP Formulation 1 with the exact algorithm and the ϵ-Rounding Heuristic, including a run time analysis and a characterization of the optimal solution. The algorithms were built in Java Eclipse with ILOG AMPL CPLEX System Version 11.0 and tested on a Quad Core Intel Xeon E5687 with 3.60GHz, 6.4 GT/s, and 48GB of RAM.

3.6.1 MIP Formulation 1

MIP Formulation 1 enables the mission planner to find the task sequence *a priori* that optimizes some mission objective with a β percent chance of reaching the end mission location without exceeding operational constraints. Table 3.16 contains the parameters for the simulations in this section, where the values attained for each task number are the average of 100 randomly generated scenarios. The mission locations for each scenario are randomly placed inside the Naragansett Bay operating area described in Section 3.5.3.

Run Time Analysis

Figure 3-31 illustrates the run time for MIP Formulation 1 with the exact algorithm on a logarithmic y-axis. Note the exponential increase in computation speed for a linear increase

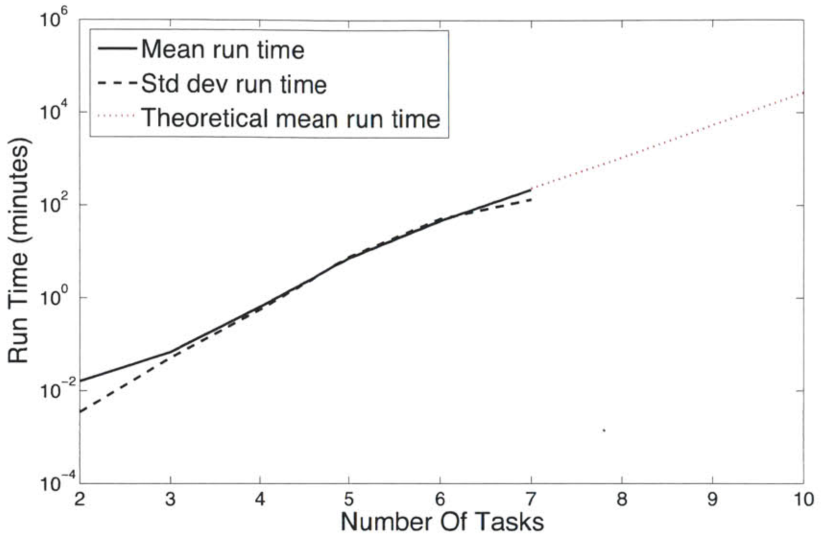


Figure 3-31: Run Time Results – MIP Formulation 1 Exact. Note the exponential increase in run time for a linear increase in number of tasks, where the values for $n \geq 8$ are given as theoretical values.

in number of tasks, where the values for $n \geq 8$ are given as theoretical values. Consistent with Cates [11], the number of tasks and number of time steps are the most important factors in determining the algorithm's run time.

The ϵ-Rounding Heuristic eliminates potential task routes that have less than an ϵ percent chance of being taken with the relaxed formulation. Figure 3-32 presents the mean and standard deviation of run times using the heuristic on a logarithmic y-axis, where the values for $n \geq 8$ are given as theoretical values. The figure shows the results with ϵ chosen from the set $\{0.01, 0.001, 0.0001, 0.00001\}$ (*i.e.* $\epsilon \in \{1{\times}10^{-2}, 1{\times}10^{-3}, 1{\times}10^{-4}, 1{\times}10^{-5}\}$). Unlike Cates, the ϵ-Rounding Heuristic produces similar computation and performance results for varying values of ϵ's ranging from $1{\times}10^{-2}$ to $1{\times}10^{-5}$. For the remainder of this section, the ϵ-Rounding Heuristic results will be reported with the singular value $\epsilon = 1 \times 10^{-2}$.

Figure 3-33 summarizes the run time results of both algorithms with MIP Formulation 1. Figure 3-33(a) presents the run times for the algorithms on a logarithmic y-axis, and Figure 3-33(b) uses Amdahl's Law [2] to display the speed up of the ϵ-Rounding Heuristic compared to the exact algorithm. On average, the ϵ-Rounding Heuristic's computation time

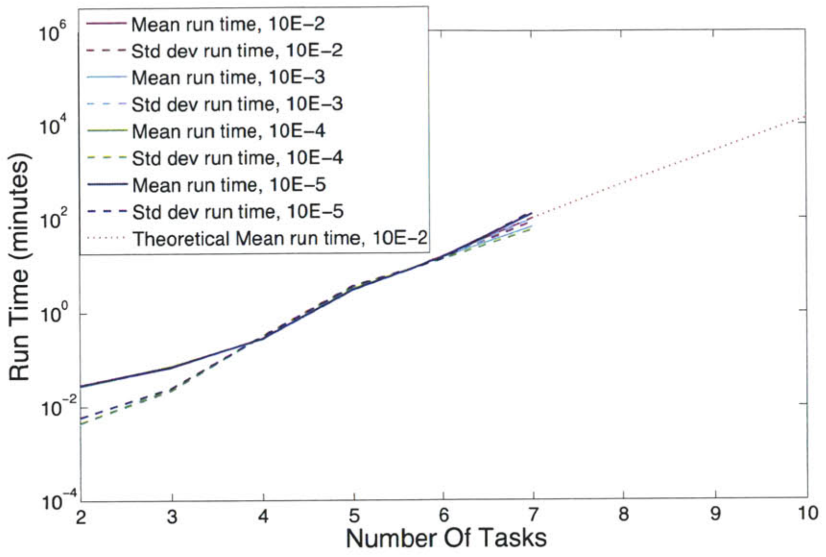


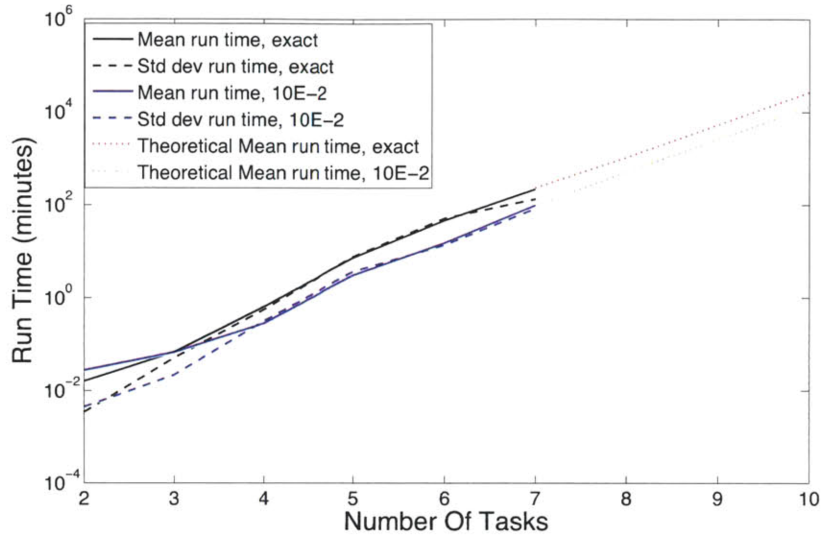
Figure 3-32: Run Time Results – MIP Formulation 1 ϵ-Rounding. The ϵ-Rounding Heuristic also experiences an exponential increase in run time for a linear increase in number of tasks, where the values for $n \geq 8$ are given as theoretical values.

is 2.98 times faster.

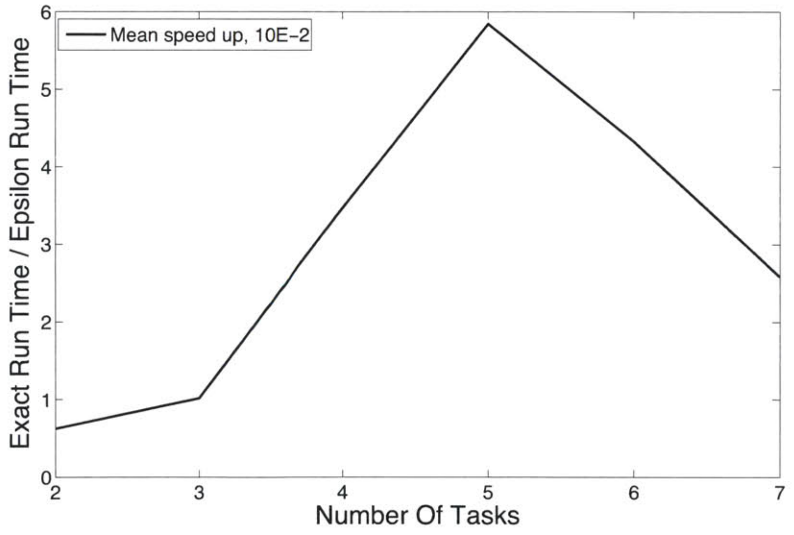
Characterization of Optimal Solution

Figure 3-34 illustrates the performance of the ϵ-Rounding Heuristic for a varying number of tasks compared to the exact algorithm. For example, a value of 0.98 in Figure 3-34 means the ϵ-Rounding Heuristic produces an *a priori* task route that achieves an objective function value on average 2% less than the exact algorithm. Although the figure only shows the results for $\epsilon = 1{\times}10^{-2}$, the algorithm performs similarly for ϵ ranging from $1{\times}10^{-2}$ to $1{\times}10^{-5}$.

After conducting over 2000 simulations with both the exact algorithm and ϵ-Rounding Heuristic with a varying number of tasks, the heuristic produces an optimal solution on average 98.11% of the objective function value with the exact formulation. For a significant speed up in run time, the level of degradation in the optimal solution may be acceptable to the mission planner.



(a) Run Time Results – MIP Formulation 1 Exact and ϵ-Rounding



(b) % Speed Up of ϵ-Rounding vs. Exact

Figure 3-33: MIP Formulation 1 Run Time Results with ϵ-Rounding

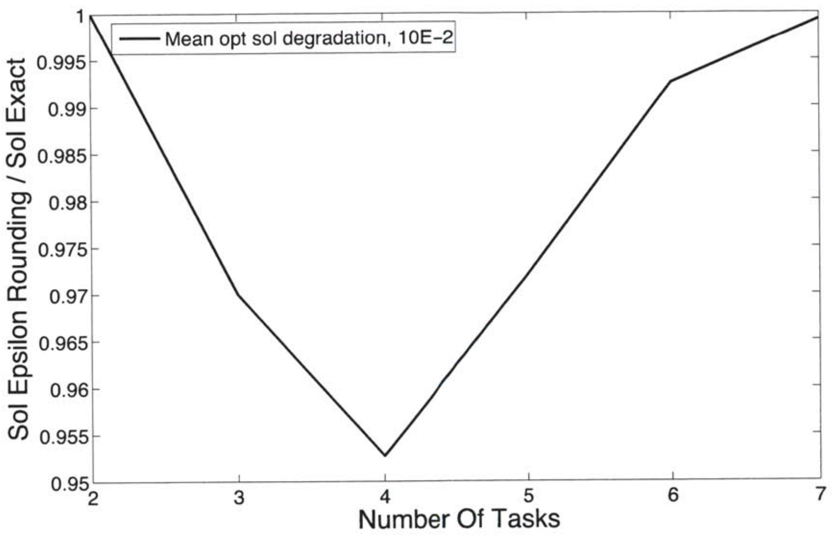


Figure 3-34: Optimal Solution Degradation – MIP Formulation 1 ϵ-Rounding. A value of 0.98 represents that the ϵ-Rounding Heuristic produces an *a priori* task route that achieves an objective function value on average 2% less than the exact algorithm. The figure displays results for $\epsilon = 1\times10^{-2}$; the algorithm experienced similar results for $\epsilon \in \{1\times10^{-2}, 1\times10^{-3}, 1\times10^{-4}, 1\times10^{-5}\}$.

3.6.2 MIP Formulation 2

MIP Formulation 2 produces task route solutions that protect against worst case disturbances; in particular, the mission planner can skip the next task or end the mission early and maneuver directly to the end mission location.

Although the Route Alteration Graph G_a adds a small number of arcs to the Time and Navigation Expanded Stochastic Graph G_s, there is a significant increase in possible solutions. In addition to determining the best task route among all possible task route combinations (Equation 3.2),

$$\sum_{k=0}^{n} \frac{n!}{k!} * 2^{k+1}$$

the formulation must also identify time and accumulated navigation error thresholds at each task that dictate when the mission planner should alter its current route. The added complexity causes the algorithm to run much slower than MIP Formulation 1, even for scenarios with a small number of tasks and time steps. The run time analysis for MIP Formulation 2 is consistent with Cates [11].

Due to the large increase in computation speed, MIP Formulation 2 is not a viable method for practical use in the UUVMPP. Chapter 4 presents an algorithm that allows for complete route alteration on-line and scales better with increasingly complex mission scenarios.

3.7 Summary

To develop an *a priori* route through the environment, the mission planner is given a finite amount of time to find the best task sequence to maximize or minimize some mission objective. MIP Formulation 1 uses network optimization techniques and a Mixed-Integer Program to determine the best route that maintains β percent chance of reaching the end mission location without exceeding operational constraints. MIP Formulation 2 enhances the result from MIP Formulation 1 by permitting the mission planner to skip tasks or end the mission early if the UUV does not have the resources available to continue its intended route. The exact algorithm and ϵ-Rounding Heuristic are practicable methods to solve MIP Formulation 1; however, MIP Formulation 2 proves to be computationally expensive with network optimization methods as the number of tasks grow.

Chapter 4 presents Dynamic Programming (DP) methods to solve the UUVMPP in both an *a priori* and on-line environment. Although exact DP algorithms quickly become intractable, there are a variety of approximate DP algorithms that perform well while maintaining a relatively low run time.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

A Priori and On-line Routing with Dynamic Programming

4.1 Introduction

The Unmanned Underwater Vehicle Mission Planning Problem (UUVMPP) attempts to maximize the overall value of a mission by finding the best possible task sequence under uncertain conditions. Dynamic Programming (DP) is a mathematical technique that can be applied to a broad range of practical problems; in particular, DP is useful in modeling situations where decision are made in stages, where the outcome of each decision may not be completely known but can be predicted by the next stage. At each stage, this technique captures the careful balance between the low (or high) present cost with the undesirability (or desirability) of future costs. In the UUVMPP, the mission planner (referred to as **the controller** in DP) faces a route decision at the conclusion of each executed task. Due to the sequential decision process the controller faces during a mission, Dynamic Programming is a natural fit for the UUV's mission-planning algorithm.

The first algorithm presented in this chapter, the Brute Force (BF) method, is an exact Dynamic Program which computes *a priori* the optimal routing policy for the controller over the duration of the mission and the globally optimal objective function value. The

BF method enumerates every possible situation presented to the controller, no matter how unlikely, and utilizes the updated state information to determine the optimal next decision recursively.

The next algorithm is an approximate Dynamic Program called the Rollout Algorithm (RA), which uses a weighted Nearest Neighbor (WNN) heuristic to estimate the remaining cost-to-go. The RA algorithm allows the controller to handle larger-scale problems at an acceptable degradation in the optimal solution. Results are then presented comparing the DP algorithms to MIP Formulation 1 in both an *a priori* and on-line setting.

4.2 Brute Force (Exact) Method

The Mixed-Integer Programming method is a feasible way to solve the UUVMPP since the network flow model consists of a directed graph; the algorithm begins at the super source node (*i.e.* start mission location) and maneuvers through the network flow until reaching the super sink node (*i.e.* end mission location). Given a directed graph with nodes and some weight assigned to each arc, formulating the UUVMMP as a shortest path problem is an alternate way to solve a network flow model of this nature. A deterministic shortest path problem finds at each node the path formed by a sequence of successor nodes that terminates at the end node and has minimum length [6]. Due to the stochasticity of the travel and task execution time, the UUVMPP can be modeled as a Stochastic Shortest Path (SSP) problem. The SSP problem is a generalization of a shortest path problem whereby at each node the arc lengths to all possible successor nodes are random, determined by a given set of probability distributions. The goal of the formulation is to find a path that leads to the end node, but in this case the path length is an expected value due to the randomness incorporated into the model.

Similar to the Traveling Salesman Problem example in [4], the UUVMPP can be formulated as an SSP problem. Figure 4-1 illustrates a two-task example problem where each task is assigned a label from the set $\mathbf{L} = \{1, 2, \ldots, n\}$, with n representing the total number of tasks. The start node and end node are given the labels S and E, respectively. Each task is

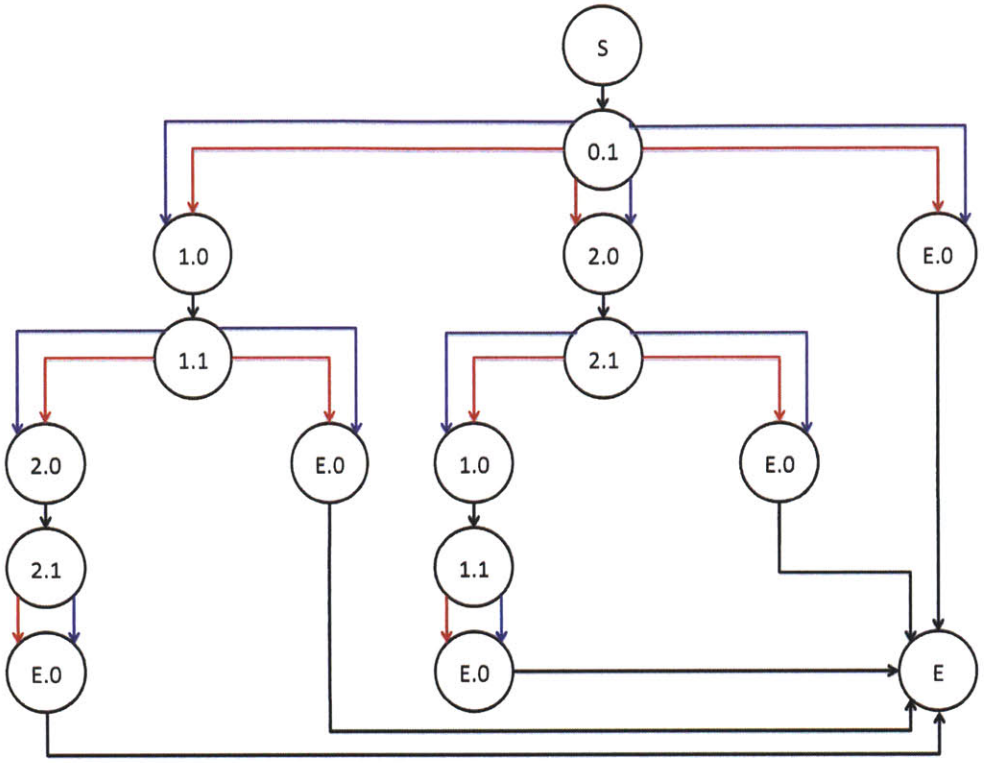


Figure 4-1: Two-Task Example, Stochastic Shortest Path Representation. S represents the start mission location and E represents the end mission location. Nodes of type $i.0$ represent being at a task prior to execution while nodes of type $i.1$ represent being at a task after execution. A red arc indicates travel from one task to another task without a navigation fix, a blue arc indicates travel from one task to another task with a navigation fix, and a black arc indicates task execution.

split into two distinct nodes, where one node represents being at the task prior to execution (denoted by **.0** after the task label) and the second node represents being at the task after execution (denoted by **.1** after the task label). A red arc indicates travel from one task to another task direct **without** a navigation fix, a blue arc indicates travel from one task to another task **with** a navigation fix, and a black arc indicates task execution.

Figure 4-1 is an accurate depiction of a two-task problem in an environment where all travel times and task execution times are constant and deterministic. These problem characteristics must be appropriately modeled; therefore, the formulation repeats each node for every situation the controller could potentially encounter. The model incorporates a constant discretization from mission time 0 to T_{MAX} and 0 to N_{MAX} by the time step parameter, T, to

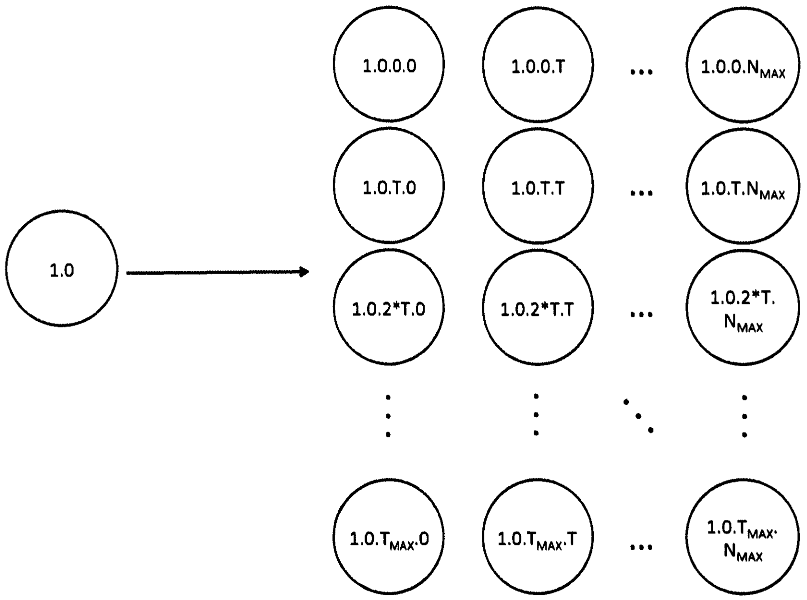


Figure 4-2: Expansion of Node 1.0 in Stochastic Shortest Path Framework. Node 1.0.t.v represents being at task 1 prior to task execution t units of time after the mission start time with v units of accumulated navigation error. The model incorporates a constant discretization from mission time 0 to T_{MAX} and 0 to N_{MAX} by the time step parameter, T, to represent every possible time and navigation error combination.

represent every possible time and navigation error combination. One can view Figure 4-1 as the top layer of the problem with $\frac{T_{MAX}}{T} * \frac{N_{MAX}}{T}$ copies of each node underneath the top layer, where each arc represents a set of arcs that follow some specified probability distribution. Figure 4-2 illustrates how the formulation expands the node 1.0 in this fashion.

Similar to Chapter 3, the DP formulations assume that if the controller decides to maneuver to a task, then as long as the UUV has ample time and positional certainty it will complete the task with probability one. In the two-task example depicted in Figure 4-1, beginning at the start node, the controller must choose one of six options:

1. Go to task 1 without a navigation fix.
2. Go to task 1 with a navigation fix.
3. Go to task 2 without a navigation fix.
4. Go to task 2 with a navigation fix.

5. Go to the end node without a navigation fix.

6. Go to the end node with a navigation fix.

The controller traverses the set of arcs indicated by its decision, completes the chosen task, and faces another set of decisions. The process continues until either there are no remaining tasks or the controller predicts it will not have enough time to complete any more tasks, and will then maneuver to the end mission location.

4.2.1 State Space

The state space summarizes all past information relevant to the controller for future optimization. In the classical Traveling Salesman Problem, the state space consists of which tasks have already been completed to ensure the mission plan does not complete a task more than once. Since the problem splits each task into two separate nodes, at a task prior to execution and at a task after execution, the state space is modeled accordingly with two distinct sets of states. Consider set X_k, which contains the states x_k where a decision must be made and its outgoing arcs represent transit to a task. The second set, Y_k, contains the states y_k where no decision is necessary and its outgoing arcs represent task execution. Set S_k is the intersection of sets X_k and Y_k, *i.e.* $S_k = X_k \bigcap Y_k$, and contains all possible states in the UUVMPP; states s_k encompass the set S_k. The k subscript represents the stage of the system, in this case how many task route decisions have been made thus far.

Figure 4-3 summarizes the state space progression in the UUVMPP. The system always begins at the start node, an element of X_k where $k = 0$, where the controller makes a decision and transitions to the first task (or the end mission location) according to some specified distribution. The controller arrives at the first task prior to execution, an element of Y_k at stage $k = 1$, and executes the task. Once the controller is at the first task after execution, an element of X_k at stage $k = 1$, it faces a new set of decisions. The mission planning algorithm terminates either when all tasks are complete or the controller predicts the UUV will exceed operational constraints, in which case the controller transitions to the end mission location,

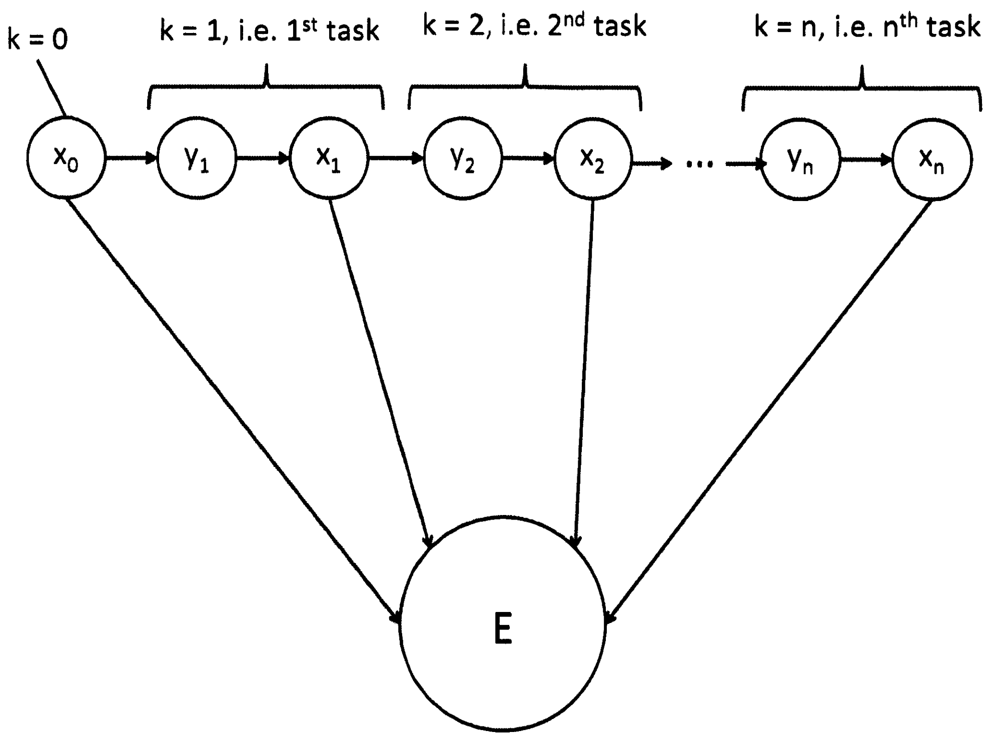


Figure 4-3: UUVMPP Stage Progression. The mission begins at the start node (*i.e.* state x_0) where the controller makes a decision and transitions to the first task according to some specified distribution. The controller arrives at the first task prior to execution (*i.e.* state y_1) and executes the task. After completing the task, the controller faces a new set of decisions at state x_1. The mission planning algorithm terminates either when all tasks are complete or the controller predicts the UUV will exceed operational constraints, in which case the controller transitions to the end mission location.

an element of Y_k denoted E. The algorithm progresses at most n stages since the UUV cannot complete a task more than once.

The states $x_k \in X_k$ represent being at a task after execution, denoted by a node with **.1** after the task label. When the controller is at a state in set X_k, it must make a decision on where to next maneuver. Each state in set X_k is defined by an $n+2$ dimensional row vector as follows: the first n elements represent the tasks the UUV has completed thus far, denoted by vector $\mathbf{a}$; element $n+1$ represents the elapsed time since the start of the mission; and element $n+2$ represents the accumulated navigation error. The location of the UUV is given by the last element in $\mathbf{a}$.

Each element of X_k is structured as $\langle \overbrace{\mathbf{a},}^{1,2,\ldots,n} \underbrace{T_r}_{\text{n+1}}, \overbrace{N_t}^{\text{n+2}} \rangle$, where:

- $\mathbf{a} = \langle a_1, a_2, \ldots, a_l.1 \rangle$, the set of l tasks that have been completed thus far.
- The UUV is located at task a_l after task execution.
- The UUV is T_r units of time into its mission.
- The UUV has accumulated N_t units of navigation error.

Consider the two-task example where $x_k = \langle 1.1,\ 0,\ 0,\ 5,\ 1 \rangle$, *i.e.* $\mathbf{a} = \langle 1.1,\ 0,\ 0 \rangle$, $T_r = 5$ units, and $N_t = 1$ unit. This state represents that the UUV has just completed task 1 five units after the mission started and has accumulated onc unit of navigation error; therefore, the controller must now choose whether to maneuver to task 2, possibly with a navigation fix, or transition to the end mission location.

To compile the states $x_k \in X_k$, every task sequence represented by row vector $\mathbf{a}$ is repeated for all possible time and accumulated navigation error combinations to represent the situations the controller could potentially encounter in its mission. Using the above example where the UUV has just completed task 1, $\mathbf{a} = \langle 1.1, 0, 0 \rangle$ is duplicated for every time step from 0 to T_{MAX} and for every allowable accumulated navigation error 0 to N_{MAX}. Figure 4-2 illustrates the discretization concept.

The states $y_k \in Y_k$ represent being at a task prior to execution, denoted by a node with **.0** after the task label. Similar to the X_k states, the Y_k states are an $n+2$ dimensional row vector where the first n elements, vector $\mathbf{a}$, represent the tasks the UUV has completed thus far or will complete this stage, element $n+1$ represents the elapsed time since the start of the mission, and element $n+2$ represents the accumulated navigation error. The location of the UUV is given by the last element in $\mathbf{a}$.

Each element of Y_k is structured as $\langle \overbrace{\mathbf{a},}^{1,2,\ldots,n} \underbrace{T_r}_{\text{n+1}}, \overbrace{N_t}^{\text{n+2}} \rangle$, where:

- $\mathbf{a} = \langle a_1, a_2, \ldots, a_l.0 \rangle$, the set of $l-1$ tasks that have been completed thus far and the l^{th} task a_l that will be completed next.

- The UUV is located at task a_l before task execution.
- The UUV is T_r units of time into its mission.
- The UUV has accumulated N_t navigation error.

If the algorithm is in state $Y_k = \langle 1.0, 0, 0, 5, 0\rangle$ in the two-task example, the UUV would have just arrived at task 1 five time units after the mission start time with no accumulated navigation error. In this case, the controller's next action is to execute task 1 with probability one. As before, every possible task sequence represented by **a** is repeated for all time and accumulated navigation error pairs to compile the states $y_k \in Y_k$.

4.2.2 Control Space

The control space consists of all decisions available to the controller at a given time. The control u_k is constrained to take values in a given set $U(s_k)$ which depends on the current state s_k, *i.e.* $u_k \in U_k(s_k)\ \forall s_k \in S_k$ and k. In the UUVMPP, the control space depends on whether the UUV is at a task before or after execution and which tasks have already been completed.

For each task $l \in \mathbf{L}$, let $\dot{l}$ represent the controller's decision to travel directly to task l **without** a navigation fix and $\bar{l}$ represent the controller's decision to travel to task l **with** a navigation fix. The control space for all possible states are defined below, where $L_C^{x_k}$ represents the set of tasks that have yet to be completed at some state x_k.

$$U_0(x_0) = \{\dot{1}, \bar{1}, \dot{2}, \bar{2}, \ldots, \dot{n}, \bar{n}, \dot{E}, \bar{E}\} \qquad \forall\, x_0 \in X_0 \qquad (4.1)$$

$$U_k(x_k) = \{L_C^{\dot{x}_k}, L_C^{\bar{x}_k}, \dot{E}, \bar{E}\} \qquad \forall\, k, \forall\, x_k \in X_k \qquad (4.2)$$

$$U_k(y_k) = \emptyset \qquad \forall\, k, \forall\, y_k \in Y_k \qquad (4.3)$$

Consider the two-task sample problem. Beginning at the start node (*i.e.* state x_0), the controller has the option to take one of six choices represented by $U_0(x_0) = \{\dot{1}, \bar{1}, \dot{2}, \bar{2}, \dot{E}, \bar{E}\}$. Assume the controller decides to travel to task 1 with a navigation fix (*i.e.* applies control

$\bar{1}$). The system then transitions to the state y_1 and the controller must now complete task 1, where there is no control to select (*i.e.* $U_1(y_1) = \emptyset$). After completion of task 1, the system is now in state x_1 and the control space decreases to four possible controls, denoted by $U_1(x_1) = \{\dot{2}, \bar{2}, \dot{E}, \bar{E}\}$.

4.2.3 Transition Probabilities

To model the transition between two states, a DP algorithm requires a description of the corresponding transition probabilities.

Since a Dynamic Program is a discrete-time system, the random variables for all state transitions is some probability mass function, either obtained directly (*e.g.* known *a priori*) or approximated with some discretized probability distribution function. All of the transitions in the UUVMPP are from an X_k state to a Y_k state (travel between tasks) or from a Y_k state to an X_k state (task execution). The notation implies that a transition cannot occur between two X_k states or two Y_k states because of the original assumption — the UUV must complete a task with probability one if the controller chooses to transit to the task.

The randomness in a Dynamic Program, whether predictable or unpredictable, is the disturbance in the system. The transition probability distributions for all states are dependent on the location of the UUV, the type of mission, and the elapsed time of the mission. All of these problem parameters are assumed to be known before the start of the mission.

The following two equations formalize the above intuition:

$$p_{x_k,y_k}(u_k) \sim \omega(x_k, u_k) \qquad \forall\, k, x_k \in X_k, u_k \in U_k(x_k), y_k \in Y_k^{x_k,u_k} \tag{4.4}$$

$$p_{y_k,x_k} \sim \omega(y_k) \qquad \forall\, k, y_k \in Y_k, x_k \in X_k^{y_k} \tag{4.5}$$

where:

- $p_{x_k,y_k}(u_k)$ is the probability mass function for the state transition x_k to y_k when applying control u_k, *i.e.* travel between tasks.

- p_{y_k,x_k} is the probability mass function for the state transition y_k to x_k, *i.e.* task execution.

- $\omega(x_k, u_k)$ is some probability distribution dependent on the state x_k and control u_k control.

- $\omega(y_k)$ is some probability distribution dependent on the state y_k.

- $Y_k^{x_k,u_k}$ is the set of all feasible Y_k states the controller could transition to given the controller is at state x_k and control u_k is applied.

- $X_k^{y_k}$ is the set of all feasible X_k states the controller could transition to given the controller is currently at state y_k.

4.2.4 Stage Cost and Arc Weights

Another critical component of a Dynamic Program is a cost function that is additive over time where each stage contributes some non-negative utility to the overall objective, possibly at the cost of future benefit. The objective of the UUVMPP is to maximize the utility of the mission while not exceeding mission constraints. To model additive cost appropriately in the Stochastic Shortest Path framework, all stage costs (*i.e.* arc weights) representing travel between two tasks have a weight of 0 and all arcs representing task execution have some non-negative weight (since the controller does not achieve a mission objective until successful execution of a given task).

The above results are summarized as follows:

$$g(x_k, u_k, y_k) = 0 \qquad \forall\, k, x_k \in X_k, u_k \in U_k(x_k), y_k \in Y_k \tag{4.6}$$

$$g(y_k, x_k) = r_i * f(y_k) \qquad \forall\, k, y_k \in Y_k, x_k \in X_k, i \in \{1, 2, \dots, n\} \tag{4.7}$$

where:

- $g(x_k, u_k, y_k)$ is the additive cost accumulated when applying control u_k at state x_k, resulting in a transition to state y_k.

- $g(y_k, x_k)$ is the additive cost accumulated when transitioning from state y_k to state x_k.

- r_i is the non-negative maximum reward attained by completing task i.

- $f(y_k)$ is some scalar dependent on the current state y_k.

4.2.5 Cost Function

The cost function summarizes the state space, control space, transition probabilities, and stage costs in a recursive algorithm that relies on the *Principle of Optimality* to find the globally optimal solution (defined in Section 2.3). The cost function is additive, *i.e.* the cost incurred at time k, defined as $g_k(x_k, u_k, w_k)$, accumulates over time.

The BF method solves the UUVMPP as follows: the algorithm first sets the remaining cost-to-go to 0 for all states where the optimal control is to maneuver to the end node, and $-\infty$ for all states where there is at least a probability $1-\beta$ the end node will not be reached within T_{MAX} or N_{MAX}[1]. The algorithm then steps backwards one stage and, for all possible states, ranks all feasible controls and chooses the one with the highest expected remaining reward, using the fact that all future states either have a terminal reward of 0 or $-\infty$. The algorithm continues until it reaches stage 0 and weighs all possible controls in the set $U_0(x_0)$. The solution to this final problem produces the globally optimal objective value and optimal initial control.

The cost functions and terminal conditions are stated below. There must be two separate cost functions in this algorithm, one for all X_k states and one for all Y_k states. $J_k(x_k)$ is interpreted as the remaining "cost-to-go" given the controller is at state x_k, and $\mathbf{F}$ represents the event that the UUV fails to reach the end node within T_{MAX} or N_{MAX} causing a mission

[1]This is analogous to the β confidence level in the Mixed-Integer Programming formulations, which ensures routes are only considered if there is at least a $\beta\%$ chance of reaching the end node within operational constraints.

failure.

$$J_k(x_k) = \max_{u_k \in U_k(x_k)} \left\{ \sum_{j \in Y_k^{x_k, u_k}} p_{x_k,j}(u_k) J_{k+1}(j) \right\} \quad \forall\, k, x_k \in X_k \tag{4.8}$$

$$J_k(y_k) = r_i * f(y_k) + \sum_{j \in X_k^{y_k}} p_{y_k,j} J_{k+1}(j) \quad \forall\, k, y_k \in Y_k \tag{4.9}$$

$$J_k(x_k) = 0 \quad \forall\, k, x_k \in \left\{ X_k | U_k(x_k) = \{\dot{E}, \bar{E}\} \right\} \tag{4.10}$$

$$J_k(x_k) = -\infty \quad \forall\, k, x_k \in \left\{ X_k | Pr\{\mathbf{F}\} > 1 - \beta \right\} \tag{4.11}$$

By computing the optimal costs-to-go for every state using the above system equations, the controller obtains two key pieces of information: (1) the optimal cost-to-go from the start node, $J_0^*(x_0)$, and (2) the optimal routing policy, π^*. The solution enumerates the optimal decisions given any potential situation the controller could encounter; *i.e.* the solution not only provides an *a priori* optimal route but it gives the optimal policy if the controller arrives at a state not anticipated beforehand, an idea elaborated upon in Section 4.4.1 Online Optimization and Re-Planning.

4.2.6 Extendability

The BF method can be appropriately modified to account for other resource constraints such as energy or risk. This section provides a technique to incorporate energy into the UUVMPP which can be extended to any given mission constraints.

The controller must first amend the state space to track the extra piece of information, for example energy consumption. As a result, each element in X_k has the structure $\langle \overbrace{\mathbf{a},}^{1,2,\ldots,n} \underbrace{T_r}_{\text{n+1}}, \overbrace{N_t}^{\text{n+2}}, \underbrace{E_e}_{\text{n+3}} \rangle$, where:

- $\mathbf{a} = \langle a_1, a_2, \ldots, a_l.1 \rangle$, the set of l tasks that have been completed thus far.

- The UUV is located at task a_l after task execution.

- The UUV is T_r units of time into its mission.
- The UUV has accumulated N_t navigation error.
- The UUV has used E_e units of energy.

The amount of discretization in the energy dimension depends on the level of accuracy the controller desires. If energy consumption is not critical to determine the UUV's task route, then a coarse resolution is sufficient; *e.g.* $E_e \in \{E_{LOW}, E_{MED}, E_{HIGH}\}$. If energy consumption is of great importance to the controller, then a fine resolution is more appropriate; *e.g.* $E_e \in \{0, T, 2{*}T, \ldots, E_{MAX}\}$. Figure 4-4 illustrates how to expand the state $\langle 1.1, 0, 0, T_r, N_t \rangle$, an element of X_k in a two-task problem, to account for energy with a fine resolution. Once the state space is modified, the transition probabilities in the model depend on which tasks have been completed, time, accumulated navigation error, and energy consumption.

4.2.7 Shortcomings

The state space grows rapidly, even for a scenario with a small number of tasks. The state space grows according to $O(n! * \frac{T_{MAX}}{T} * \frac{N_{MAX}}{T})$, where T represents the discrete time step. Table 4.1 demonstrates the growth of the state space for a various number of tasks with T_{MAX}, N_{MAX}, and T held constant. The parameters in Table 4.1 are fixed at T_{MAX} = 72,000 seconds, N_{MAX} = 36,000 seconds, and T = 3,600 seconds. Figure 4-5 presents the information on a logarithmic y-axis to demonstrate the relative growth of the state space as a function of the number of tasks. Due to computational limitations, the number of states beyond eight tasks is given as an approximation.

The practical use of the exact algorithm for a large number of tasks is not tractable because of the exponential growth in the state space. The UUVMPP necessitates the use of a suboptimal control technique which enables the controller to plan and re-plan for problems of higher complexity.

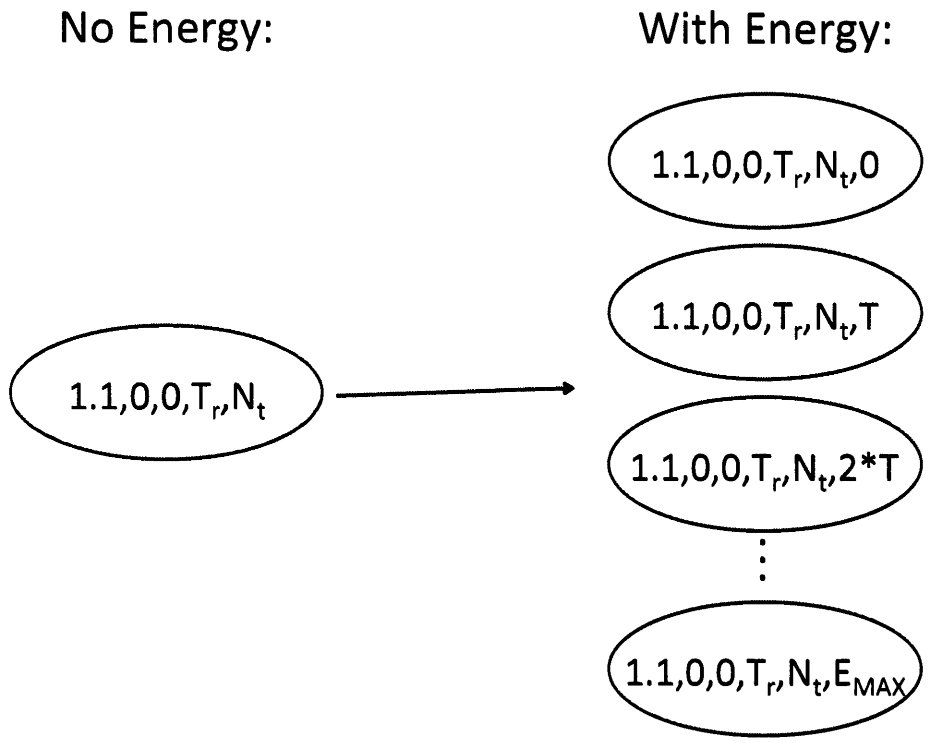


Figure 4-4: State Modification of $\langle 1.1, 0, 0, T_r, N_t \rangle$ for Energy. The figure shows how to expand the state $\langle 1.1, 0, 0, T_r, N_t \rangle$, an element of X_k in a two-task problem, to account for energy with a fine resolution (*i.e.* $E_e \in \{0, T, 2 * T, \dots, E_{MAX}\}$).

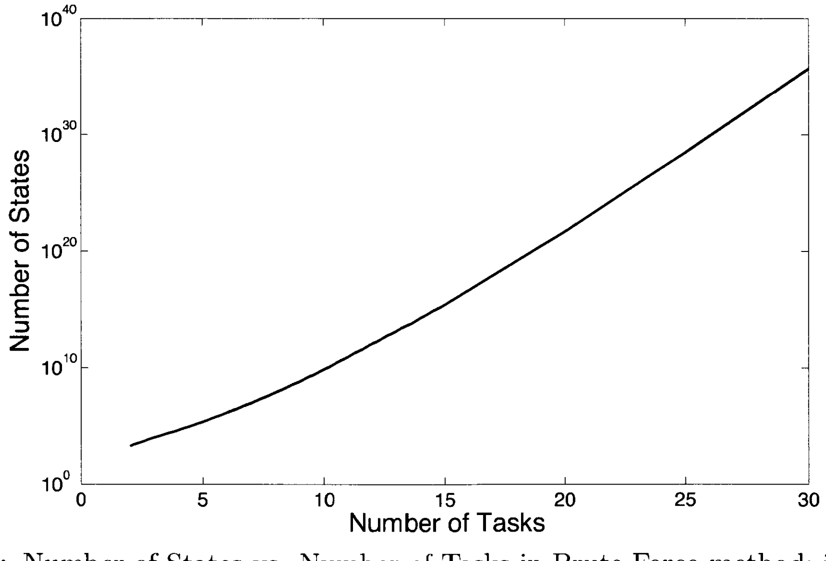


Figure 4-5: Number of States vs. Number of Tasks in Brute Force method; illustration of Table 4.1.

Number Of Tasks	Number Of States
2	3.005×10^3
3	1.063×10^4
4	4.456×10^4
5	2.255×10^5
6	1.356×10^6
7	9.494×10^6
8	7.595×10^7
9	$\sim6.836\times10^8$
10	$\sim6.836\times10^9$
15	$\sim2.463\times10^{15}$
20	$\sim4.583\times10^{21}$
25	$\sim2.922\times10^{28}$
30	$\sim4.997\times10^{35}$

Table 4.1: Number of States vs. Number of Tasks in Brute Force method; see Figure 4-5.

4.3 Approximate DP Algorithms – A Priori Route Optimization

To improve the run-time capabilities for the UUVMPP, this section presents two suboptimal control techniques, the Weighted Nearest Neighbor (WNN) and Rollout Algorithm (RA), and how the controller can use both methods to compute an *a priori* task route solution.

4.3.1 Weighted Nearest Neighbor

The classical greedy heuristic Weighted Nearest Neighbor finds a feasible solution to the Traveling Salesman Problem (TSP) by choosing tasks that create the most immediate gain to the optimal solution without considering future ramifications. The heuristic begins at stage 0 and progresses forward in time; the BF method, on the other hand, begins at stage $n+1$ and moves backwards in time until stage 0. Dissimilar to a TSP, there is an added time constraint that may prohibit the controller from completing all available tasks. Therefore, the WNN algorithm must be amended slightly to solve the UUVMPP, a Prize Collecting Traveling Salesman Problem [12]. Let:

- P be the path formed by the WNN, where $P = \{S = i_0, i_1, \ldots, i_M, E\}$.
- $f(i, j)$ be the distance measure between tasks i and j.

```
Begin:
    $U = \{S\}$.
    loop
        Find all feasible controls from the last element in $U$, call it $L$.
        Calculate $f(L, j)$ for all feasible controls $j$.
        Find $U_{MIN}$, the control that produces the minimum $f(L, j)$.
        $U = U \cap U_{MIN}$.
    loop until There are no feasible controls.
    $P = U \cap E$.
End
```

The controller's choice of $f(i, j)$ is paramount to the effectiveness of the algorithm. The most elementary $f(i, j)$ is:

$$f(i, j) = E[t_{i.1}^{j.0}] + E[t_{j.0}^{j.1}] \tag{4.12}$$

where $t_{i.1}^{j.0}$ represents the random variable for the travel time from task i to task j and $t_{j.0}^{j.1}$ represents the random variable for the execution of task j. Therefore, Equation 4.12 calculates the expected amount of time from task i after execution until task j after execution. The distance measure in Equation 4.12 finds the task that is expected to take the least amount of time, which in turn is least penalizing to the time constraint. If another resource parameter is involved, *e.g.* energy, $f(i,j)$ could be presented as:

$$f(i,j) = \alpha * \left(E[t_{i.1}^{j.0}] + E[t_{j.0}^{j.1}]\right) + (1-\alpha) * \left(E[e_{i.1}^{j.0}] + E[e_{j.0}^{j.1}]\right) \tag{4.13}$$

where $0 \leq \alpha \leq 1$. The term $\{E[e_{i.1}^{j.0}] + E[e_{j.0}^{j.1}]\}$ measures the expected energy consumption from task i until after the execution of task j, similar to the random variable t. The α serves as a weighting factor that the controller chooses based on which constraint is more critical to the UUV's task route.

The distance measures in Equations 4.12 and 4.13 do not factor in the relative reward of each task in the UUVMPP, *i.e.* the stage cost accumulated for task execution (presented in Equation 4.7). To account for the task benefit, the above distance measures are amended as follows:

$$f(i,j) = \frac{E[t_{i.1}^{j.0}] + E[t_{j.0}^{j.1}]}{r_j * f(y_k)} \tag{4.14}$$

$$f(i,j) = \frac{\alpha * \left(E[t_{i.1}^{j.0}] + E[t_{j.0}^{j.1}]\right) + (1-\alpha) * \left(E[e_{i.1}^{j.0}] + E[e_{j.0}^{j.1}]\right)}{r_j * f(y_k)} \tag{4.15}$$

Equation 4.14 is similar to the greedy heuristic in the Integer Knapsack Problem (Section 2.1.4), where at each stage the controller chooses the task that in the near term maximizes its utility per unit space (in this case, time) at a possible sacrifice to future utility.

Unlike the Integer Knapsack Problem, the utility per unit time in the UUVMPP varies due to the stochasticity in the travel times and task execution times. As a result, the exact calculation of the Weighted Nearest Neighbor solution is too cumbersome. Relying on expected values does not produce a robust solution that performs well in a variety of situations, and assuming the controller encounters worst case disturbances throughout is an overly-conservative approach. To mitigate these concerns, this section considers an approximation technique known as Monte Carlo to obtain sufficient estimates of the WNN solution.

The Monte Carlo simulation method utilizes a random number generator to send sample trajectories through the Stochastic Shortest Path model. Each trajectory represents one possible route the controller would take through the mission environment given a particular set of disturbances, where each subsequent task is chosen using the Weighted Nearest Neighbor algorithm. This approximation method allows the algorithm to rapidly generate WNN solutions.

By sending a large number of trajectories from the start mission time, the controller gets numerous samples of Weighted Nearest Neighbor algorithm objective function values. The controller has a number of options to develop a suitable task route with this range of solutions, *e.g.* taking the mean objective function value of all trajectories. If the controller desires a higher level of confidence in generating a successful UUV task route, the algorithm could instead produce the value where some percent, say 90%, of trajectories exceed that baseline. With either calculation, the controller would have a Weighted Nearest Neighbor solution that includes the objective function value and an *a priori* route through the mission environment.

The next section presents the Rollout Algorithm which uses Dynamic Programming and the WNN solution in tandem. The Rollout Algorithm takes longer to compute than WNN alone, but is guaranteed to outperform the heuristic.

4.3.2 Rollout Algorithm

The Rollout Algorithm (RA) is an Approximate Dynamic Programming technique that uses a limited look-ahead scheme to estimate the cost-to-go function. As discussed in [4], an efficient way to reduce computation in DP is to truncate the time horizon and utilize at each stage a decision based on lookahead of some number of stages. This section considers a *one-step lookahead policy*, defined below.

Definition. *Given some N-stage Dynamic Program at stage k and at state x_k, a one-step lookahead policy applies the control $\bar{\mu}_k(x_k)$ which attains the minimum in the expression*

$$\min_{u_k \in U_k(x_k)} E\Big[g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k))\Big]$$

where $\tilde{J}_{k+1}$ is some approximation of the exact cost-to-go function J_{k+1}, with $\tilde{J}_N = g_N$.

A one-step lookahead policy uses as the terminal cost-to-go function an approximation to the optimal cost that starts at the end of the lookahead. Only a single maximization problem needs to be solved in the case of a one-step lookahead, a considerable reduction in complexity compared to the Brute Force method.

The RA begins at stage 0 and progresses forward in time, similar to the Weighted Nearest Neighbor heuristic. In the Rollout Algorithm, the approximating function $\tilde{J}_{k+1}$ is the cost-to-go of some known heuristic policy, otherwise known as the base policy. Although the base policy can be any feasible heuristic, this thesis considers the WNN.

The system equations in the Rollout Algorithm are identical to the cost functions in the Brute Force method (Equations 4.8 and 4.9) except the cost-to-go functions J_{k+1} are replaced by approximate cost-to-go functions $\tilde{J}_{k+1}$. The resulting equations are:

$$\tilde{J}_k(x_k) = \max_{u_k \in U_k(x_k)} \left\{ \sum_{j \in Y_k^{x_k,u_k}} p_{x_k,j}(u_k)\tilde{J}_{k+1}(j) \right\} \qquad \forall\, k, x_k \in X_k \tag{4.16}$$

$$\tilde{J}_k(y_k) = r_j * f(y_k) + \sum_{j \in X_k^{y_k}} p_{y_k,j}\tilde{J}_{k+1}(j) \qquad \forall\, k, y_k \in Y_k \tag{4.17}$$

Let P be the path formed by the Rollout Algorithm. A pseudocode description of the algorithm is as follows:

```
Begin:

    Find all feasible controls from the start mission location.

    Calculate $\tilde{J}_k(x_k)$ and $\tilde{J}_k(y_k)$ for all feasible controls using the base heuristic.

    Find the control, $P_{MAX}$, that produces the maximum expected one-stage DP cost plus
    estimated remaining cost-to-go formed by route $U$.

    $P = P_{MAX} \cap U$.

End
```

The RA solves a one-stage Dynamic Programming problem where the terminal cost-to-go functions, $\tilde{J}_k(x_k)$ and $\tilde{J}_k(y_k)$, are approximated by the base policy. Once the algorithm solves the one-stage maximization problem, the controller applies the determined control (*i.e.* P_{MAX}) and follows the route returned by the base policy (*i.e.* U).

Rollout Algorithms are beneficial due to the *sequentially improving property*, which guarantees an improved performance over the corresponding base policy[2]. The pseudocode above enhances the solution returned by the base policy since it "looks ahead" one stage, reducing the length of the horizon that the base policy estimates.

The Rollout Algorithm provides an effective way to reduce computation time over the Brute Force method while producing high quality solutions. For other examples of the Rollout Algorithm and its application to the Vehicle Routing Problem, see Bertsekas [5] and Secomandi [13].

4.4 Approximate DP Algorithms – On-line Route Optimization

4.4.1 On-line Re-planning

As the UUV maneuvers through the mission environment, there is some chance the problem data changes. For example, some task may become unavailable due to some unforeseen circumstance or the travel times are severely altered due to inclement weather. In these situations, the controller needs to develop a modified mission plan with the new data. When a problem is resolved on-line with different problem parameters, this is known as on-line re-planning.

During the mission sortie, the controller is not afforded the opportunity to meticulously develop a new route since the vehicle would be wasting valuable time and energy capacity; finding a "good" result quickly often trumps determining the new optimal route, so the controller relies on suboptimal control techniques to re-plan in an on-line environment.

[2]Bertsekas, Tsitsiklis, and Wu [7] provide a formal proof of the sequentially improving property.

Weighted Nearest Neighbor

The main advantage of the WNN heuristic is that it requires very little computation overhead to evaluate. This is particularly beneficial in an on-line environment since the controller must quickly decide where to alter its route to avoid unnecessary resource consumption or exposing the vehicle's position to potential adversaries.

To execute the WNN heuristic on-line, let $P = \{S, i_1, i_2, \ldots, i_M\}$ represent the M task path the controller has already taken with the UUV located at task i_M. The controller runs the following:

```
Begin:
    $P = \{S, i_1, i_2, \ldots, i_M\}$.
    loop
        Find all feasible controls from the last element in $P$, call it $L$, at state $x_k$.
        Calculate $f(L, j)$ for all feasible controls $j$.
        Find $P_{MIN}$, the control that produces the minimum $f(L, j)$.
        $P = P \cap P_{MIN}$.
    loop until There are no feasible controls.
    $P = P \cap E$.
End
```

As mentioned in Section 4.3.1, the controller can generate multiple trajectories via simulation to approximate the WNN solution. The controller may have more or less time to compute the route on-line, so the appropriate number of trajectories depend on the mission situation.

Rollout Algorithm

Since the Rollout Algorithm requires multiple calculations of the base policy (in this case, the WNN), the algorithm takes longer to run than the heuristic itself. Despite this, due to the sequentially improving property the RA determines a better task route through the mission environment than the base policy alone.

To execute the RA in an on-line setting, the controller runs the following:

Begin:

$P = \{S, i_1, i_2, \ldots, i_M\}$.

Find all feasible controls from the last element in P, i_M.

Calculate $\tilde{J}_k(x_k)$ and $\tilde{J}_k(y_k)$ for all feasible controls using the base heuristic.

Find the control, P_{MAX}, that produces the maximum expected one-stage DP cost plus estimated remaining cost-to-go formed by route U.

$P = P \cap P_{MAX} \cap U$.

End

Even when problem parameters do not change, there is a possibility that the current route becomes suboptimal due to unexpected disturbances. The Hybrid Algorithm presents a way for the controller to update an *a priori* route solution on-line.

Hybrid Algorithm

Secomandi [14] motivates the development of the Hybrid Algorithm (HA), which focuses on computing a reoptimization-type routing policy for the single vehicle routing problem with stochastic demands. The HA addresses situations where the controller encounters an unlikely set of disturbances during its mission and a better route may be possible given its current state information. For example, if the UUV completes the first task ahead of schedule, the controller may have time to complete additional tasks; on the other hand, if the UUV is behind schedule, then there may be doubt whether the given sequence can be completed within the allowable time and the controller should choose an alternate route. Let:

- $U = \{S, i_1, i_2, \ldots, i_M, E\}$ represent the M task *a priori* UUV route which attains the expected reward Q_U; U can be obtained by any algorithm presented in this thesis.

- $R(x_k)$ represent running the Rollout Algorithm at state x_k, which returns the route P generating expected reward Q_P.

```
Begin:
    Given $U = \{S, i_1, i_2, \ldots, i_M, E\}$ and $Q_U$.
    loop
        Execute first control in $U$ and arrive at state $x_k$.
        At state $x_k$, UUV has remaining task route $V$ with expected reward $Q_V$.
        Calculate $R(x_k)$ to obtain $P$ and $Q_P$.
        If Pr{mission fail with $V \mid x_k$} $> 1 - \beta$
            $U = P$, i.e. continue with new route computed via $R(x_k)$.
            $Q_U = Q_P$.
        Else If $Q_V < Q_P$
            $U = P$, i.e. continue with new route computed via $R(x_k)$.
            $Q_U = Q_P$.
        Else
            $U = V$, i.e. continue with original route.
            $Q_U = Q_V$.
        End
    loop until UUV reaches end mission location.
```

After the completion of each task, the HA runs the Rollout Algorithm with the updated state information and checks two items:

1. If the current route can be completed with probability β.
2. If the current route produces a higher objective function value than the route determined via the Rollout Algorithm.

If both conditions are met, then the UUV maintains its current trajectory; otherwise, the controller alters its route according to the Rollout Algorithm solution.

The Hybrid Algorithm effectively ties together the *a priori* and on-line methods by using the power of information to update the task route. The controller has less concern for computation time before the mission sortie, so a more exhaustive algorithm is acceptable. Once on-line, however, the controller requires heightened computation speeds to produce a "good" solution.

4.4.2 On-line Planning

The controller may not be able to develop an *a priori* task route for a variety of reasons; for example, when:

- The UUV must be launched immediately in response to a terrorist attack.
- The controller is not confident enough in future problem parameters.
- The mission is too complex to develop the full task route.

In these situations, on-line planning is a viable approach to solve the UUVMPP.

The key concept with on-line planning is that the controller follows a routing **policy** instead of a task route. More precisely, the controller uses its current state information during the mission to find the next optimal control with some estimate of the remaining cost-to-go; however, the controller is not immediately concerned with the future task sequence. Due to the limited time available to the UUV during the mission, approximate methods are the preferred approach for on-line planning. In particular, the Rollout Algorithm with Weighted Nearest Neighbor as its base policy provides high quality solutions and maintains a relatively low run time.

Let:

- P represent the final path formed via the on-line planning algorithm where $P = \{S, i_0, i_1, \ldots, i_M, E\}$.

- $R(x_k)$ represent running the Rollout Algorithm at state x_k, which returns the route Q with first control q.

A pseudocode description of how the controller uses the Rollout Algorithm as an on-line planning algorithm follows:

```
Begin:
    $P = \{S\}$ and $x_k$ is the start mission location at time 0.
    loop
        Find all feasible controls from the last element in $P$ at state $x_k$.
        Calculate $R(x_k)$ to obtain route $Q$ with first control $q$.
        Apply control $q$, execute task, and arrive at new state $x_k^*$.
        $P = P \cap q$.
        $x_k = x_k^*$, i.e. update algorithm with new state information.
    loop until There are no feasible controls.
    $P = P \cap E$.
End
```

On-line planning is similar to the Hybrid Algorithm described in Section 4.4.1 except the controller does not have access to its future mission plan. With on-line planning, after the controller ensures there are feasible controls in the next stage (*i.e* at least one task can be completed that allows the UUV to reach the end mission location within mission constraints) the controller selects the most profitable task determined by running the Rollout Algorithm with its current state information.

4.5 Experimental Setup

4.5.1 Problem Assumptions

The assumptions discussed in Section 3.5 also hold in this chapter, which include:

- Travel times and task execution times model accumulated navigation error as having a linear relationship with time; the formulations do not rely on the linearity assumption to solve the UUVMPP.
- *a priori* selection of navigation fixes.
- Constant discretization by parameter T, the controller's choice of time step.

4.5.2 Sample Problem

This section illustrates how the DP algorithms can be used to find a task route solution in a realistic three-task scenario. Figure 4-6 contains a snapshot of the mission environment and Table 4.2 displays the mission and task parameters. For the Sample Problem, $r_i = 1$, $i \in \{1, 2, 3\}$ and the Linear Decrease reward function is used, where the objective value contribution of each task depends on the amount of accumulated navigation error. This means the stage cost in Equation 4.7 is:

$$g(y_k, x_k) = \frac{N_{MAX} - v}{N_{MAX}} \qquad \forall k, y_k \in Y_k, x_k \in X_k, i \in \{1, 2, 3\}$$

where v represents the amount of accumulated navigation error when the UUV arrives at task i.

A Priori – Brute Force Method

The Brute Force method returns an objective function value of 2.5688 where the optimal first control is $\bar{2}$. As mentioned previously, the BF method provides the optimal policy for every potential situation the controller may encounter during its mission. Since the complete

(a) Mission Parameters

# of tasks	3
# of navigation points	2
# of navigation areas	0
# of avoidance zones	0
T_{MAX}	21 time steps
N_{MAX}	11 time steps
T	3,600 seconds
Objective function	Linear Decrease

(b) Task Parameters

Task #	Mean execution	Fix?
1	7,200 seconds	no
2	7,200 seconds	no

Table 4.2: Three-task Sample Problem Parameters

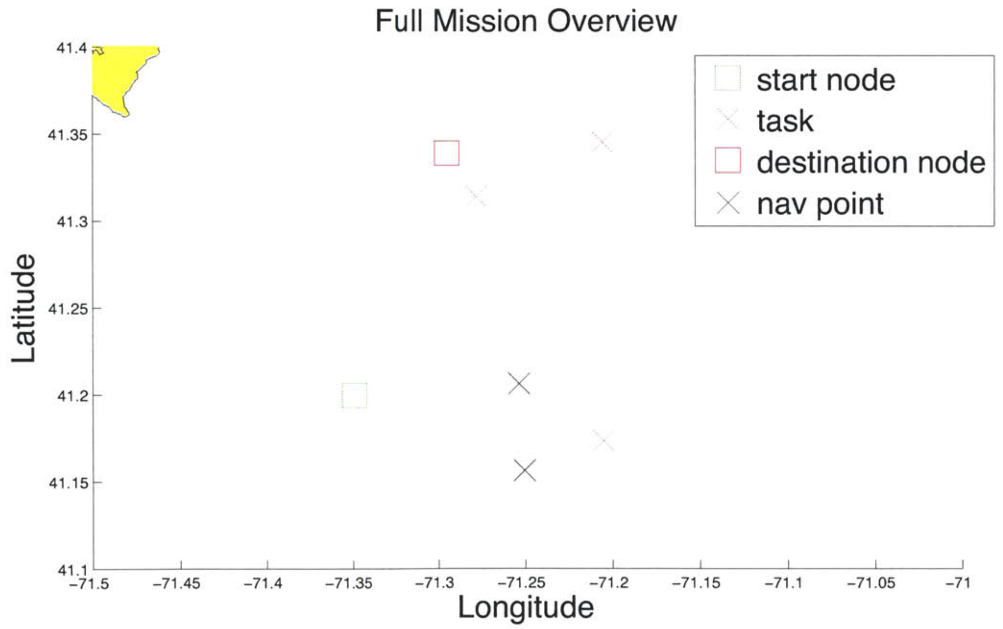


Figure 4-6: Three-Task Sample Problem Map Illustration

Mission time (sec)	Accumulated Navigation Error (m)	Optimal Control
$T \leq 18000$	Any	$\bar{1}$
$18000 < T \leq 36000$	Any	$\bar{3}$
$36000 < T \leq 46800$	Any	$\bar{1}$
$T = 46800$	$N \leq 14400$	$\dot{E}$
$T = 46800$	$N > 14400$	$\bar{E}$
$T > 46800$	Any	$\bar{E}$

Table 4.3: Brute Force Sample Problem Solution – After completion of Task 2. Note how the optimal control depends on the elapsed time and accumulated navigation error (*i.e.* the controller's information state).

solution is the length of the total number of states in this system, 10,628 states, Table 4.3 portrays the subset of solutions after the controller completes task 2 (viewed as thresholds depending on the time and accumulated navigation error constraints).

Through the analysis of the result, the BF method appears to produce a logical solution. The optimal first control is to maneuver to task 2, which is the closest task to the start mission location. From task 2, if the UUV has enough remaining time and has a chance to complete both remaining tasks, then the controller chooses task 1 since path $\bar{2} \to \bar{1} \to \dot{3} \to E$ takes less expected time than path $\bar{2} \to \bar{3} \to \dot{1} \to E$. However, if the UUV only has the resources to complete one of the two remaining tasks, the controller weighs the two options and selects the task which maximizes its expected reward while still maintaining a confidence level of β. If the UUV does not have enough available time, then the controller transitions directly to the end mission location.

A Priori – Weighted Nearest Neighbor

To illustrate the performance of the WNN heuristic, the Sample Problem was solved by using the Monte Carlo method to generate 200 trajectories (*i.e.* WNN solutions). Table 4.4 provides an overview of the algorithm's solution quality, including objective function value

	Objective Function Value	**Associated Route**
Worst Case Route	1.8	$S \rightarrow \dot{2} \rightarrow \bar{3} \rightarrow \dot{1} \rightarrow E$
10% Best Route	1.9	$S \rightarrow \dot{2} \rightarrow \bar{1} \rightarrow \dot{3} \rightarrow E$
50% Best Route	2.1	$S \rightarrow \dot{2} \rightarrow \bar{1} \rightarrow \bar{3} \rightarrow E$
90% Best Route	2.4	$S \rightarrow \bar{2} \rightarrow \bar{1} \rightarrow \bar{3} \rightarrow E$
Best Case Route	2.5	$S \rightarrow \bar{2} \rightarrow \bar{1} \rightarrow \bar{3} \rightarrow E$

Table 4.4: Weighted Nearest Neighbor Sample Problem Solution

and the route associated with that value. The task sequence given by the 90% worst case solution — $S \rightarrow \dot{2} \rightarrow \bar{1} \rightarrow \dot{3} \rightarrow E$ — attains an objective function value of 1.9. A high percentage of the solutions starts with executing tasks 2 and 1, and then only completing task 3 if there is enough remaining time prior to T_{MAX}. While not nearly as robust of a solution as the BF method, the WNN algorithm produces a similar task route solution at a fraction of the computation time.

A Priori – Rollout Algorithm

The *a priori* Rollout Algorithm enhances the performance of the WNN algorithm by solving a one-stage DP maximization problem, which reduces the length of the horizon estimated by the base policy. Since there are three tasks in this scenario, the RA maximizes over the six feasible initial controls plus the remaining cost-to-go; see Table 4.5. As a result, the RA selects the control that maximizes the total expected reward, in this case $\dot{2}$ which yields an expected objective function value of 2.1082, and follows the route returned by the base policy. The strict dominance of the *a priori* RA solution compared to the WNN 50% and 90% trajectory solutions in Table 4.4 supports the sequentially improving property.

1st Control	1st Stage Reward	Estimated Cost-To-Go	Total Reward
$\dot{1}$	0.5357	1.1359	1.6716
$\bar{1}$	0.6	0.4929	1.0929
$\dot{2}$	0.7309	1.3773	2.1082
$\bar{2}$	0.6	1.3	1.9
$\dot{3}$	0.6637	1.1378	1.8015
$\bar{3}$	0.6	0.3302	0.9302

Table 4.5: Rollout Algorithm A Priori Sample Problem Solution. "1st Stage Reward" represents the expected reward for completing the control in the first column, "Estimated Cost-To-Go" represents the expected remaining reward obtained via the WNN given the control in the first column is applied, and "Total Reward" is the sum of columns 2 and 3.

Iteration Number	Route	Reward for this Iteration
1	$S \to \dot{2} \to \bar{1} \to \bar{3} \to E$	2.1
2	$S \to \dot{2} \to \bar{1} \to \bar{3} \to E$	2.3
3	$S \to \dot{2} \to \bar{1} \to \bar{3} \to E$	2.1
4	$S \to \dot{2} \to \bar{1} \to \bar{3} \to E$	2.1
5	$S \to \dot{2} \to \bar{1} \to \bar{3} \to E$	2.0
mean		2.12

Table 4.6: Rollout Algorithm On-line Sample Problem Solution. Note how each iteration produced the same task route solution with minor variation in the reward the controller realized during the mission.

On-line – Rollout Algorithm

If the controller does not have access to an *a priori* task route, the controller can use the Rollout Algorithm as an on-line routing policy. To evaluate the performance of the RA as an on-line planner, a testbed was built in MATLAB to simulate a UUV's movement through the mission environment represented by the Sample Problem. Due to the stochastic mission parameters, the controller's route may vary for different iterations of the same scenario; therefore, Table 4.6 displays the task route and associated objective function value attained with five iterations of the simulation. Note how each iteration produced the same task route solution with minor variation in the reward the controller realized during the mission.

# of tasks	variable
# of navigation points	2
# of navigation areas	0
# of avoidance zones	0
T_{MAX}	21 time steps
N_{MAX}	11 time steps
T	3,600 seconds
Objective function	Linear Decrease

Table 4.7: Chapter 4 Simulations Parameters

4.6 Results

The results section considers diverse mission scenarios to demonstrate the performance of the DP algorithms in both an *a priori* and on-line setting. Table 4.7 contains the operational parameters for the simulations presented in this chapter. The algorithms are tested with the same 100 randomly generated scenarios inside the Naragansett Bay operating area for each n as in Chapter 3. The algorithms were built in MATLAB and tested on a Quad Core Intel Xeon E5687 with 3.60GHz, 6.4 GT/s, and 48GB of RAM.

4.6.1 A Priori Optimization

Run Time Analysis

Before the UUV begins its operation, the controller typically has ample time to develop a robust task route solution. The Brute Force method enables the controller to determine the optimal decision given any potential situation the vehicle could encounter. Figure 4-7 displays the run time for the Brute Force method on a logarithmic y-axis; note the rapidly increasing computation speed necessary for relatively small mission scenarios, where the values for $n \geq 5$ are given as theoretical values.

The Weighted Nearest Neighbor algorithm quickly finds a task route solution by selecting the control that creates the most immediate gain to the mission's objective. Due to the stochasticity in the travel times and task execution time, the Monte Carlo technique sends

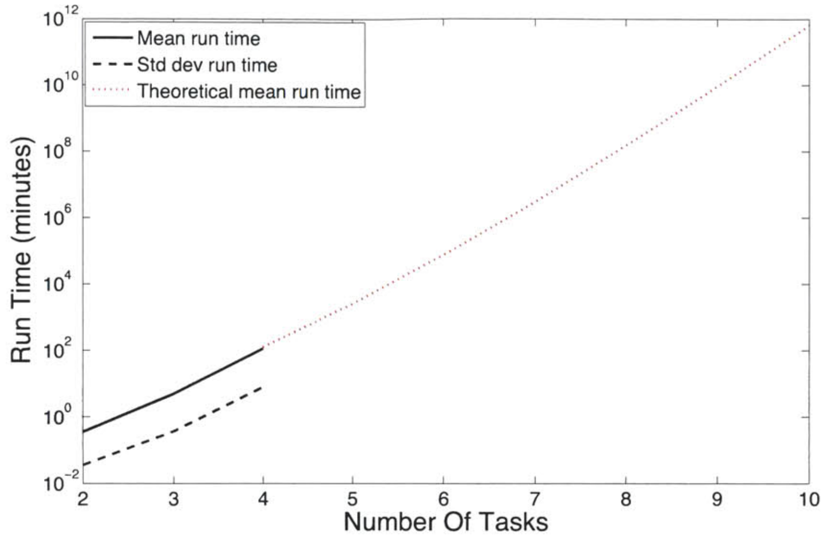


Figure 4-7: Run Time Results – Brute Force method. The values for $n \geq 5$ are given as theoretical values.

multiple trajectories through the Stochastic Shortest Path model to approximate the WNN solution. The WNN algorithm's run time is a function of n, T, and the number of trajectories the controller chooses for the given scenario. Depending on the time available, the controller can adjust the number of trajectories accordingly. Throughout this section, the analysis shows the performance of the algorithm with 200 trajectories[3].

The run time of the Rollout Algorithm depends on n, T, and the run time of the base policy (*e.g.* WNN). In the UUVMPP, the RA finds all feasible controls from the start mission location and runs the base policy given the potential completion of each feasible control. Therefore, RA must execute the base policy $2 * n$ times – once for each task **with** a navigation fix and once for each task **without** a navigation fix. As a result, the RA's run time is $O(2 * n * B(n))$ where $B(n)$ represents the time required to execute the base policy as a function of n. Table 4.8 displays the mean speed up of the WNN and RA compared to the Brute Force method using Amdahl's Law [2].

While the Brute Force method's run time becomes intractable after a small number of tasks, the Weighted Nearest Neighbor and Rollout Algorithms scale much better with

[3]See Secomandi [13] for the rationale behind the choice of 200 trajectories.

Number Of Tasks	% Speed Up – WNN	% Speed Up – RA
2	6.97×10^1	1.24×10^1
3	4.51×10^2	4.02×10^1
4	5.80×10^3	3.46×10^2

Table 4.8: Speed Up of Approximate DP Algorithms. There is a significant improvement in run times compared to the Brute Force method. The RA exhibits the $O(2 * n * B(n))$ complexity, where $B(n)$ represents the time required to execute the WNN.

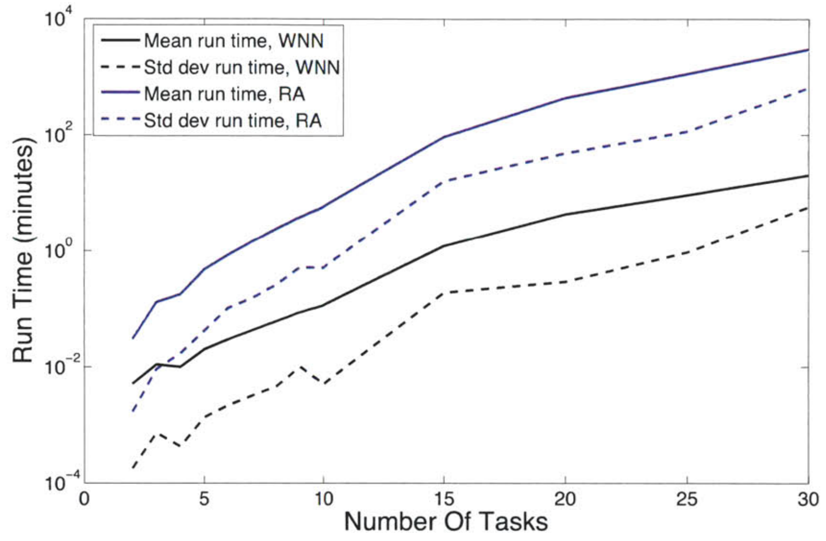


Figure 4-8: Run Time Results – Approximate DP Algorithms, including Weighted Nearest Neighbor and Rollout Algorithm.

increased complexity. Figure 4-8 presents the mean and standard deviation of run times using WNN and RA on a logarithmic y-axis. For $n = \{20, 25, 30\}$, the number of trajectories was reduced from 200 to 10 to allow for computation of all 100 scenarios. The values in Figure 4-8 for these values of n are scaled to represent theoretical run time for 200 trajectories. Both approximate DP methods remain computationally feasible for much larger problems; the largest scenarios tested in this thesis involve 30 tasks.

Characterization of Optimal Solution

Table 4.9 and Figure 4-9 show the quality of the *a priori* solution generated by the Weighted Nearest Neighbor and Rollout Algorithms compared to the Brute Force method; *e.g.* a value

	% Optimal Solution		
# Of Tasks	**WNN 90%**	**WNN 50%**	**RA**
2	0.92	0.99	1.02
3	0.91	0.99	1.03
4	0.88	0.97	1.02
mean	0.91	0.98	1.02

Table 4.9: Mean Optimal Solution Degradation – Approximate DP vs. BF. Figure 4-9 displays the results of this table.

of 0.92 means the approximate DP algorithm produces an *a priori* task route that attains an objective function value on average 8% less than the Brute Force method. Table 4.9 and Figure 4-9 illustrate the following three performance measures:

1. Weighted Nearest Neighbor solution – mean of 200 trajectories.

2. Weighted Nearest Neighbor solution – 90% worst case trajectory.

3. Rollout Algorithm solution (using Weighted Nearest Neighbor as base policy).

For simulations involving less than five tasks, the Rollout Algorithm provides an *a priori* task route that achieves an objective function value on average 2% greater than the Brute Force method. By comparison, the Weighted Nearest Neighbor attains a value on average 2% or 9% less than the Brute Force method, depending on which trajectory solution the controller chooses.

The Rollout Algorithm produces a solution greater than the Brute Force method because the routes found with RA are not guaranteed to reach the end mission location with sufficient operational resources. The terminal condition with the BF method (Equation 4.11) ensures that all routes have at least a β% chance of successful completion, while the RA method approximates the remaining cost-to-go with the 50% WNN trajectory solution (*i.e.* at least 50% of trajectories reach the end mission location). With on-line re-planning the controller can verify the solution at the conclusion of each stage to ensure the UUV reaches the end location within mission constraints, which mitigates the concern of having an overly-aggressive task route solution.

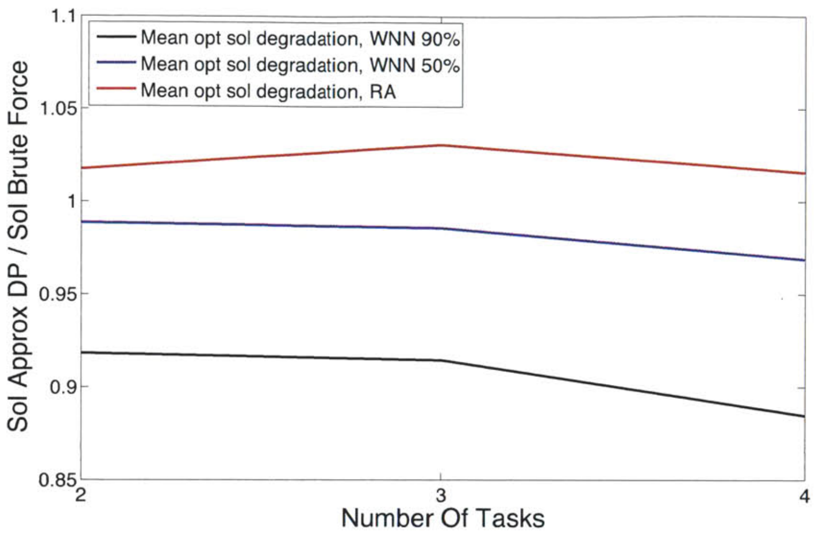


Figure 4-9: Mean Optimal Solution Degradation – Approximate DP vs. BF (illustration of Table 4.9). The figure compares the performance of Weighted Nearest Neighbor (50% and 90%) and the Rollout Algorithm to the Brute Force method.

Figure 4-10 displays the performance of the Rollout Algorithm compared to the Weighted Nearest Neighbor for scenarios up to 30 tasks. Note how the RA produces a solution greater than or equal to the WNN, demonstrating the sequentially improving property.

4.6.2 Comparison to Network Optimization Methods

Run Time Analysis

The main benefit of the Rollout Algorithm (with the Weighted Nearest Neighbor as its base policy) is the significant reduction in computation speed compared to the Network Optimization methods in Chapter 3. Figure 4-11 shows the mean speed up of RA versus MIP Formulation 1 with the exact algorithm and the ϵ-Rounding Heuristic on a logarithmic y-axis, where the values for $n \geq 8$ are given as theoretical values. For example, with scenarios involving seven tasks the Rollout Algorithm computes an *a priori* task route 308 and 134 times faster on average than the exact algorithm and heuristic, respectively. The run time improvements are even more valuable when the controller must re-plan on-line and quickly

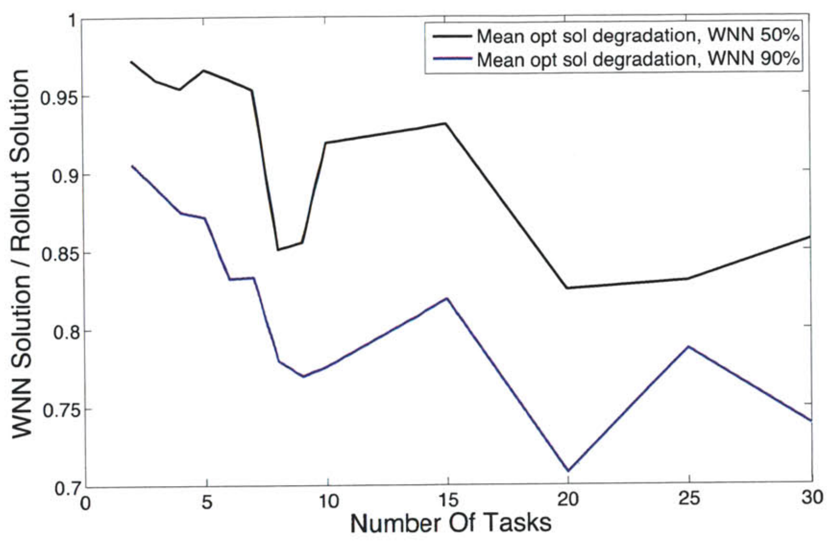


Figure 4-10: Mean Optimal Solution Degradation – Rollout Algorithm vs. Weighted Nearest Neighbor. Note how the RA always produces a solution greater than WNN, demonstrating the sequentially improving property.

develop a new task route.

Characterization of Optimal Solution

Figure 4-12 illustrates the degradation in solution quality with the Brute Force method compared to the MIP Formulation 1 exact algorithm[4]. On average, the BF *a priori* task route attains an objective function value approximately 5% below the task route with MIP Formulation 1.

The way each exact algorithm protects from exceeding operational constraints causes the optimality gap. With the Brute Force method, the algorithm sets the remaining cost-to-go to $-\infty$ for all states where there is a probability $\geq 1 - \beta$ the end node will not be reached **in the final stage** with sufficient mission resources. On the other hand, the β constraint in MIP Formulation 1 builds a confidence factor **throughout the duration of the mission**; specifically, the β constraint allows the mission planner to develop a route that maintains

[4]The performance of the DP methods are not compared to the ϵ-Rounding Heuristic. As shown in Chapter 3, the ϵ-Rounding Heuristic calculates solutions similar in quality to the exact algorithm in MIP Formulation 1.

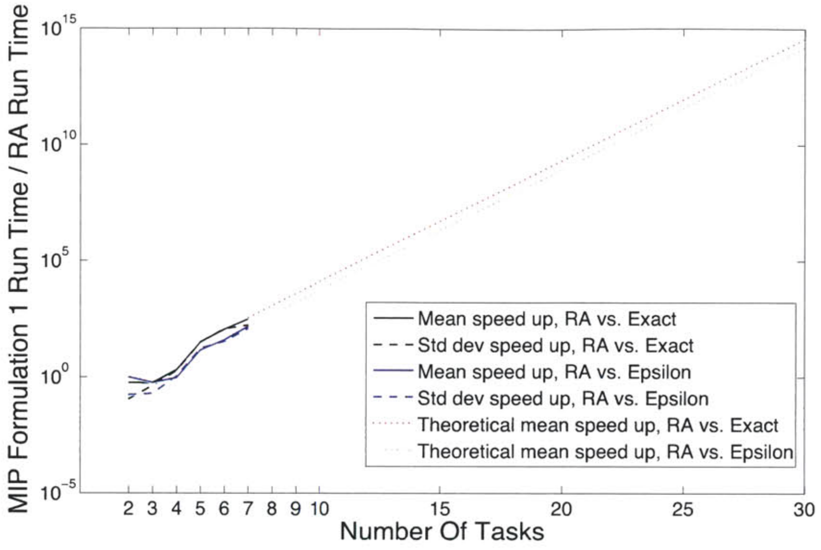


Figure 4-11: % Speed Up – Rollout Algorithm vs. MIP Formulation 1. With scenarios involving seven tasks, the Rollout Algorithm computes an *a priori* task route 308 and 134 times faster on average than the exact algorithm and heuristic, respectively. Note that the values for $n \geq 8$ are given as theoretical values.

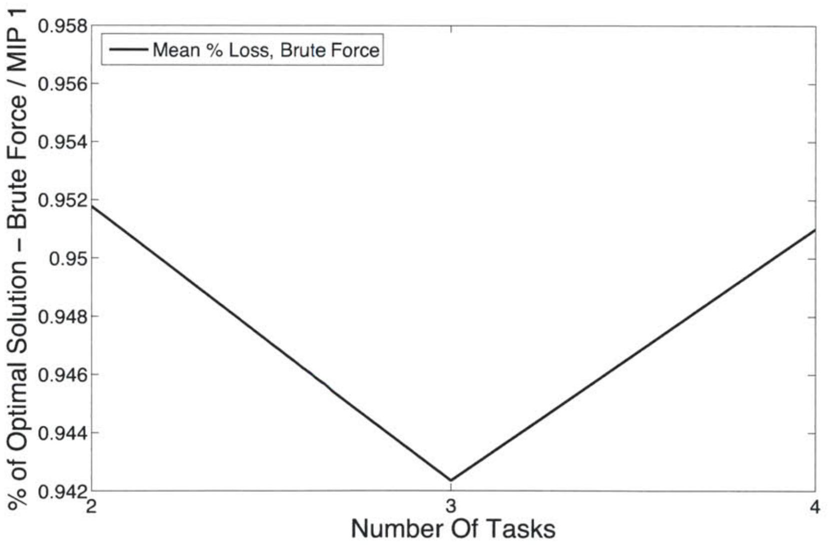


Figure 4-12: Mean Optimal Solution Degradation – Brute Force method. On average, the Brute Force *a priori* task route attains an objective function value approximately 5% below the task route with MIP Formulation 1.

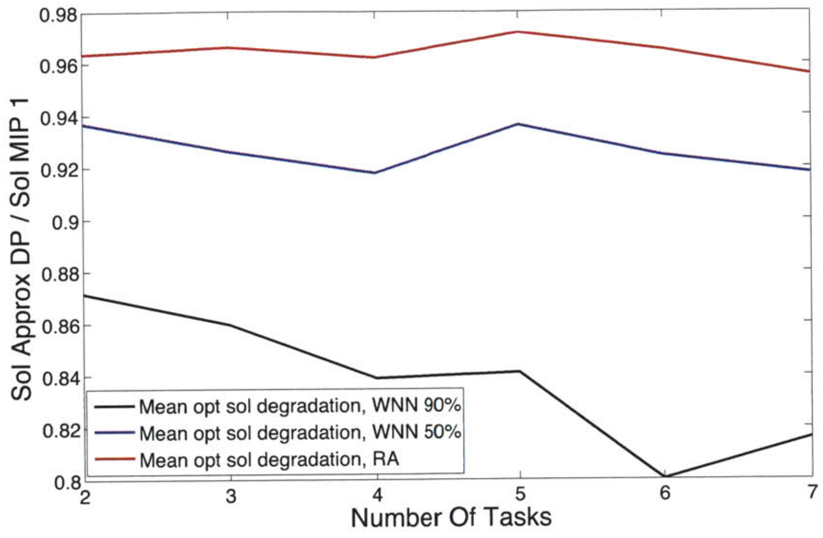


Figure 4-13: A Priori Mean Optimal Solution Degradation – Approximate DP vs. MIP Formulation 1. The RA develops an *a priori* task route that attains an objective function value on average 4% less than MIP Formulation 1, while the WNN produces a value on average 8% and 16% below exact methods.

an appropriate confidence level for satisfying operational constraints. As a result, the Brute Force method creates a more conservative mission plan.

Figure 4-13 compares the objective function value for the approximate DP algorithms to the exact algorithm with MIP Formulation 1. With scenarios involving seven tasks, the RA develops an *a priori* task route that attains an objective function value on average 4% less than the exact algorithm with MIP Formulation 1. The WNN, on the other hand, produces a value on average 8% and 16% less than MIP Formulation 1, depending on whether the controller chooses the 50% or 90% trajectory solution. Further, the WNN exhibits a decrease in solution quality as the number of tasks increases.

Although the routes calculated with RA are more aggressive than the solutions with MIP Formulation 1 (due to the β constraint), the controller can use the Hybrid Algorithm on-line to verify the route reaches the end mission location within operational constraints. For a significant speed up in run time, the level of degradation in the optimal solution with the Rollout Algorithm (with WNN as its base policy) may be practical to develop an *a priori* task route.

4.6.3 On-line Optimization

On-line Re-planning

When problem parameters change, the ability for the controller to know when and how to re-plan on-line is critical in the UUVMPP. Due to operational limitations during the mission (*e.g.* time and energy), finding a "good" result quickly often trumps determining the new optimal route; therefore, the Weighted Nearest Neighbor and Rollout Algorithms are preferred to exact methods. The run times for the two algorithms are of the same order as their *a priori* counterparts with similar parameters (*i.e.* number of tasks n and number of time steps T).

Section 4.4.1 presents the Hybrid Algorithm (HA), an on-line re-planning method that takes advantage of its current state information to update an *a priori* task route via the Rollout Algorithm. After the completion of each task, the HA verifies that the current route can be completed with at least probability β and there is no better task sequence available. If both of these conditions are met, then the controller continues with the original task route; if either condition fails, then the controller re-plans and maneuvers to the task location identified by the Rollout Algorithm.

It is unclear how frequently the HA would update the *a priori* solution on-line given various situations; for example,

- If best case disturbances are experienced.
- If worst case disturbances are experienced.
- If task locations become unavailable during the mission.
- If a new task becomes available during the mission.
- If task execution times or travel times change.

- If avoidance zones are realized during the mission.

A testing platform with these capabilities is left for future work.

On-line Planning

When the controller does not have access to an *a priori* task route, the controller can use the Rollout Algorithm to determine a routing policy throughout the UUV's mission. To asses the performance of the RA as an on-line planner, a testbed was built in MATLAB to simulate a UUV's movement through the mission environment with the same 100 randomly generated scenarios for each n. The run time of the on-line planning algorithm is the same as the run time of the Rollout Algorithm in an *a priori* setting with similar parameters (*i.e.* number of tasks n and number of time steps T).

Letting P represent the final path formed via the on-line planning algorithm, the following procedure simulates one possible plan (*i.e.* iteration) from S to E through a given mission environment with the RA in an on-line setting:

```
Begin:
    $P = \{S\}$ and $x_k$ is the start mission location at time 0.
    loop
        Find all feasible controls from the last element in $P$ at state $x_k$.
        Calculate $R(x_k)$ to obtain route $Q$ with first control $q$.
        Simulate the controller's state information after applying control $q$, call it $x_k^*$.
        $P = P \cap q$.
        $x_k = x_k^*$, i.e. update algorithm with new state information.
    loop until There are no feasible controls.
    $P = P \cap E$.
```

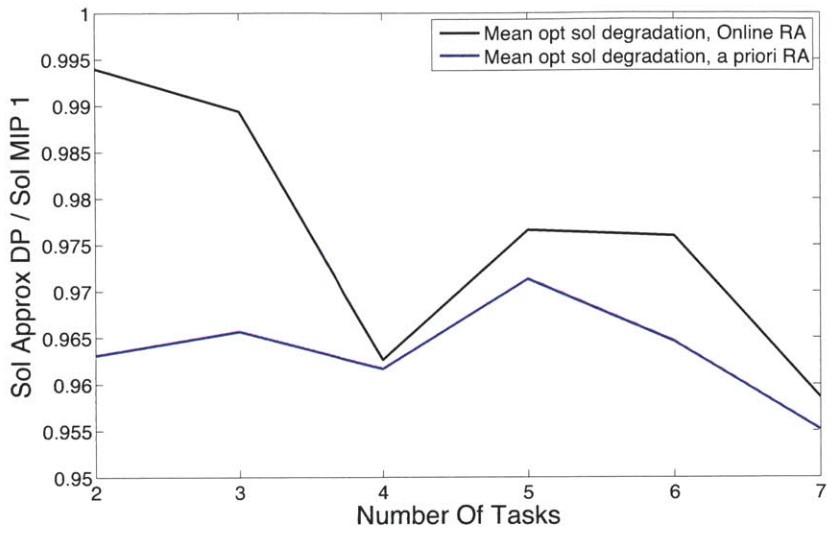


Figure 4-14: Mean Optimal Solution Degradation – RA vs. MIP Formulation 1. On average, the on-line RA finds routes through the mission environment that achieve an objective function value 2% below exact algorithms, a 2% improvement over the RA's performance as an *a priori* route planner.

End

The above procedure returns the reward attained by the controller from the start location to the end location for one iteration of a given scenario. Due to the stochastic mission parameters, the controller's route and subsequent objective function value may vary for different iterations of the same scenario; therefore, multiple iterations of each scenario must be run to gauge the distribution of the performance of the on-line planning algorithm.

Figure 4-14 illustrates the solution quality of the RA as an *a priori* and on-line planner versus the objective function value obtained with *a priori* task routes found with MIP 1. For the on-line RA planning algorithm, the results show the mean reward attained from the task routes generated from 10 iterations of the same scenario. On average, the on-line RA finds routes through the mission environment that achieve an objective function value 2% below exact algorithms, a 2% improvement over the RA's performance as an *a priori* route planner.

Since the on-line planning algorithm determines which controls are feasible at the conclusion of each stage, the UUV is guaranteed to reach the end mission location within operational constraints (*i.e.* without exceeding T_{MAX} and N_{MAX}) given accurate mission inputs. The algorithm reduces the risk of being too conservative or too aggressive in determining a path through the mission environment since it uses the controller's updated state information to choose subsequent controls. With a small degradation in objective function values and tractable run times, the Rollout Algorithm as an on-line planner serves as a reasonable approach to solve the UUVMPP.

4.7 Summary

Since the UUVMPP can be decomposed into a series of stages, the controller can use Dynamic Programming to find a task route through a stochastic mission environment.

The **Brute Force method**, an exact Dynamic Program, solves the UUVMPP *a priori* by calculating the optimal policy given any potential situation the controller could encounter. However, the method quickly becomes computationally infeasible.

The **Weighted Nearest Neighbor**, a greedy algorithm found in various Prize Collecting Traveling Salesman problems, determines which control provides the most immediate benefit with an appropriate choice of distance measure. Although WNN significantly improves the run time to develop an *a priori* or on-line task route, the algorithm demonstrates deteriorated performance as the number of tasks increases.

The **Rollout Algorithm**, an approximate Dynamic Program, uses a limited look-ahead scheme to sequentially improve a base policy, in this case the WNN, to calculate an *a priori* or on-line solution. Since the RA reduces the number of stages approximated by the base policy, the algorithm finds higher quality solutions than the heuristic alone. The RA maintains computation tractability for complex mission scenarios.

The **Hybrid Algorithm** shows how an *a priori* solution can be updated on-line by executing the Rollout Algorithm with the controller's current state information. For larger problems (*i.e.* $n \geq 8$), the HA eliminates some of the concern of using a greedy heuristic as an *a priori* optimization tool because the controller checks, at the conclusion of every stage, for higher quality solutions that reach the end mission location within operational constraints.

If the controller cannot determine an *a priori* task route prior to the start of the mission, using the RA as an **on-line planning** tool is a feasible way to solve the UUVMPP. With on-line planning, the controller follows a routing policy; *i.e.* the controller uses its state information to decide which control is optimal in the next stage without concern for the future task sequence. The results show that the on-line planning algorithm produces task route solutions that perform well in a variety of situations at a fraction of the computation time of exact methods.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

Conclusion

The research proposed various formulations to solve the Unmanned Underwater Mission Planning Problem (UUVMPP), where an Unmanned Underwater Vehicle (UUV) must choose which tasks to perform (or not perform) in which sequence to maximize or minimize some mission objective. The thesis sought to identify task route solutions that performed well in a stochastic mission environment with constraints on time, energy, and navigation error. The work tested the formulations with diverse, realistic mission scenarios with increased computational complexity.

5.1 Outline

Chapter 1 introduced the UUVMPP and provided motivation for this area of research. The chapter also presented a thesis outline and relevant terminology and notation found in the thesis.

Chapter 2 outlined pertinent mathematical research topics which included a discussion of relevant literature. In particular, the chapter defined the UUVMPP as a Prize Collecting Traveling Salesman Problem.

Chapter 3 showed how the UUVMPP can be solved with Network Optimization methods. The chapter developed various network flow models to add layers of complexity to the mission planning problem, culminating with the Time and Navigation Expanded Stochastic Graph (TANESG). With the TANESG, two formulations were established:

- **MIP Formulation 1** determined the task route *a priori* that provides the greatest expected utility and maintains a certain probability of successfully completing the mission. Further, the ϵ-Rounding Heuristic was found to improve the run time by a factor of 2.98 over the exact algorithm while achieving a 2% degradation in optimal solution quality.
- **MIP Formulation 2** extended MIP Formulation 1 by allowing the UUV to skip tasks or end the mission early if the UUV experiences worst-case disturbances.

MIP Formulation 1 is an extension of Cates [11] Formulation 4.3.7; although the formulation developed by Cates experienced better run times, it does not incorporate navigation fixes into the mission planner. Cates proposed a way to add flow variables to the Time Expanded Stochastic Graph to account for navigation error that would maintain a similar order of computational complexity, but this would not allow the mission planner to explicitly model situations where navigation error affects the UUV's operations (*e.g.* if reduced positional certainty causes the UUV to complete a task at a slower rate). Although time and accumulated navigation error impacts on the UUV's mission in most situations, it is thought that the expansion into the time and navigation error dimension (*i.e.* the TANESG) is not always required. If the mission planner expands the network flow model in only one dimension (*i.e.* either time or navigation error), then MIP Formulation 1 would likely experience similar computational complexity as Formulation 4.3.7 in Cates.

Chapter 4 modeled the UUVMPP as a Stochastic Shortest Path problem and presented various Dynamic Programming (DP) solving algorithms, which included the following:

- The **Brute Force** method, an exact Dynamic Program, provided the ideal solution to the UUVMPP by returning the optimal decision at every situation the controller can potentially encounter. However, the algorithm experiences the "curse of dimensionality" and quickly becomes intractable.

- The **Weight Nearest Neighbor** (WNN), a greedy heuristic, was shown to handle diverse mission scenarios while maintaining extremely low run times.

- The **Rollout Algorithm** (RA), an approximate Dynamic Program, solved a one-stage DP algorithm that used WNN to estimate the remaining cost-to-go.

The Rollout Algorithm provided an *a priori* solution that performs on average 4% worse than the exact algorithm with MIP Formulation 1. With a significant speed up compared to the exact algorithm and the ϵ-Rounding Heuristic, the reduction in performance is acceptable particularly for more complex mission scenarios where solutions are not yet realized with network optimization methods.

When problem parameters change, the controller requires the ability to quickly re-plan on-line; further, the controller may encounter low-probability states and a better route may exist. The **Hybrid Algorithm** (HA) presented a way to improve the *a priori* solution on-line by utilizing the Rollout Algorithm with the controller's updated state information. It is thought that by using the power of information, the Hybrid Algorithm mitigates some of the concern of developing a lower quality or overly-aggressive *a priori* solution with approximate methods.

Using the Rollout Algorithm as an **on-line planning** tool, the controller does not need an *a priori* task route and instead follows a routing policy through the mission environment. By using the controller's updated state information and checking for feasible controls at the conclusion of every stage, the on-line planning algorithm is guaranteed to reach the end mission location within operational constraints. On average, the on-line RA finds routes through the mission environment that achieve an objective function value 2% below the exact algorithm with MIP Formulation 1.

5.2 Summary

Overall, the thesis developed many viable formulations to solve the UUVMPP in both an *a priori* and on-line setting. Before the UUV's mission sortie, it is assumed the mission planner or controller typically has ample time to develop a mission plan; this includes identifying tasks and mission priorities, predicting tide and current data, and many other mission-specific items. If the controller has to plan or re-plan during the mission, however, finding a "good" solution quickly often is necessary as opposed to finding the new optimal route.

MIP Formulation 1, particularly with the ϵ-Rounding Heuristic, produced the best *a priori* solution that maintained a certain confidence level of reaching the end mission location with sufficient operational resources. As the number of tasks n increases, MIP Formulation 1's run time grows exponentially; the largest problems tested in this thesis with the exact algorithm and ϵ-Rounding Heuristic consisted of seven tasks. The Brute Force method, while producing the ideal solution to the UUVMPP, quickly became intractable after problems greater than four tasks. The Rollout Algorithm, using the Weighted Nearest Neighbor as its base policy, produces a high quality *a priori* solution that scales better with increased complexity. Further, the Rollout Algorithm provides exceptional results when optimizing on-line by allowing the controller to use its updated state information to choose subsequent controls. Approximate DP algorithms were run with scenarios up to 30 tasks in duration.

Therefore, the formulations found in this thesis should be considered for application onboard UUVs.

5.3 Future Work

5.3.1 UUVMPP Extensions

Multiple UUVs

This work solved the UUVMPP as a single vehicle routing problem, where a UUV operates independently with its own individual mission objectives. The UUVMPP complexity significantly increases with multiple UUVs working in tandem to complete tasks, especially since UUVs have limited communication ability once underwater. It is thought the mission-planning algorithm would have to consider approximate methods to maintain computational tractability.

Human Interaction

The formulations in Chapters 3 and 4 find task route solutions autonomously. Before the mission begins, however, human interaction can increase the likelihood of determining the best task route. For example, if the formulations had the ability to return the "five best" solutions and associated objective function value, a human planner can use his/her subject area knowledge and intuition to select the optimal route given the mission situation.

5.3.2 Network Optimization

Protection From Uncertainty

Chapter 3 uses the β constraint to find task routes that maintain a certain confidence level of successful completion. For all simulations, β was set to 0.9 which indicates that the *a priori* solution has at least a 90% chance of reaching the end mission location without exceeding operational limitations. However, sensitivity analysis on β is required to see how this parameter affects the performance of exact and approximate solving methods.

The formulations in Chapter 3 assume that the distribution of travel times and task execution times are fully known before the mission starts (*i.e.* the $\mathbf{p}$ vector in MIP Formulations 1 and 2). If the mission planner desires to model $\mathbf{p}$ as a random variable, the Bertsimas-Sim method of protecting from uncertainty [8] is a viable approach.

Choice of Time Step

The formulations proposed in this thesis depend on discretized time, where all mission parameters were applied with a constant discretization over the duration of the mission. While it is assumed that a finer level of discretization more accurately represents a real world situation, the impact of discretization on the final task solution compared to the continuous model has not been realized. Table 3.7 showed how the choice of time step affects the size of the network flow models.

A time step that adapts with the mission situation could improve the performance of

UUVMPP solving methods. For example, if the controller approaches a portion of the mission where minimal decision-making is necessary (*e.g.* travel between two distant tasks), a coarse resolution could be adequate and reduce computation time. On the other hand, a fine resolution may be more appropriate in situations involving precise routing decisions.

5.3.3 Dynamic Programming

Other Approximate Methods

There are other approximate DP methods that have the potential to improve upon the results found in this thesis. State aggregation groups states together that exhibit certain characteristics to reduce the size of the state space. In the UUVMPP, state aggregation can be used to reduce the level of discretization in the time and accumulated navigation error dimensions. For example, instead of keeping track of the exact amount of accumulated navigation error during a UUV's mission, state aggregation can group states together into three categories:

- Low navigation error – navigation fix not necessary before next task.
- Medium navigation error – navigation fix probable before next task.
- High navigation error – navigation fix required before next task.

Stage aggregation allows the Brute Force method to be applied to problems of higher complexity with reduced solution quality. To further enhance the run times of the method with an aggregated state policy, approximate policy and value iteration techniques can be applied to the reduced-size problem (see [4]). Since the Brute Force method determines the optimal decision given any potential situation the controller encounters, it produces the ideal solution to the UUVMPP and should be considered if computationally feasible.

Rollout Algorithm Improvements

The Rollout Algorithm can sequentially improve any base policy to develop a UUVMPP solution *a priori* or on-line. This thesis considered the Weighted Nearest Neighbor heuristic due to its exceptional computation speeds, but the results show that its performance deteriorates as the number of tasks increases. Feillet, Dejax, and Gendreau [12] discuss other Prize Collecting Traveling Salesman Problems heuristics such as Simulated Annealing and Tabu Search. Other heuristics could potentially perform well in the UUVMPP with heightened computation speeds over exact methods; coupled with the Rollout Algorithm, the task route solutions could approach the quality of the exact solutions at a fraction of the computation speed.

Extending the length of the look-ahead scheme could also enhance the performance of the Rollout Algorithm. Since the solution quality of the Weighted Nearest Neighbor decreased as the number of tasks increased, it is believed that reducing the length of the approximated horizon would find task routes that gencrate a higher expected reward. However, extending the look-ahead scheme increases computational complexity. The Rollout Algorithm with a two- or three-stage look-ahead scheme could yield higher quality solutions at an acceptable increase in computation speed.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix A

Terminology and Notation

- UUV: Unmanned Underwater Vehicle.
- UUVMPP: Unmanned Underwater Vehicle Mission-Planning Problem.
- Route: A sequence of tasks the UUV follows.
- *a priori*: Solving the UUVMPP prior to the start of a mission.
- On-line: Solving the UUVMPP during the mission.
- n: Total number of tasks in a given mission scenario.
- S: Start mission location.
- E: End mission location.
- L: Set of all possible tasks in a given mission scenario, $L = \{1, 2, \ldots, n\}$.
- L_C: During a given mission, set of all completed tasks thus far.
- L_U: During a given mission, set of all uncompleted tasks thus far.
- T: Time step resolution.
- T_{MAX}: Total allowable time in a given mission scenario.

- N_{MAX}: Total accumulated navigation error allowed in a given mission scenario.
- E_{MAX}: Total allowable energy in a given mission scenario.
- G: Decision Graph.
- G_s: Time and Navigation Expanded Stochastic Graph.
- G_a: Route Alteration Graph.
- $\mathbf{y}$: Flow variable on Decision Graph G.
- $\mathbf{x}$: Flow variable on Time and Navigation Expanded Stochastic Graph G_s.
- $\mathbf{z}$: Flow variable on Route Alteration Graph G_a.

Bibliography

[1] Ravindra Ahuja, Thomas Magnanti, and James Orlin. *Network Flows.* Prentice Hall, Inc., Upper Saddle River, New Jersey, 1993.

[2] Gene Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. *AFIPS Conference Proceedings*, (20):483–485, 1967.

[3] Russell Bent and Pascal Van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4):515–530, 2004.

[4] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, Belmont, Massachusetts, third edition, 2005.

[5] Dimitri P. Bertsekas. Rollout algorithms for constrained dynamic programming. April 2005. http://web.mit.edu/dimitrib/www/Rollout_Constrained.pdf.

[6] Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, August 1991.

[7] Dimitri P. Bertsekas, John N. Tsitsiklis, and Cynara Wu. Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 3:245–262, 1997.

[8] Dimitris Bertsimas and Melvin Sim. The price of robustness. *Operations Research*, 52:35–53, 2004.

[9] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, Belmont, Massachusetts, 1997.

[10] Olli Braysy and Michel Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, February 2005.

[11] Jacob Cates. Route optimization under uncertainty for unmanned underwater vehicles. Master's thesis, Massachusetts Institute of Technology, June 2011.

[12] Dominique Feillet, Pierre Dejax, and Michel Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188–205, May 2005.

[13] Nicola Secomandi. A rollout policy for the vehicle routing problem with stochastic demands. *Operations Research*, 49(5):796–802, September-October 2001.

[14] Nicola Secomandi and Francois Margot. Reoptimization approaches for the vehicle-routing problem with stochastic demands. *Operations Research*, 57(1):214–230, January-February 2009.

[15] The navy unmanned undersea vehicle (uuv) master plan, April 2004.